МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

САМАРКАНДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА «ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Дата регистрации:
№_____
2019 г. «___» _____

«Утверждаю»
Проректор СамГУ
по учебной работе
_____проф. А. С. Солеев
«____» _____ 2019 г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО ДИСЦИПЛИНЕ ОСНОВЫ ПРОГРАММИРОВАНИЯ

Часть 1

Для студентов факультета
Прикладной математики и информатики

Сфера знания: 100000-Гуманитарная сфера
Сфера образования: 120000- Гуманитарные дисциплины
Направление обучения: 5130200-Прикладная математика и информатика

Разработчик: **доцент Кобилов С. С.**

Зав. кафедрой: **профессор Жуманов И. И.**

Декан факультета: **доцент Бабаяров А. И.**

САМАРКАНД – 2019

Данный учебно-методический комплекс предназначен для предмета «Основы программирования», который включён в учебную программу бакалавриата направления подготовки «5130200 - Прикладная математика и информатика». Разработано на основе типичной учебной программы, утвержденной Самаркандским государственным университетом.

Учебно-методический комплекс:

Обсужден и утверждён на заседании кафедры от 27 августа 2019 г. (Постановление №1)
Зав. кафедрой: проф. И .Жуманов

Обсужден и утверждён на учебно-методическом совете факультета от 29 августа 2019 г. (Постановление №1).
Председатель совета: доц. Ш. Маматов

Обсужден и утверждён на Учёном совете факультета от 30 августа 2019 г. (Постановление №1).
Председатель совета, декан: доц. А. Бабаяров

Согласовано:
Начальник учебно-методического отдела: Б. Аликулов

Структура учебно-методического комплекса

(Содержание)

1.	Титульный лист……	
2.	Содержание.....	
3.	Силлабус.....	
4.	Учебная программа предмета.....	
5.	Рабочая программа.....	
6.	Текст лекции.....	
7.	Практические занятия.....	
8.	Лабораторные работы.....	
9.	Учебно-методическое обеспечение……	
10.	Глоссарий.....	

Аннотация

Данный учебно-методический комплекс посвящен важнейшему разделу информатики – программированию. Здесь рассматриваются основы классического и современного методов программирования. В качестве языков и систем программирования изучаются языки Object Pascal, Ассемблер и C++ и соответствующие среды программирования.

Для языков приводятся необходимые сведения и примеры, чтобы сложилась общая картина, и стало возможным самостоятельное решение относительно несложных задач по программированию. Эти языки заслуженно является наиболее популярными при традиционном – процедурном – подходе к программированию, пригодны для разработки прикладных программ для самых различных предметных областей, а также объектно-ориентированном программировании.

СИЛЛАБУС

<table>
<tr><td>Полное название учебного курса</td><td colspan="4">Основы программирования</td></tr>
<tr><td>Краткое название курса</td><td colspan="4">Основы программирования</td></tr>
<tr><td>Кафедра</td><td colspan="4">Информационные технологии</td></tr>
<tr><td>Информация о преподавателе</td><td colspan="4">Кобилов Сами Салиевич</td></tr>
<tr><td>Длительность семестра и учебного курса</td><td colspan="4">Семестр 1-2, недель 36</td></tr>
<tr><td rowspan="6">Объём учебных часов</td><td>Всего:</td><td colspan="3">196</td></tr>
<tr><td>В том числе:</td><td>Сем. 1</td><td>Сем. 2</td><td>Всего</td></tr>
<tr><td>Лекции</td><td>20</td><td>20</td><td>40</td></tr>
<tr><td>Практические</td><td>24</td><td>24</td><td>48</td></tr>
<tr><td>Лабораторные</td><td>24</td><td>24</td><td>48</td></tr>
<tr><td>Самостоятельная работа</td><td>30</td><td>30</td><td>60</td></tr>
<tr><td>Статус учебного курса</td><td colspan="4">Блок общепрофессиональных дисциплин</td></tr>
<tr><td>Начальная подготовка</td><td colspan="4">Курс основан на знание, полученное до поступления в ВУЗ по информатике и математике</td></tr>
<tr><td>Предмет и содержание дисциплины</td><td colspan="4">Обучать студентов основам алгоритмизации и программирования на базе современных языков, систем и технологий программирования</td></tr>
<tr><td>Значение учебного предмета и его место в высшем специальном образовании</td><td colspan="4">Данный курс посвящен важнейшему разделу информатики – программированию. Здесь рассматриваются основы классического и современного методов программирования. В качестве языков и систем программирования изучаются языки Object Pascal, ассемблер и C++ и соответствующие среды программирования. Для языков приводятся необходимые сведения и примеры, чтобы сложилась общая картина, и стало возможным самостоятельное решение относительно несложных задач по программированию. Эти языки заслуженно является наиболее популярным при традиционном – процедурном – подходе к программированию, пригодны для разработки прикладных программ для самых различных предметных областей, а также объектно-ориентированном программировании.</td></tr>
<tr><td>Цели и задачи предмета</td><td colspan="4">Цель изучения факультативной дисциплины - приобретение студентами знаний о математических моделях, базовых алгоритмических структурах и ознакомление методами и технологиями программирование.</td></tr>
</table>

Квалификационные требования предмета, предъявляемые к обучаемому	В результате изучения дисциплины студент должен: - уметь согласованно решать задачи разработки эффективных моделей данных и алгоритмов их обработки при создании прикладного программного обеспечения, а также получать программные реализации полученных решений на универсальном алгоритмическом языке высокого уровня; - знать основные принципы и методологию разработки прикладного программного обеспечения, типовые способы организации программных данных, а также типовые подходы к построению программных алгоритмов; - знать синтаксис и семантику универсального алгоритмического языка программирования высокого уровня; - иметь навыки решения на персональных ЭВМ простейших задач программной обработки данных; - иметь навыки использования инструментальных программных средств в процессе разработки и сопровождения программных продуктов; - иметь представление о тенденциях и направлениях развития современных технологий программирования и обработки данных.
Взаимосвязь предмета с другими предметами	Для изучения данного предмета обучающиеся должны обладать знаниями, умениями и навыками в рамках среднего специального образования по следующим предметам: Математика, Информатика. Знания полученные в рамках данного курса являются необходимым минимумом для всех предметов приведенные в учебном плане. Так как обработка информации осуществляются с использованием полученных знаний в рамках данного предмета.

Календарьно-тематический план лекционного занятия по предмету
Основы программирования

I-Семестр
Лекции
Раздел А. Основы классического программирования

№	Раздел (тема) лекции	Часы	Литература
	1. Введение в алгоритмику и программирование	(4)	
1.1	Архитектура компьютера и его программного обеспечения. Представление данных и принципы фон Неймана.	2	[1,3,5,9,11]
1.2	Алгоритмы, их свойства и способы описания.	2	[1,3,5,9,11]
	2. Языки и основы программирования	(12)	
2.1	Структуры ЯП и программы. Типы данных, переменные, стандартные типы и функции.	2	[1,6,7,15]
2.2	Определение типов и описание переменных. Ввод-вывод данных.	2	[1,6,7,15]
2.3	Сложные операторы. Организация ветвлений.	2	[6,7,14]
2.4	Операторы цикла и принципы их использования.	2	[2,6,7]
2.5	Составные типы. Массив, запись и файл.	2	[1,2,5,7]
2.6.	Подпрограммы. Процедуры и функции.	2	[1,6,14,15]
	3. Методы и технологии программирования	(4)	
3.1	Структурное программирование.	2	[1,2,6,16]
3.2	Модули и модульное программирование.	2	[2,6,16]
	Всего	20	

Практические занятия

№	Тема занятия	Часы	Литература
	1. Характеристики компьютера. Данные и системы счисления.	(6)	
1.1	Типы данных, кодирование и системы счисления.	2	[1,3,4,9,11]
1.3	Изучение и создание блок-схем.	2	[1,5,8]
1.3	Создание алгоритмов на базе алгоритмического языка.	2	[1,5,8,16]
	2. Алгоритмика и программирование.	(16)	
2.1	Изучение системы(среды) программирования. Простые операторы и создание линейных программ.	2	[1,6,7]
2.2	Составные операторы, описание разветвляющихся процессов.	2	[1,5,7,8]
2.3	Индексные переменные, их описание и обработка.	2	[5,6,7,8]
2.4	Изучение типов и принципов действия операторов цикла.	2	[5,6,7,8]
2.5	Проектирование вложенных циклов.	2	[1,5,6,8]
2.6	Организация и обработка записей.	2	[1,5,6,8]
2.7	Разработка процедур и функций.	2	[1,5,6,8]
2.8	Описание и обработка файлов.	2	[5,6,8,15]
	3. Методы программирования.	(2)	
3.1	Использование стандартных модулей, создание структурных и модульных программ.	2	[1,5,6,7,8]
	Всего	24	

Лабораторные работы

№	Тема работы	Часы	Литература
	Создание алгоритмов и программ		
	Лабораторная работа №1. Алгоритмы разветвляющихся и циклических процессов.	(6)	
1.1.	Постановка и анализ задачи.	2	[1,5,6,8]
1.2.	Проектирование алгоритма и программы.	2	[1,5,6,8]
1.3.	Отладка и демонстрация программы, обсуждение результатов.	2	[1,5,6,8]
	Лабораторная работа №2. Программирование циклов и матричных задач.	(6)	
2.1.	Постановка и анализ задачи.	2	[1,5,6,8]
2.2.	Проектирование алгоритма и программы.	2	[1,5,6,8]
2.3.	Отладка и демонстрация программы, обсуждение результатов.	2	[1,5,6,8]
	Лабораторная работа №3. Проектирование и реализация подпрограмм.	(6)	
3.1.	Постановка и анализ задачи.	2	[1,5,6,7,8]
3.2.	Проектирование алгоритма и программы.	2	[5,6,7,8]
3.3.	Отладка и демонстрация программы, обсуждение результатов.	2	[5,6,7,8]
	Лабораторная работа №4. Описание и обработка файлов.	(6)	
4.1.	Постановка и анализ задачи.	2	[1,2,7,8]
4.2.	Проектирование алгоритма и программы.	2	[1,2,7,8]
4.3.	Отладка и демонстрация программы, обсуждение результатов.	2	[1,2,7,8]
	Всего	24	

II-Семестр

Лекции

Раздел Б. Архитектура микропроцессора и язык ассемблер

№	Раздел (тема) лекции	Часы	Литература
	1. Микропроцессоры семейства Intel x86.	(10)	
1.1	Платформы и архитектуры ПК. Процессор, регистры, память и шины.	2	[1,3,4,9,11]
1.2	Ассемблер. Структура программы, сегменты и модели памяти.	2	[9,10,11,13]
1.3	Описание процессов на ассемблере. Условные переходы и ветвления.	2	[6,7,11,12]
1.4	Безусловные переходы. Организация циклов.	2	[6,7,11,12]
1.5	Процедуры на ассемблере. Параметры и общие переменные.	2	[6,7,11,12]
	2. Элементы современного программирования на		

	базе С++.	(10)	
2.1	Введение. Тип данных, структура программы и ввод-вывод на С++.	2	[1,14,15,16]
2.2	Условные операторы **if** и **switch.**	2	[14,15,16]
2.3	Описание цикла. Цикла типа **While. Массивы.**	2	[14,15,16]
2.4	Циклов типа **for** и **do/while.**	2	[14,15,16]
2.5	Структура, объединение и описание функций.	2	[14,15,16]
	Всего	20	

Практические занятия

№	Тема занятия	Часы	Литература
	1. Базовая структура ПК и ассемблер-программы.	(12)	
1.1	Ассемблер. Применение команд и операндов.	2	[10,11,12,13]
1.2	Основы создания приложений на ассемблере.	2	[10,11,12]
1.3	Применение операторов встроенного ассемблера.	2	[6,7,15]
1.4	Описание условных и безусловных переходов.	2	[10,11,12]
1.5	Организация циклов.	2	[11,12,13]
1.6	Проектирование подпрограмм.	2	[11,12,13]
	2. Начало программирования на С++.	(12)	
2.1	Условные операторы и создание разветвляющихся программ.	2	[8,14,15]
2.2	Разработка циклических программ.	2	[8,15,16]
2.3	Описание и обработка массивов.	2	[8,14,15]
2.4	Динамические массивы, структуры и пользовательские типы.	2	[8,15,16]
2.5	Проектирование подпрограмм Параметры и их типы.	2	[14,15,16]
	3. Языковые программные интерфейсы.	(2)	
3.1	Организация интерфейсов. Взаимодействия процедур и программ.	2	[1,11,12]
	Всего	24	

Лабораторные работы

№	Тема работы	Часы	Литература
	Программирование на ассемблере и С++. **Лабораторная работа №1.** Создание линейных программ на языке ассемблер.	(6)	
1.1	Постановка и анализ задачи.	2	[8,11,12,13]
1.2	Проектирование алгоритма и программы.	2	[8,12,13]
1.3	Отладка и демонстрация программы, обсуждение результатов.	2	[11,12,13]
	Лабораторная работа №2. Организация разветвлений и циклов.	(6)	

2.1	Постановка и анализ задачи.	2	[8,12,13]
2.2	Проектирование алгоритма и программы.	2	[8,12,13]
2.3	Отладка и демонстрация программы, обсуждение результатов.	2	[11,12,13]
	Лабораторная работа №3. Обработка массивов и структур на С++.	(6)	
3.1	Постановка и анализ задачи.	2	[8,15,16]
3.2	Проектирование алгоритма и программы.	2	[8,15,16]
3.3	Отладка и демонстрация программы, обсуждение результатов.	2	[8,14,15]
	Лабораторная работа №4. Разработка	(6)	
4.1	подпрограмм.	2	[14,15,16]
4.2	Постановка и анализ задачи.	2	[14,15,16]
4.3	Проектирование алгоритма и программы.		
	Отладка и демонстрация программы, обсуждение результатов.	2	[14,15,16]
	Всего	24	

Тематика самостоятельных работ

I-семестр

Раздел А. Основы классического программирования

1. Платформы персональных компьютеров.
2. Классификация и структура языков программирования.
3. Способы описания синтаксиса языков программирования.
4. Специализированный алгоритмический язык и его конструкции.
5. Структура и компоненты интегрированных систем программирования.
6. Стандартные модули систем программирования, их структура и компоненты.

II-семестр

Раздел Б. Архитектура МП, языки ассемблер и С++

1. Процессоры Intel Pentium в современных разработках.
2. Особенности многоядерных микропроцессоров.
3. Операции со строками и массивами на ассемблере.
4. Структура и способы создания .exe-файлов.
5. Способы определения пользовательских типов в С++-программах.
6. Специализированные функции обработки файлов языка С++.

Порядок и критерии оценки знаний студентов

Система оценки и контроля знаний студентов по предмету «Основы программирования»

1. Общие положения

Данное положение разработано на основе приказа № 333 Министерство Высшего и Среднего Специального Образования Республики Узбекистан от 25 августа 2010 года “Олий таълим муассасаларида талабалар билимини

назорат килиш ва баҳолашнинг рейтинг тизими тугрисидаги низомга узгартириш ва кушимчалар киритиш ҳақида”.

1.1. Настоящее положение о рейтинговой системе **устанавливает**:

- правила оценки в баллах объема и качества знаний студентов по результатам контроля знания по всем видам учебной деятельности предмета «Основы программирования»;
- правила построения накопленной оценки учебной деятельности студента за рассматриваемый период по данному предмету;
- правила перевода баллов по какому-либо виду учебной деятельности в традиционные оценки («отлично» - 5, «хорошо» – 4, «удовлетворительно» – 3, «неудовлетворительно» – 2);
- правила определения рейтинга студентов;

1.2. Рейтинговая система имеет своей целью:

- обеспечение прозрачности требований к уровню подготовки студента и повышение объективности оценки результатов его труда;
- стимулирование ритмичной и систематической учебной деятельности студента в течение всего семестра, повышение учебной дисциплины;
- формализацию действий преподавателя в учебном процессе по организации работы студента и оценки результатов этой работы;
- стимулирование борьбы за лидерство в студенческой среде.

1.3. Определение понятий и терминов, используемых в данном положении.

Балл – единица количественной оценки успешности освоения дисциплины образовательной программы. Балл является целым числом. Максимальное значение баллов равно 100.

Текущая оценка – выставляется в баллах по результатам освоения материалов практических и лабораторных занятий.

Рубежная оценка – выставляется в баллах по результатам освоения материалов лекционных занятий.

Итоговая оценка – выставляется в баллах и проводится по окончанию курса или семестра.

Показатель освоения – определяется как сумма Текущей, Рубежной и Итоговых оценок.

Рейтинг – расчетная величина, позволяющая сравнивать итоговые показатели успеваемости студентов.

Оценка по дисциплине – целочисленное значение традиционной пятибалльной шкалы путем преобразования балльной оценки.

Проходной балл – целочисленное значение, которое вычисляется из 55% от максимального балла.

2. Методика формирования семестровой балльной оценки

Оценочные меры

для предмета «Основы программирования» на основе рейтинговой системы.

Направление образования: 5130200 – Прикладная математика и информатика.

Учебный год: 2019-2020; Курс, семестр: 1,1

Лекции: 20 ч; Практические занятия: 24 ч.

Лабораторные работы: 24 ч;

Самостоятельная работа: 30 ч;

Всего: 98 ч.

Оценка текущих контролей (ТК)

№	Оцениваемые виды работ (баллы)	1ТК	2ТК	Всего
		17	18	35
1	Активность на занятиях	2	2	4
2	Домашнее задание	3	3	6
3	Технологические работы на компьютере	2	3	5
4	Лабораторная работа	4	4	8
5	Контрольная работа	5	5	10
6	Самостоятельное образование	1	1	2

Оценка рубежных контролей(РК)

№	Оцениваемые виды работ (баллы)	1РК	2РК	Всего
		17	18	35
1	Письменная контрольная работа	8	8	16
2	Коллоквиум	8	9	17
3	Использование информационных ресурсов	1	1	2

Оценка итогового контроля (ИК)

№	Оцениваемые виды работ (баллы)	Всего	Перевод шкал	
		100		
1	Теоретический вопрос 1	20	100 балльная шкала	5 балльная шкала
2	Теоретический вопрос 2	20	90-100	Отлично
3	Теоретический вопрос 3	20	80-89,9	Хорошо
4	Теоретический вопрос 4	20	60-79,9	Удовлетворительно
5	Практическое задание	20	0-59,9	Неудовлетворительно

График

рейтингового контроля по предмету «Основы программирования»

Направление образования: 5130200 – Прикладная математика и информатика

Учебный год: 2019–2020., Курс, семестр: 1,1

Номер темы в рабочей программе	Часы				Вид контроля	Балл		Сроки (неделя)
	Лекция	Практика	Лабора-торная	всего		Макс.	Мин.	
1.1. – 1.3.	4	6	6	16	1ТК	17		По графику
2.1. – 2.4.	4	6	6	16	1РК	17		По графику
2.5. – 2.7.	6	6	6	18	2ТК	18		По графику
2.8. – 3.3.	6	6	6	18	2РК	18		По графику
Всего	20	24	24	68		70	39	
					ИК	100	60	По графику деканата
Итого						170	99	

Оценочные меры

для предмета «Основы программирования» на основе рейтинговой системы. Направление образования: 5130200 – Прикладная математика и информатика.

Учебный год: 2019-2020; Курс, семестр:1,2

Лекции: 20 ч., Практические занятия: 24 ч.

Лабораторные работы: 24 ч.,

Самостоятельная работа: 30 ч.,

Всего: 98 ч.

Оценка текущих контролей (ТК)

№	Оцениваемые виды работ(баллы)	1ТК	2ТК	Всего
		17	18	35
1	Активность на занятиях	2	2	4
2	Домашнее задание	3	3	6
3	Технологические работы на компьютере	2	3	5
4	Лабораторная работа	4	4	8
5	Контрольная работа	5	5	10
6	Самостоятельное образование	1	1	2

Оценка рубежных контролей(РК)

№	Оцениваемые виды работ(баллы)	1РК	2РК	Всего
		17	18	35
1	Письменная контрольная работа	8	8	16
2	Коллоквиум	8	9	17
3	Использование информационных	1	1	2

	ресурсов			

Оценка итогового контроля (ИК)

№	Оцениваемые виды работ(баллы)	Всего 100	Перевод шкал	
1	Теоретический вопрос 1	20	100 балльная шкала	5 балльная шкала
2	Теоретический вопрос 2	20	90-100	Отлично
3	Теоретический вопрос 3	20	80-89,9	Хорошо
4	Теоретический вопрос 4	20	60-79,9	Удовлетворительно
5	Практическое задание	20	0-59,9	Неудовлетворительно

График

рейтингового контроля по предмету «Основы программирования»
Направление образования: 5130200 – Прикладная математика и информатика
Учебный год: 2019–2020., Курс, семестр: 1,2

Номер темы в рабочей программе	Часы				Вид контроля	Балл		Сроки (неделя)
	Лекция	Практика	Лабора-торная	Всего		Макс.	Мин.	
1.1. – 1.4.	4	6	6	16	1ТК	17		По графику
1.5. – 1.9.	4	6	6	16	1РК	17		По графику
2.1. – 2.4.	6	6	6	18	2ТК	18		По графику
2.5. – 3.2.	6	6	6	18	2РК	18		По графику
Всего	20	24	24	68		70	39	
					ИК	100	60	По графику деканата
Итого						179	99	

Список литературы

1. Симанович С. и др. Специальная информатика: универсальный курс. - М.: АСТ-ПРЕСС, 2000. – 480 с.
2. Вирт Н. Алгоритмы + структуры данных= программы. - М.: Мир, 1985. - 405 с.
3. Бройдо В., Ильина О. Архитектура ЭВМ и систем. – СПб.: Питер, 2006.-718 с.
4. Жмакин А. Архитектура ЭВМ. – СПб.: ВХБ, 2006. – 320 с.
5. Информатика. Учебник под ред. Н. Макаровой. –М.: ФиС, 2000. -284 с.

6. Фаронов В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. –М.: Нолидж, 1998. - 616 с.
7. Сурков Д. и др. Программирования в среде Borland Pascal для Windows. – Минск: Высшая школа, 1996. – 426 с.
8. Абрамов А. и др. Задачи по программированию. - Наука, 1988. – 224 с.
9. Нортон П. Программно-аппаратная организация IBM PC. – М.: Мир, 1996. – 327 с.
10. Абель П. Язык ассемблера для IBM PC и программирования. –М.: Высшая школа, 1992. – 447 с.
11. Магда Ю. Ассемблер для процессоров Intel Pentium. - СПб.: Питер, 2006. – 410 с.
12. Скляров В. Программирование на языке ассемблера. - Минск: Высшая школа, 1999. – 192 с.
13. Юров В. Assembler: практикум. – СПб.: Питер, 2002. – 400 с.
14. Страуструп Б. Язык программирования C++. Специальное издание. – М.: Бином–Пресс, 2006. – 1104 с.
15. Павловская Т. C++. Программирование на языке высокого уровня. – СПб.: Питер, 2005. – 461 с.
16. Культин Н. C++ Builder в задачах и примерах. – СПб.: BXБ, 2005. - 336 с.

Электронные ресурсы

1. http://www.informatica.ru
2. http://www.intuit.ru
3. http://www.pascaldax.ru
4. http://www.cppstudio.ru
5. http://www.compteacher.ru/programming
6. http://www.citforum.ru/programming

O'ZBEKISTON RESPUBLIKASI OLIY VA O'RTA MAXSUS TA'LIM VAZIRLIGI

SAMARQAND DAVLAT UNIVERSITETI

Ro'yxatga olindi
№ 2822
«___» _____ 2019 y.

"Tasdiqlayman"
O'quv ishlari bo'yicha prorektor
_____ prof. A. Soleyev
«___» _____ 2019 y.

PROGRAMMALASH ASOSLARI FANINING ISHCHI O'QUV DASTURI

Bilim sohasi:	100 000- Gumanitar soha
Ta'lim sohasi:	130 000 - Matematika
Ta'lim yo'nalishi:	5130200 - Amaliy matematika va informatika

Samarqand - 2019

Ishchi o'quv dasturi ta'lim yo`nalishining o`quv rejasi va fanning o`quv dasturiga muvofiq ishlab chiqildi.

Tuzuvchilar:

Qobilov S. – SamDU, «Axborotlashtirish texnologiyalari» kafedrasi dotsenti, t.f.n.
Nurmamatov M.-SamDU, «Axborotlashtirish texnologiyalari» kafedrasi assistenti.

Taqrizchilar:

O'runbayev E. – SamDU, Matematik modellashtirish va kompleks dasturlash kafedrasi dotsenti.

Abdullayev A. - SamDU, Axborotlashtirish texnologiyalari kafedrasi dotsenti

Ishchi o'quv dasturi «Axborotlashtirish texnologiyalari» kafedrasining 2019 yil «27» avgustidagi **1**-sonli yig'ilishida muhokama qilindi va fakultet kengashida tasdiqlash uchun tavsiya qilindi.

Kafedra mudiri: **prof. I. Jumanov**

Ishchi o'quv dasturi SamDU "Amaliy matematika va informatika" fakulteti o'quv-uslubiy kengashida muhokama qilindi va fakultet kengashida tasdiqlash uchun tavsiya qilindi (Bayonnoma №. 1. 29.08 2019 yil)

Kengash raisi: **dots. Mamatov Sh.**

Ishchi o'quv dasturi SamDU "Amaliy matematika va informatika" fakulteti kengashida muhokama etilgan va foydalanishga tavsiya qilingan (2019 yil «30» avgust 1 sonli majlis bayonnomasi)

Fakultet kengashi raisi: **dots. Babayarov**

Kelishildi: O'quv-uslubiy boshqarma boshlig'i: **B. Aliqulov**

Kirish

Informatika fan va inson faoliyatining sohasi sifatida bir nechta predmet sohalaridan iboratdir. Ana shular ichida eng asosiysi **programmalash** hisoblanadi. Shuning uchun **“Programmalash asoslari”** fani talabalarga algoritmika va programmalash asoslarini o’rgatadi. Programmalash murakkab jarayon bo’lib qo’yilgan masalani hal qilish algoritmini loyihalash, uni programma ko’rinishda kodlashtirish, hisoblash tizimi muhitida taxlab natija olish va uni tahlil qilish bilan xarakterlanadi. Fanning bo’limlarida shu jarayonlar bilan bog’liq bo’lgan nazariy va amaliy masalalar muhokama qilinadi va o’rganiladi.

Fanning maqsad va vazifalari

Fanning o’qitishdan maqsad – Amaliy matematika va informatika ta’lim yo’nalishining bakalavr bosqichi talabalariga programmalash asoslarini yetarli darajada o’qitish, shu bilimlarga tayangan holda tadbiqiy masalalarni yechish uchun programma ta’minotini yaratish va ixtisoslashgan fanlarni o’zlashtirishda tayanch bilimlarga ega bo’lish.

Fanning vazifalari: Tadbiqiy masalalarni hisoblash tizimi yordamida yechish bosqichlari xususiyatlarini o’rganish, algoritm xossalari, tasvirlash usullari va murakkabligini tahlil qilish, yuqori va quyi darajali programmalash tillari yordamida ma’lumotlarni tasvirlash, qayta ishlash usul va vositalarini nomoyon etish, zamonaviy programmalash muhitlari va texnologiyalarini qo’llash fanning asosiy vazifasi hisoblanadi.

Fan bo’yicha talabalarning bilim, malaka va ko’nikmalariga qo’yiladigan talablar

Programmalash asoslari fanini o’zlashtirish jarayonida talabalar:

Nazariy jihatdan: ShEHM va uning programma ta’minoti arxitekturasi; algoritmlarning turlari, murakkabligi, ularni yaratishning to’liq bosqichlari; programmalash tillari va tizimlarining tarkibi; ma’lumotlar strukturasining xususiyatlari; programmalshning klassik metodlari va zamonaviy texnologiyalari g’oyalarini bilishlari kerak.

Amaliy jihatdan: Turli sanoq sistemalarida berilgan sonlar ustida amallar bajarish; algoritmlarni tasvirlash usullarini qo’llash; ma’lumotlarni oddiy va murakkab, statik va dinamik strukturalarni tasvirlash; operatsion sistema bilan muloqat qilish; programmalash tizimlari imkoniyatlaridan foydalanish; quyi (assembler) va yuqori darajali (Pascal, C++, Delphi) programmalash tillari yordamida programmalar yaratish; programmalashning klassik metodlari (strukturali va modulli) va zamonaviy texnologiyalari (o’byektga- mo’ljallangan, vizual va komponentli) yordamida programma ta’minotini loyihalash kabi masalalarni yecha olishi kerak.

Fanning o’quv rejadagi boshqa fanlar bilan o’zaro bog’liqligi va uslubiy jihatdan uzviy ketma-ketligi.

Mazkur fan o’quv rejasidagi Informatika, Algoritmlar nazariyasi, Diskret matematika va matematik mantiq, tizimli va amaliy dasturlash va ixtisoslik fanlari bilan uzviy bog’liqdir. Fanni o’rganishda akademik litsey, kasb-hunar kollejlarida

informatika kurslarida olingan nazariy va amaliy bilmlar zarur bo'lsa, o'z navbatida bu fandan olingan bilimlar o'quv rejasiga kiritiladigan ixtisoslik fanlari: Operatsion sistemalar, ma'lumotlar bazalarini boshqarish sistemalari, Web-programmalash va programmalash texnologiyalarini umumiy nazariyasini chuqur o'zlashtirishda tayanch hisoblanadi.

Fanni o'qitishda zamonaviy axborot va pedagogik texnologiyalar.

O'quv jarayoni bilan bog'liq ta'lim sifatini belgilovchi hollar quyidagilar:

Yuqori ilmiy-pedagogik darajada dars berish, muammoli ma'ruzalar o'qish, darslarni savol–javob tarzda qiziqarli tashkil qilish, ilg'or pedagogik texnologiyalardan va multimedia vositalaridan foydalanish tinglovchilarni undaydigan, o'ylantiradigan muammolarni ular oldiga qo'yish, talabchanlik, tinglovchilar bilan individual ishlash, erkin muloqat yuritishga, ilmiy izlanishga jalb qilish.

Fanni o'qitishda algoritmlarni tasvirlash usullari, zamonaviy programmalash tillari, muhit va texnologiyalari qo'llaniladi. Programmalash asoslari kursini loyixalashturishda quyidagi asosiy konseptual yondashuvlardan foydalaniladi:

Shaxsga yo'naltirilgan ta'lim. *Bu ta'lim o'z mohiyatiga ko'ra ta'lim jarayonining barcha ishtirokchilarini tulaqonli rivojlanishlarini ko'zda tutadi. Bu esa ta'limni loyihalashtirilayotganda, albatta, ma'lum bir ta'lim oluvchining shaxsini emas, avvalo, kelgusidagi mutaxassislik faoliyati bilan bog'liq o'qish maqsadlaridan kelib chiqqan holda yondoshilishni nazarda tutadi.*

Tizimli yondashuv. *Ta'lim texnologiyasi tizimning barcha belgilarini o'zida mujassam etmog'i lozim: jarayonning mantiqiyligi, uning barcha bo'g'inlarini o'zaro bog'langanligi, yaxlitligi.*

Faoliyatga yo'naltirilgan yondashuv. *Shaxsning jarayonli sifatlarini shakllantirishga, ta'lim oluvchining faoliyatni aktivlashtirish va intensivlashtirish, o'quv jarayonida uning barcha qobiliyati va imkoniyatlari, tashabbuskorligini ochishga yo'naltirilgan ta'limni ifodalaydi.*

Dialogik yondashuv. *Bu yondashuv o'quv munosabatlarini yaratish zaruriyatini bildiradi. Uning natijasida shaxsning o'z-o'zini faollashtirishi va o'z-o'zini ko'rsata olishi kabi ijodiy faoliyati kuchayadi.*

Hamkorlikdagi ta'limni tashkil etish. *Demokratik, tenglik, ta'lim beruvchi va ta'lim oluvchi faoliyat mazmunini shakllantirishda va erishilgan natijalarni baholashda birgalikda ishlashni joriy etishga e'tiborni qaratish zarurligini bildiradi.*

Muammoli ta'lim. *Ta'lim mazmunini muammoli tarzda taqdim qilish orqali ta'lim oluvchi faoliyatini aktivlashtirish usullaridan biri. Bunda ilmiy bilimni ob'yektiv qarama-qarshiligi va uni hal etish usullarini, dialektik mushohadani shakllantirish va rivojlantirishni, amaliy faoliyatga ularni ijodiy tarzda qo'llashni, mustaqil ijodiy faoliyati ta'minlanadi.*

Axborotni taqdim qilishning zamonaviy vositalari va usullarini qo'llash - **yangi kompyuter va axborot texnologiyalarini o'quv jarayoniga qo'llash.**

O'qitishning usullari va texnikasi. *Ma'ruza (kirish, mavzuga oid, vizuallash), muammoli ta'lim, keys-stadi, pinbord, paradoks va loyixalash usullari, amaliy ishlar.*

O'qitishni tashkil etish shakllari: *dialog, polilog, muloqot hamkorlik va o`zaro o'rganishga asoslangan frontal, kollektiv va guruh.*

O'qitish vositalari: *o'qitishning an'anaviy shakllari (darslik, ma'ruza matni) bilan bir katorda - kompyuter va axborot texnologiyalari.*

Kommunikatsiya usullari: *tinglovchilar bilan operativ teskari aloqaga asoslangan bevosita o'zaro munosabatlar.*

Teskari aloqa usullari va vositalari: *kuzatish, blits-so'rov, oraliq va joriy, yakunlovchi nazorat natijalarini tahlili asosida o'qitish diagnostikasi.*

Boshqarish usullari va vositalari: *o'quv mashg'uloti bosqichlarini belgilab beruvchi texnologik karta ko'rinishidagi o'quv mashgulotlarini rejalashtirish, qo'yilgan maqsadga erishishda o'qituvchi va tinglovchining birgalikdagi xarakati, nafaqat auditoriya mashg'ulotlari, balki auditoriyadan tashqari mustaqil ishlarning nazorati.*

Monitoring va baholash: *o'quv mashg'ulotida ham, butun kurs davomida ham o'qitishning natijalarini rejali tarzda kuzatib borish. Kurs oxirida test topshiriqlari yoki yozma ish variantlari yordamida tinglovchilarning bilimlari baholanadi.*

Programmalash asoslari fanidan mashg'ulotlarning mavzular va soatlar bo'yicha taqsimlanishi.

№	Mavzular nomi	Jami soat	Ma'ruza	Ama-liyot	Labora-toriya ishi	Mustaqil ta'lim
1	Kirish. Kompyuter arxitekturasi va programma ta`minoti. Fon-Neyman prinsplari	12	4	4		4
2	Algoritmlar, ularning xossalari va tasvirlash usullari.	14	2	4	4	4
3	Programmalash tili strukturasi. Ma`lumotlar tipi o'zgaruvchilar va tasvirlashlar.	16	4	4	4	4
4	Oddiy va murakkab operatorlar. Murakkab tiplar va operatorlar.	24	4	8	6	6
5	Programmalsh metodlari va texnologiyalari	20	4	4	6	6
6	Mikropretsessor aritekturasi, xotira adreslash usullari, xotira modelari	16	4	4	2	6
7	Assemblerda jarayonlarni tasvirlanishi. Shart va	18	4	4	4	6

	shartsiz o'tishlar. Sikllar					
8	Assemblerda protseduralar. Parametrlar va umumiy o'zgaruvchilar	18	2	4	6	6
9	C++ tilida ma'lumotlar tipi, programma strukturasi, amallar va tiplarni keltirish	18	4	4	4	6
10	Shartli va sikl operatorlari. Massivlar va strukturalar	20	4	4	6	6
11	Funksiyalar, ularni tashkil qilinishi va parametrlarni qo'llanilishi	20	4	4	6	6
	Jami	196	40	48	48	60

ASOSIY QISM

Fanning uslubiy jihatdan uzviy ketma-ketligi.

Asosiy qismda (ma’ruza) fanni mavzulari mantiqiy ketma-ketlikda keltiriladi. Har bir mavzuning mohiyati asosiy ta’rif va tushunchalar , tezislar va misollar orqali ochib beriladi. Bunda mavzu bo’yicha talabalarga DTS asosida etkazilishi zarur bo’lgan bilim va ko’nikmalar to’la qamrab olinishi kerak.

Asosiy qism sifatida qo’yiladigan talab mavzularning dolzarbligi, ularning ish beruvchilar talablari va ishlab chiqarish ehtiyojlariga mosligi, mamlakatimizda bo’layotgan ijtimoiy siyosiy va demokratik o’zgarishlar, iqtisodiyotni erkinlashtirish, iqtisodiy–huquqiy va boshqa sohalardagi islohatlarning ustivor masalalarini qamrab olishi hamda fan va texnologiyalarning so’nggi yutuqlari e’tiborga olinishi tavsiya etiladi.

Ma’ruza (nazariy) mashg’ulotlari mavzulari.

1. Kirish. Kompyuter arxitekturasi va uning programma ta’minoti. Shaxsiy kompyuter strukturasi va komponentalari. Kompyuterda masalalarni yechish bosqichlari. Programma va programmalash.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A3; A8; Q4; Q7; Q8.

2. Ma’lumotlar, ularni ko'dlashtirish va xotirada joylashtirish. Fon-Neyman prinsplari. Sanoq sistemalari. Ikkilik va o’n oltilik sanoq sistemalari. Fon Neyman prinsiplari. Mashina arifmetikasi va matematik mantiq elementlari. Ma’lumotlar formatlari. Kompyuter tezkorligi va unumdorligi.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A3; Q4; Q7; Q8.

3. Algoritmlar, ularning xossalariva tasvirlash usullari. Algoritm xossalari. Algoritmni tasvirlash usullari. Blok-sxemalar va maxsus algoritmik til. Algoritmlar turlari. Sxemalar. Algoritmni to’liq tuzush bosqichlari.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A3; A7;A10; Q7.

4. Programmalash tili. Programma strukturasi. Programmalash tili elementlari: alfavit, operator, sintaksis, semantika va progmatika. Sintaksisni tasvirlash usullari. Metalingvistik formulalar. Programmalash tili darajalari.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A3; A7; A8; Q1; Q7.

5. Ma’lumotlar tipi. O’zgaruvchilar, standart tiplar va funksiyalar. Til lug’ati. Identifikatorlar. Butun va haqiqiy sonlar. Simvollar va satrlar. O’zgaruvchilar. Ma’lumot. Ma’lumotlar tipi. Ifodalar.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A3; A7; A8; Q1; Q7.

6. Tiplarni aniqlash va o’zgaruvchilarni tasvirlash. Standart tiplar. Yangi tiplarni aniqlash. Amallar va standart funksiyalar. Amallar prioritetlari

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A4; A7; A8; Q1; Q6; Q7.

7. Oddiy operatorlar. Ma'lumotlarni kiritish-chiqarish usullari. Sarlavha. Ishchi qism. Bloklar, ob’ektlarning mavjudlik va amal qilish sohalari. Lokal va global o’zgaruvchilar.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A2; A4; A7; A8; Q1; Q3; Q6.

8. Murakkab operatorlar. Tarmoqlanishni tashkil qilish. Ta’minlash, o’tish, kiritish va chiqarish, tarkibiy operatorlar. Ma’lumotlarni kiritish - chiqarish konsepsiyasi. Formatlar.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A2; A4; A7; A8; Q1; Q3; Q6

9. Sikl operatorlari va ularni qo'llash prinsiplari. Massiv**,** to’plam va yozuv. Massiv indeksi va o’lchovi. To’plam ustida maxsus amallar. Yozuvni tasvirlash va qayta ishlash. Selektor tushunchasi. Bog’lanish operatori. Massivlarni statik va dinamik tasvirlanishi.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A7; A8; Q1; Q3; Q6.

10. Murakkab(tarkibli) tiplar. Massiv, yozuv va fayl**.** Shartli operator. Tanlash operatori. Sikl operatorlari: “oldshartli”, “so’ngshartli” va parametrli sikl operatorlari. Masalalar va sikl operatorini tanlash.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A7; A8; Q1; Q3; Q6;

11. Fayllar. Fayl tipidagi ma'lumotlarning xususiyatlari. Fayl turlari. Mantiqiy va fizik fayllar. Fayllarni qayta ishlash maxsus mexanizmlari (prosedura va funksiyalar).

Ta'lim texnologiyasi: dialogik yondashuv, muammoli ta'lim, ma'ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A4; A7; Q1; Q3; Q6; Q7.

12. Qimprogrammalar. Proseduralar va funksiyalar. Parametrlar turlari. Funksiyalarning proseduradan farqi. Rekursiya va rekursiv qismprogrammalar.

Ta'lim texnologiyasi: dialogik yondashuv, muammoli ta'lim, ma'ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A4; A7; Q1; Q3; Q6.

13. Ma'lumotlar va xotirani taqsimlash. Ma'lumotlarni statik va dinamik tasvirlash va tashkil qilish. Ko'rsatkichli tip. Ro'yxatlar turi, ularni tashkil qilish va qayta ishlash.

Ta'lim texnologiyasi: dialogik yondashuv, muammoli ta'lim, ma'ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A2; A3; A4; A7; A8; Q1; Q3; Q6; Q7.

14. Programmalash usullari va texnologiyalari. Programmalarni optimallash. Testlar tuzish va testlashtirish. Strukturali programmalash. Modular va modulli programmalsh.

Ta'lim texnologiyasi: dialogik yondashuv, muammoli ta'lim, ma'ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A3; A7; A8; Q1; Q6; Q7.

15. Mikroprotsessorlar arxitekturasi. Intel mikroprotsessorlarning rivojlanishi va xususiyatlari. Protsessor, registr, shina tushunchalari va ularning vazifalari. Komandalar tizimi, xotira strukturasi va xotirani taqsimlanishi.

Ta'lim texnologiyasi: dialogik yondashuv, muammoli ta'lim, ma'ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A1; A6; Q4; Q5; Q7; Q8.

16. Assembler tili sintaksisi. Ma'lumotlarni xotiraga joylashtirish. Assembler tili asosiy elementlari. Intel Pentium protsessori programmaviy modeli. Xotira modellari.

Ta'lim texnologiyasi: dialogik yondashuv, muammoli ta'lim, ma'ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A6; A9; Q4; Q5; Q9.

17. Assembler tilida programma strukturasi. Segmentlarning tashkil qilinishi. Makroassembler(MASM) direktivalari. Assembler–programma strukturasi.

Ta'lim texnologiyasi: dialogik yondashuv, muammoli ta'lim, ma'ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A6; A9; Q4; Q5; Q9.

18. Assembler tilida ilovalar yaratish asoslari. Boshlang'ich programmani assemblerlash. Yuklatgichlarni qo'llash. Programmalarni komponovka qilish(yig'ish va yuklash).

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A6; A9; Q4; Q5; Q9.

19. Assemblerda arifmetik amallar. Butun sonlarni qayta ishlash. ASCII va BCD formatli ma’lumotlarni qayta ishlash. Formatlarni almashtirish (keltirish).

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A6; A9; Q4; Q5; Q9.

20. Hisoblash sikllarni tashkil qilish. Shartli o’tishlar va tarmoqlanishlar. Shartsiz o’tish komandalari. Sikllarni tashkil qilish.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A6; A9; Q4; Q5; Q9.

21. Matematik soprotsessorni qo’llash. Soprossesor arxitekturasi va ma’lumotlar tipi. Soprossesor komandalari tizimi. Initsiallashtirish.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A6; A9; Q4; Q5; Q9.

22. Assembler tilida protseduralar. Stekni tashkil qilish. Qismprogrammalarni loyihalash prinsiplari. Parametrlar va umumiy o’zgaruvchilar.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A6; A9; Q4; Q5; Q9.

23. Assembler va yuqori darajali tillar interfeysi. Interfeys tuzishning umumiy prinsiplari. Assembler protseduralari. Yuqori darajali tillar va konsol ilovalari.

Ta’lim texnologiyasi: dialogik yondashuv, muammoli ta’lim, ma’ruza, savol- javob, munozara, algorimik yondashuv.

Adabiyotlar: A6; A9; Q4; Q5; Q9.

Programmalash asoslari fani bo’yicha ma’ruza mashg’ulotlarining kalendar tematik rejasi.

№	Ma’ruza(bob) mavzusi	soat
1	2	3
I-semestr		
	1.Algoritmika va programmalashga kirish.	
1.1	Kompyuter arxitekturasi va programma ta’minoti.	2
1.2	Algoritmlar, ularning xossalariva tasvirlash usullari.	2
	2.Dasturlash tillari va asoslari.	
2.1	Programmalash tili. Programma strukturasi	2
2.2	Tiplarni aniqlash va o’zgaruvchilarni tasvirlash.	2
2.3	Murakkab operatorlar. Tarmoqlanishni tashkil qilish	2
2.4	Sikl operatorlari va ularni qo'llash prinsiplari	2
2.5	Murakkab (tarkibli) tiplar. Massiv, yozuv va fayl	2

2.6	Qismprogrammalar. Protseduralar va funksiyalar.	2
	3.Programmalash metodlari va texnologiyalari.	
3.1	Strukturali programmalash.	2
3.2	Modullar va modulli programmalash.	2
I-semestr jami		20
II-semestr		
	1. Intel x86 oilasidagi mikroprotsessorlar.	
1.1	ShK platformalari va arxitekturasi. Protsessor, registr, xotira va shinalar.	2
1.2	Programma strukturasi, segmentlar va xotira modellari	2
1.3	Assemblerda jarayonlarning tasvirlanishi. Shartli, shartsiz o'tishlar va tarmoqlanishlar	2
1.4	Sikllarni tashkil qilish.	2
1.5	Assemblerda protseduralar. Parametrlar va umumiy o'zgaruvchilar.	2
	2. C++ da programmalash elementlari.	
2.1	Kirish. Ma'lumotlar tipi, programma strukturasi va kiritish-chiqarish	2
2.2	**If** va **switch** shartli operatorlari	2
2.3	Sikl operatorlari. **While** tipidagi sikl	2
2.4	**For** va **Do/while** tipidagi sikllar	2
2.5	Funksiyalarni tasvirlash va qo'llash.	2
II semester jami		20
O’quv yili jami		40

Amaliy mashg’ulotlarning mavzulari

1. Kompyuterning ma’lumotlarni saqlashi va arifmetik asoslari.

Sanoq sistemasi. Ikkilik sanoq sistemasida amallar. O’n oltilik sanoq sistemasida amallar. Sonlarni bir sanoq sistemasidan boshqasiga o’tkazish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A2; A5; A6; A8; Q5; Q7; Q8; Q9.

2. Algoritmika asoslari.

Algoritmlar turlari, xossalari va xususiyatlari bilan tanishish. Algoritmlarni blok-sxemalar yordamida tasvirlash usulini tahlil qilish. Blok – sxemalar yaratish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A5; A7; A8; A10; Q1; Q7;

3. Maxsus algoritmik til.

Algoritm umumiy ko’rinishi. Kattaliklar turi, komandalar, jadvallar va yordamchi algoritmlar xuxusiyatlarini o`rganish. Maxsus algoritmik tilni turli algorimlar yaratishda qo`llash.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A5; A7; A8; A10; Q1; Q7.

4. Programmalashga kirish.

Programmalash tizimi (muhiti) strukturasi bilan tanishish. Oynalar tizimi, menyu, komandalar va yordamchi qismsistemasini o’rganish. Programmani terish, taxlash va bajarish bosqichlarini tahlil qilish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q3; Q6; Q7.

5. Programmalash tili asosiy konstruksiyalari.

Oddiy tiplar, konstantalar va standart funksiyalarni qo`llash. Oddiy operatorlarni ishlatish va chiziqli programmalarni yaratish

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q3; Q6; Q7.

6. Murakkab operatorlar.

Shartli operator, tanlash operatori va sikl operatorlarini o`rganish. Tarmoqlanuvchi va takrorlanuvchi programmalarni loyihalash. Sikl operatorlarini taqqoslash.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q3; Q6; Q7.

7. Murakkab tiplar.

Massiv, to`plam va yozuv tipidagi ma’lumotlarni tashkil qilish va tasvirlash. Shu tipdagi ma’lumotlarni kiritish – chiqarish, qayta ishlash va saqlash usullaridan foydalanish. Ichma-ich joylashgan sikllarni loyihalash.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q6; Q7.

8. Qismprogrammalarni loyihalash. Protseduralarni ishlab chiqish. Funksiyalarni yaratish. Qismprogrammalar parametrlarining turli ko`rinishlaridan foydalanish. Rekursiv qismprogrammalarni ishlab chiqish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q6; Q7.

9. Fayllar va dinamik strukturali ma’lumotlarni tashkil qilish. Mantiqiy fayllarni tasvirlash. Fizik fayllarni yaratish. Fayllarni qayta ishlash qismprogrammalaridan foydalanish. Dinamik strukturalarni loyihalsh.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q6; Q7.

10. Programmalash texnologiyalarini qo`llash.

Programmalash tizimi standart modullaridan foydalanish. Strukturali programmalarni yaratish. Modulli programmalarni loyihalash. Programmalarni optimallash va testlashtirish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q6; Q7.

11. ShK arxitekturasi. Registrlar turlari va qo`llanishi. Xotira strukturasi va turlarini tahlil qilish va o`rganish. Ma’lumotlarni va o`tishlarni adreslash usul va mexanizmlarini o`rganish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A6; A7; Q4; Q5; Q8; Q9.

12. Assembler komandalar tizimi va programma strukturasi. Komandalar tizimi bilan tanishish. Xotira modullaridan foydalanish. Nishonlar, komandalar va operandlarni qo`llash. Ma’lumotlarni e’lon qilish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A6; A7; A9; Q5; Q9;

13. Oddiy assembler – programma yaratish. Ma’lumotlarni ko`chirish va va arifmetik amallarni bajarish. Mantiqiy amallarni qo`llash va formatlarni keltirish. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A7; A9; Q5; Q9.

**14. Shartli va shartsiz o`tishlarni tashkil qilish.** Komandalarni o`rganish. Tarmoqlanuvchi jarayonlarni tasvirlash usullarini qo`llash. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A7; A9; Q5; Q9.

15. Sikllarni tashkil qilish. Komandalarni o`rganish. Siklik jarayonlarni tasvirlash usullarini qo`llash. Masala tanlash va muhokama qilish. Programmani yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A7; A9; Q5; Q9.

16. Satrlar bilan ishlash. Satrlarni tashkil qilish va qayta ishlash komandalarini o`rganish. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A7; A9; Q5; Q9.

17. Assemblerda qismprogrammalar. Stekni tashkil qilish usulini o`rganish. Qismprogrammalarni loyihalash. Parametrlarni uzatish mexanizmlarini muhokama qilish. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A7; A9; Q5; Q9.

18. Assembler va yuqori darajali tillar interfeysi. Kiritilgan assembler xususiyatlarini tahlil qilish. Maxsus operatorlar va usullarini o`rganish. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A9; Q1; Q9.

19. Assembler va pascal oilasidagi tillar. Interfeysni yaratish. Ma’lumotlar tiplarini muvofiqlashtirish. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A7; A9; Q5; Q9.

20. Assembler va C oilasidagi tillar. Interfeysni yaratish. Ma’lumotlar tiplarini muvofiqlashtirish. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A9; Q4; Q5; Q9.

Programmalash asoslari fani bo’yicha amaliy mashg’ulotlarining kalendar tematik rejasi.

№	Amaliy mashg’ulot mavzusi	soat
1	**2**	**3**
I-semestr		

	1. Kompyuter, ma'lumotlar va sanoq sistemalari.	
1.1	Ma'lumotlar tipi va ko'dlashtirish.	2
1.2	Blok sxemalarni o'rganish va yaratish	2
1.3	Maxsus algoritmik til va algoritmlarni yaratish	2
	2. Algoritmika va programmalash.	
2.1	Programmalash tizimi (muxiti) strukturasi va komponentalarini o'rganish	2
2.2	Murakkab (tarkibiy) operatorlar. Tarmoqlanuvchi jarayonlarni tasvirlash	2
2.3	Indeksli o'zgaruvchilar ularni tasvirlash va qo'llash	2
2.4	Sikl operatorlarining turlari va ishlash prinsiplarini o'rganish.	2
2.5	Ichma-ich joylashgan sikliklarni loyixalash	2
2.6	Yozuvlarni tashkil qilish va qayta ishlash.	2
2.7	Pratsedura va funksiyalarni ishlab chiqish	2
2.8	Fayllarni tashvirlash va qayta ishlash	2
	3.Programmalash metodlari.	
3.1	Standart modullardan foydalanish, strukturali va modulli programmalarni yaratish.	2
	I- semestr jami	**24**
II-semestr		
	1. ShK strukturasi va assembler-programmalar.	
1.1	Nishonlar, komandalar va operandlarni qo'llash	2
1.2	Assemblerda ilovalar yaratish	2
1.3	Kiritilgan assembler operatorlarini qo'llash	2
1.4	Shartli va shartsiz o'tishlarni tasvirlash	2
1.5	Sikllarni tashkil qilinishi	2
1.6	Qismprogrammalarni loyixalash	2
	2. C++da programmalash boshlanishi.	
2.1	Shartli operatorlar va tarmoqlanuvchi programmalarni yaratish	2
2.2	Siklik programmalarni ishlab chiqish	2
2.3	Massivlarni tasvirlash va qayta ishlash	2
2.4	Dinamik massivlar, strukturalar va foydalanuvchi tiplari	2
2.5	Qismprogrammalarni loyixalash. Parametrlar va ularning tiplari	2
	3. Programmalash tillariaro interfeyslar.	
3.1	Programmalar va protseduralar xamkorligini tashkil qilish	2
	II- semestr jami	24
	O'quv yili jami	48

Laboratoriya mashg'ulotlarning mavzulari.

I - Semestr

Algoritmlar va programmalarni loyixalash.

Laboratoriya ishi №1. Tarmoqlanuvchi va siklik jarayonlar algoritmlari.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A2; A5; A6; A8; Q5; Q7; Q8; Q9.

Laboratoriya ishi №2. Sikllarni programmalash.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A5; A7; A8; A10; Q1; Q7;

Laboratoriya ishi №3. Qismprogrammalarni loyihalash va ishlab chiqish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q3; Q6; Q7. **8. Labaratoriya ishi №5. Modullararo interfeyslarni tashkil qilish.** Ma’lumotlarni ko`chirish va va arifmetik amallarni bajarish. Mantiqiy amallarni qo`llash va formatlarni keltirish. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A7; A9; Q5; Q9.

Laboratoriya ishi №4. Fayllarni tasvirlash va qayta ishlash. Ma’lumotlarni ko`chirish va va arifmetik amallarni bajarish. Mantiqiy amallarni qo`llash va formatlarni keltirish. Masala tanlash va muhokama qilish. Programmalarni yaratish, taxlash va namoyish etish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A5; A6; A7; A9; Q5; Q9.

II-Semestr

Assembler va C++ da programmalash.

Laboratoriya ishi №1. Assemblerda chiziqli (ketma-ket) programmalarni yaratish. Komandalar tizimi bilan tanishish. Xotira modullaridan foydalanish. Nishonlar, komandalar va operandlarni qo`llash. Ma’lumotlarni e’lon qilish.

Ta’lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta’limni tashkil etish, muammoli ta’lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A6; A7; A9; Q5; Q9;

Laboratoriya ishi №2. Tarmoqlanish va sikllarni taxlil qilish.

Shartli operator, tanlash operatori va sikl operatorlarini o`rganish. Tarmoqlanuvchi va takrorlanuvchi programmalarni loyihalash. Sikl operatorlarini taqqoslash.

Ta'lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta'limni tashkil etish, muammoli ta'lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q3; Q6; Q7.

Laboratoriya ishi №3. C++da massivlar va strukturalarni qayta ishlash.

Massiv, to`plam va yozuv tipidagi ma'lumotlarni tashkil qilish va tasvirlash. Shu tipdagi ma'lumotlarni kiritish – chiqarish, qayta ishlash va saqlash usullaridan foydalanish. Ichma-ich joylashgan sikllarni loyihalash.

Ta'lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta'limni tashkil etish, muammoli ta'lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q6; Q7.

Laboratoriya ishi №4. Qismprogrammalarni ishlab chiqiash. Protseduralarni ishlab chiqish. Funksiyalarni yaratish. Qismprogrammalar parametrlarining turli ko`rinishlaridan foydalanish. Rekursiv qismprogrammalarni ishlab chiqish.

Ta'lim texnologiyasi: Tizimli yondashuv, dialogik yondashuv, hamkorlikdagi ta'limni tashkil etish, muammoli ta'lim, loyihalash usullari, munozara, savol – javob, algoritmik yondashuv, kommunikatsiya usullari.

Adabiyotlar: A1; A3; A7; Q1; Q6; Q7.

Laboratoriya mashg'ulotlarining kalendar tematik rejasi

№	**Laboratoriya mashg'ulotlarining mavzusi**	
	I-Semestr	
	Algoritmlar va programmalarni loyixalash. Laboratoriya ishi №1. Tarmoqlanuvchi va siklik jarayonlar algoritmlari.	(6)
1.1	Masalaning qo'yilishi va tahlili	2
1.2	Algoritmlarni loyihalash va tasvirlash	2
1.3	Programmani taxlash va namoyish etish. Natijalarni muhokama qilish	2
	Laboratoriya ishi №2. Sikllarni programmalash.	(6)
2.1	Masalaning qo'yilishi va tahlili	2
2.2	Algiritmlarni loyihalash va tasvirlash	2
2.3	Programmani taxlash va namoyish etish. Natijalarni mohokama qilish	2
	Laboratoriya ishi №3. Qismprogrammalarni loyihalash va ishlab chiqish.	(6)
3.1	Masalaning qo'yilishi va tahlili	2
3.2	Algoritmlarni loyihalash va tasvirlash	2
3.3	Programmani taxlash va namoyish etish. Natijalarni muhokama qilish	2

	Laboratoriya ishi №4. Fayllarni tasvirlash va qayta ishlash.	(6)
4.1	Masalaning qo'yilishi va tahlili	2
4.2	Algoritmlarni loyihalash va tasvirlash	2
4.3	Programmani taxlash va namoyish etish. Natijalarni muhokama qilish	2
	I-Semestr jami	24
	II-Semestr	
	Assembler va C++ da programmalash.Laboratoriya ishi №1. Assemblerda chiziqli (ketma-ket) programmalarni yaratish.	(6)
1.1	Masalaning qo'yilishi va tahlili	2
1.2	Algoritm va programmani loyihalash	2
1.3	Programmani taxlash va namoyish etish. Natijalarni muhokama qilish	2
	Laboratoriya ishi №2. Tarmoqlanish va sikllarni taxlil qilish.	(6)
2.1	Masalaning qo'yilishi va tahlili	2
2.2	Algoritm va programmani loyihalash	2
2.3	Programmani taxlash va namoyish etish. Natijalarni muhokama qilish	2
	Laboratoriya ishi №3. C++da massivlar va strukturalarni qayta ishlash.	(6)
3.1	Masalaning qo'yilishi va tahlili	2
3.2	Algoritm va programmani loyihalash	2
3.3	Programmani taxlash va namoyish etish. Natijalarni muhokama qilish	2
	Laboratoriya ishi №4. Qismprogrammalarni ishlab chiqiash.	(6)
4.1	Masalaning qo'yilishi va tahlili	2
4.2	Algoritm va programmani loyihalash	2
4.3	Programmani taxlash va namoyish etish. Natijalarni muhokama qilish	2
	II-Semestr jami	24
	O'quv yili jami	48

Mustaqil ta'limni tashkil etishning shakli va mazmuni

Programmalash asoslari fani bo'yicha talabaning mustaqil ta'limi shu fanning o'rganish jarayonining tarkibiy qismi bo'lib, uslubiy va axborot resurslari bilan to'la ta'minlanadi.

Talabalar auditoriya mashg'ulotlarida professor–o'qituvchilarning ma'ruzasini tinglaydilar, algoritmlar va programmalar loyihalaydilar. Auditoriyadan tashqarida talaba darsga tayyorlanadi, ma'ruzalarni qayta ishlaydi, qo'shimcha adabiyotlarni konspekt qiladi, uy vazifasi sifatida berilgan konkrekt masalalarni yechish algoritmlari, programmalarni ishlab chiqadi. Ayrim mavzularni kengroq o'rganish uchun qo'shimcha adabiyotlarni o'qib reja asosida referatlar tayyorlaydi va seminarlarda himoya qiladi. Mustaqil ta'lim natijalari reyting tizimi asosida baholanadi.

Uy vazifalarini tekshirish va baholash amaliy mashg’ulot olib boruvchi o’qituvchi tamonidan, konspektlarni va mavzuni o’zlashtirish darajasini tekshirish va baholash esa ma’ruza darslarrini olib boruvchi o’qituvchi tomonidan har darsda amalga oshiriladi.

Programmalash asoslari fanidan mustaqil ish majmuasi fanning barcha asosiy mavzularini qamrab olgan va quyidagi jadvalda shakllantirilgan.

Talabalar mustaqil ta’limining mazmuni va hajmi.

№	Mustaqil ta’lim mavzulari	Beriladigan topshiriqlar	Bajarish muddati	Hajmi (soatda)
	I semester			
1.	Shaxsiy kompyuterlar platformalari	Konspekt tayyorlash. Muhokama qilish	1- hafta	4
2	Programmalash tillari sinflanishi va strukturasi	Konspekt tayyorlash. Muhokama qilish	2- hafta	4
3.	Programmalash tillari sintaksisi tasvirlash usullari	Konspekt tayyorlash. Muhokama qilish	3-hafta	4
4.	Maxsuslashtirilgan algoritmik til va uning konstruksiyalari	Konspekt tayyorlash. Muhokama qilish	4-5 haftalar	6
5.	Integrallashgan programmalash tizimlari strukturasi va komponentalari	Konspekt tayyorlash. Muhokama qilish	6-7 haftalar	6
6.	Programmalash tizmlari standart modullari tarkibi	Konspekt tayyorlash. Muhokama qilish	8-9 haftalar	6
Jami:				30
II semester				
1.	Intel Pentium protsessorlarning istiqbolli modellari	Konspekt tayyorlash. Muhokama qilish	1- hafta	4
2.	Ko'pyadroi mikroprotsessorlar xususiyatlari	Konspekt tayyorlash. Muhokama qilish	2- hafta	4
3.	Assemblerda satrlar va massivlar ustida amallar	Konspekt tayyorlash. Muhokama qilish	3- hafta	4
4.	.exe fayl strukturasi va yaratish usullari	Konspekt tayyorlash. Muhokama qilish	4-5 haftalar	6
5.	C++ programmalarida foydalanuvchi tiplarni aniqlash va qo'llash	Konspekt tayyorlash. Muhokama qilish	6-7 haftalar	6
6.	C++ tilining fayillarini qayta ish uchun maxsuslashtirilgan funksiyalari	Konspekt tayyorlash. Muhokama qilish	8-9 haftalar	6
Jami:				30
Hammasi				60

Ishchi o’quv dasturining information uslubiy ta’minoti.

Mazkur fanni o’qitish jarayonida ta’limning zamonaviy metodlari, pedagogik, axborot-kommunikatsiya texnologiyalarini qo’llash nazarda tutilgan.

- Nazariy mavzular (ma’ruzalar) zamonaviy kompyuter texnologiyalari yordamida jihozlangan elektron sinfda prezentatsiya va elektron–didaktik materiallardan foydalangan holda o’tkaziladi;

- Amaliy mashg’ulotlar auditoriyada “aqliy hujum” , “guruhli fikrlash” va “loyihani himoya qilish” pedagogik texnologiyalar asosida algoritmlar yaratish, kompyuter sinfida programmani loyihalash, taxlash va ishlash prinsiplarini namoyish etish orqali olib boriladi;

- Mustaqil ta’lim uchun ajratilgan mavzularni muhokamasi seminarlarda ma’ruza qilish, kichik ishchi guruh musobaqasini tashkil qilish va to’garaklarda dasturiy ta’minotni namoyish etish shaklida tashkil qilinadi.

Ushbu fandan talabalar bilimini reyting tizimi asosida baholash mezoni

Fan bo‘yicha reyting jadvallari, nazorat turi, shakli, soni, hamda har bir nazoratga ajratilgan maksimal ball, shuningdek joriy va oraliq nazoratlarning saralash ballari haqidagi ma’lumotlar birinchi mashg‘ulotda talabalarga e’lon qilinadi.

Davlat ta’lim standartlariga muvofiq quyidagi nazorat turlari o‘tkaziladi.

Joriy nazorat (JN). Talabaning fan mavzulari bo‘yicha bilim va amaliy ko‘nikma darajasini aniqlash va baholash usuli. JN amaliy mashg‘ulotlarda og‘zaki so‘rov, test o‘tkazish, suhbat, nazorat ishi, kollokvium, uy vazifalarini tekshirish va shu kabi boshqa nazorat shakllarida o‘tkaziladi.

Oraliq nazorat (ON). Semestr davomida o‘quv dasturining tegishli (fanning bir necha mavzularini o‘z ichiga olgan) bo‘limi tugallangandan keyin, talabaning nazariy bilim va amaliy ko‘nikma darajasini aniqlash va baholash usuli. ON bir semestrda ikki marta o‘tkaziladi va shakli (yozma, og‘zaki, test va h.k.) o‘quv faniga ajratilgan umumiy soatlar hajmidan kelib chiqqan holda belgilanadi.

Yakuniy nazorat (YaN). Semestr yakunida muayan fan bo‘yicha nazariy bilim va amaliy ko‘nikmalarni talabalar tomonidan o‘zlashtirish darajasini baholash usuli. YaN asosan tayanch tushuncha va iboralarga asoslangan “yozma ish” shaklida o‘tkaziladi.

ON o‘tkazish jarayoni kafedra mudiri tomonidan tuzilgan komissiya ishtirokida muntazam ravishda o‘rganib boriladi va uni o‘tkazish tartiblari buzilgan hollarda ON natijalari bekor qilinishi mumkin. Bunday hollarda ON qayta o‘tkaziladi.

OTM rektorining buyrug‘i bilan ichki nazorat va monitoring bo‘limi rahbarligida tuzilgan komissiya YaNni o‘tkazish jarayonini muntazam ravishda kuzatib boradi va uni o‘tkazish tartiblari buzilgan hollarda YaN natijalari bekor qilinishi mumkin. Bunday hollarda YaN qayta o‘tkaziladi.

Talabaning bilim saviyasi, ko‘nikma va malakalarini nazorat qilish reyting tizimiga asosan, talabani fan bo‘yicha o‘zlashtirish darajasi ballar orqali ifodalanadi.

Talabaning semestr davomida o‘zlashtirish ko‘rsatkichlari joriy nazorat(JN) va oraliq nazorat(ON) boýicha 70 ballik tizimda baxolanadi. Ushbu 70 ball baxolash turlari boýicha quyidagicha taqsimlanadi: JN - 35 ball; ON – 35 ball

Ball	**Baho**	**Talabaning bilim darajasi**
60-70		Ijodiy fikrlay olish; mustaqil mulohaza yurita olish;

	A’lo	olgan bilimlarini amalda qo‘llay olish; mohiyatini tushuntirish; tushunchalarni bilish, aytib berish, tasavvurga ega bo‘lish; xulosa va qaror qabul qilish
50-59,9	Yaxshi	Mustaqil mulohaza qilish; olgan bilimlarini amalda qo‘llay olish; mohiyatini tushuntirish; tushunchalarni bilish, aytib berish, tasavvurga ega bo‘lish.
39-49,9	Qoniqarli	Mohiyatini tushuntirish; tushunchalarni bilish, aytib berish, tasavvurga ega bo‘lish.
0-38,9	Qoniqarsiz	Aniq tasavvurga ega bo‘lmaslik, bilmaslik

Fan bo‘yicha saralash bali 100 ballik tiizimida 60 ballni tashkil etadi. Talabaning saralash balidan past bo‘lgan o‘zlashtirishi reyting daftarchasida qayd etilmaydi.

Talabaning o‘quv fani bo‘yicha mustaqil ishi JN. ON va YaN jarayonida tegishli topshiriqlarni bajarishi va unga ajratilgan ballardan kelib chiqqan holda baholanadi.

Talabaning fan bo‘yicha reytingi quyidagicha aniqlanadi $R = \frac{i\ * \hat{A}}{100}$, bu erda O‘ – fan bo‘yicha o‘zlashtirish darajasi (ball), V – semestrda fanga ajratilgan umumiy o‘quv yuklamasi (soat).

Fan bo‘yicha JN va ONlariga ajratilgan umumiy ballning 50% saralash bali hisoblanib, ushbu foizdan kam ball to‘plagan talaba YaNga kiritilmaydi.

JN va ON turlari bo‘yicha 55 ball va undan yuqori ballni to‘plagan talaba fanni o‘zlashtirgan deb hisoblanadi va ushbu fan bo‘yicha YaNga kirmasligi mumkin.

Talabaning semestr davomida fan bo‘yicha to‘plagan umumiy bali har bir nazorat turidan to‘plagan ballari yig‘indisiga teng.

ON va YaN turlari kalendar tematik rejasiga muvofiq dekanat tomonidan tuzilgan reyting nazorat jadvallari asosida o‘tkaziladi. YaN semestrning ohirgi ikki xaftasi mobaynida o‘tkaziladi.

JN va ON nazoratlarida saralash balidan kam ball to‘plagan va uzrli sababalrga ko‘ra nazoratda qatnasha olmagan talabaga qayta topshirish uchun navbatdagi shu nazorat turigacha, so‘nggi JN va ON uchun esa YaNgacha bo‘lgan muddat beriladi.

Talabaning semestrda JN va ON turlari bo‘yicha to‘plagan ballari ushbu nazorat turlari umumiy balining 50% dan kam bo‘lsa yoki semestr JN, ON va YaN bo‘yicha to‘plagan ballari yig‘indisi **55 balldan kam bo‘lsa** u **akademik qarzdor** deb hisoblanadi.

Talaba nazorat natijalaridan norozi bo‘lsa, fan bo‘yicha nazorat turi natijalari e’lon qilingan vaqtdan boshlab, bir kun mobaynida fakultet dekaniga ariza bilan murojaat etish mumkin. Bunday holda, dekanning taqdimnomasiga ko‘ra, rektor buyrug‘i bilan 3 (uch) a’zodan kam bo‘lmagan tarkibda apellyasiya komissiyasi tashkil etiladi.

Appelyasiya komissiyasi talabaning arizalarini ko‘rib chiqib, shu kunning o‘zida xulosasini bildiradi.

Baholashning o‘rnatilgan talablar asosida, belgilangan muddatlarda o‘tkazilishi, hamda rasmiylashtirilishi fakultet dekani, kafedra mudiri, o‘quv-uslubiy boshqarma hamda ichki nazorat va monitoring bo‘limi tomonidan nazorat qilinadi.

Talabalar JN dan to’playdigan ballarining na’munaviy mezonlari

№	Baholanadigan ish turi (ballarda)	1-JN	2-JN	Jami
		17	18	35
1	Darsdagi faollik	0-2	0-2	4
2	Uy vazifasini bajarilishi	0-3	0-3	6
3	Kompyuterda texnologik ishlar	0-2	0-3	5
4	Labaratoriya ishi	0-4	0-4	8
5	Nazorat ishi (og'zaki/yozma)	0-5	0-5	10
6	Mustaqil ta'lim topshiriqlari	0-1	0-1	2

Talabalar ON dan to’playdigan ballarining na’munaviy mezonlari

№	Baholanadigan ish turi (ballarda)	1-ON	2-ON	Jami
		17	18	35
1	Yozma nazorat ishi	0-8	0-8	16
2	Kollokvium	0-8	0-9	17
3	Axborot resurslaridan foydalanish	0-1	0-1	2

Izoh: Ballar jadvalidagi qavslar ichida ON bitta turdan iborat bo’lgan holatdagi ballarning taqsimlanishi ko’rsatilgan

Talabalar YaN dan to’playdigan ballarining na’munaviy mezonlari

№	Baxolanadigan ish turi (ballarda)	Jami	O'tkazilish shkalasi	
		100		
1	Nazariy savol 1	20	100 ballik shkala	5 ballik shkala
2	Nazariy savol 2	20	90-100	A'lo
3	Nazariy savol 3	20	80-89,9	Yaxshi
4	Nazariy savol 4	20	60-79,9	Qoniqarli
5	Amaliy masala/mustaqil ta'lim mavusi	20	0-59,9	Qoniqarsiz

Yakuniy nazoratda "Yozma ish"larni baholash mezoni

Yakuniy nazorat "Yozma ish" shaklida belgilangan bo’lsa u 100 ballik “Yozma ish” variantlari asosida o’tkaziladi. Har bir variant 4 ta nazariy savol va 1 ta amaliy topshiriqdan iborat. Nazariy savollar fan bo’yicha tayanch so’z va iboralar asosida tuziladi. Har bir nazariy va amaliy savolga yozilgan javoblar 0-6 ball oraligida baholanadi. Amaliy topshiriq esa 0-20 ball oralig’ida baholanadi.

Yozma sinov bo'yicha umumiy o'zlashtirish ko'rsatkichini aniqlash uchun variantda berilgan savollarning har biri uchun yozilgan javoblarga qo'yilgan o'zlashtirish ballari qo'shiladi va yig'indi talabaning yakuniy nazorat bo'yicha o'zlashtirish bali hisoblanadi.

Yakuniy nazoratda "Test" larni baholash mezoni

Yakuniy nazorat "Test" shaklida amalga oshirilganda, sinov ko'p variantli usulda o'tkaziladi. Har bir variant 30ta test savoldan iborat. Har bir test savoliga belgilangan javoblar bo'yicha o'zlashtirish ko'rsatkichi 0 yoki 1 ball baholanadi. Shuning uchun talaba maksimal 30 ball to'plashi mumkin.

Tavsiya etiladigan adabiyotlar ro'yxati

Asosiy adabiyotlar

1. Симанович С.В. и другие. Специальная информатика: Универсалный курс. - М.: АСТПресс, 2000. -480 с
2. Вирт Н. Алгоритмы + структкры данных = программы. М.:Мир, 1985.-405с
3. Фаронов Б. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. - М.: Нолидж, 1998. – 616 с.
4. Сурков Д. и др. Программирование в среде Borland Pascal для Windows. – Минск: Вышейшая школа, 1996. – 426 с.
5. Aripov M va boshq. Algoritm asoslari va algoritmik tillar. – Toshkent: O'zMU, 2000 . -246 b.
6. A.A.Xaldjigitov, Sh.F.Madraximov, U.E.Adamboev . Informatika va programmalsh.O'quv qo'llanma, O'zMU, 2005.- 145 b.
7. Абрамов А. и др. Задачи по программированию. – М. : Наука, 1988.-224с
8. Жмакин А. Архитектура ЭВМ. – СПб.: ВХБ, 2008. -320 с.
9. Бройдо В., Ильина О. Архитектура ЭВМ. – СПб.: Питер, 2006,-718 с
10. Магда Ю. Ассемблер для процессоров Intel Pentium. – СПб.:Питер, 2006.-410 с
11. Юров.В, Хорошенко С. Assembler : Учебный курс. - СПБ.: Питер, 2000.
12. Страуструп Б. Язык программирования С++. Специальное издание. – М.: Бинам – ПРЕСС, 2006, 1104 с
13. Павловская Т. С++ Программирование на языке высокого уровня. – СПБ. : Питер, 2005. -461 с.
14. Мадрахимов Ш., Гайназаров С. С++ тилида программалаш асослари: услубий кулланма – Тошкент: УзМУ 2009 . – 196 б
15. Madraximov Sh. va boshq. C++ tilida programmalash bo'yicha masalalar to'plami. O'quv qo'llanma. - Toshkent: O'ZMU, 2014.-160 b
16. Павловская Т.С. Щупак Ю.С. С++. Объектно- ориентированное программирование. Практикум.-СПб.: Питер,2005-265с

Qo'shimcha adabiyotlar

1. Информатика. Учебник под ред. Н. Макаровой – М. : ФиС, 2000, - 284 с

2. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. – М.: Мир, 1981. -368с
3. Qosimov S. Axborot texnologiyalari. O`quv qo`llanma. – Toshkent: Aloqachi, 2006. – 370 b
4. Абель П. Assembler для IBM PC и программирования. 1991. М.: “Высшая школа”, 1992.- 447 с.
5. Скляров В. Программирование на языке Ассемблера. – Минск: Высшая школа, 1999.-152 с.
6. Юров В. Assembler: практикум. -СПб.: Питер, 2002.- 400с.
7. Романчик В., Люлькин А. Программирование в С++ BUILDER. Учебное пособие. – Минск: БГУ, 2007. -126 с
8. Культин Н. С ++ Builder в задачах и примерах. – СПб. : ВХБ, 2005.- 336с
9. Axatov A., Qarshiyev H. C++ tilida programmalash(misollar va masalalar). 2016.SamDU nashri. 140b

Internet manbalar

1. http://www.intuit.ru
2. http://code-live.ru/tag/cpp-manual/
3. http://www.dasturchi.uz
4. http://www.c-cpp.ru/
5. http://tc.belhard.com/courselist/17.php
6. http://cppstudio.com/

Самаркандский государственный университет

Факультет прикладной математики и информатики

Кафедра информационных технологий

№612

«Утверждаю»
Проректор СамГУ
по учебной работе
проф. А. Солеев
«__» _____ 2019 г.

Бакалавриатура

Направление образования: 5130200 – Прикладная математики и информатика

Рабочая программа

предмета «Основы программирования»

(1 курс, 1-2 семестры)

Лекций: 40 ч., Практических занятий: 48 ч.

Лабораторные работы: 48 ч., Самостоятельное образование: 60 ч.

Всего: 196 ч.

Рабочая программа:

Обсуждена и утверждена на заседании кафедры от 27 августа 2019 г.

(Постановление №1)

Зав. кафедрой: проф. И .Жуманов

Обсуждена и утверждена на учебно-методическом совете факультета от 29.08 2019 г. (Постановление №1).

Председатель совета: доц. Ш. Маматов

Обсуждена и утверждена на Учёном совете факультета от 30.08 2019 г.

(Постановление №1).

Председатель совета, декан: доц. А. Бабаяров

Самарканд - 2019

Содержание предмета
I-Семестр
Лекции
Раздел А. Основы классического программирования

№	Раздел (тема) лекции	Часы	Литература
	4. Введение в алгоритмику и программирование	(4)	
1.1	Архитектура компьютера и его программного обеспечения. Представление данных и принципы фон Неймана.	2	[1,3,5,9,11]
1.2	Алгоритмы, их свойства и способы описания.	2	[1,3,5,9,11]
	5. Языки и основы программирования	(12)	
2.1	Структуры ЯП и программы. Типы данных, переменные, стандартные типы и функции.	2	[1,6,7,15]
2.2	Определение типов и описание переменных. Ввод-вывод данных.	2	[1,6,7,15]
2.3	Сложные операторы. Организация ветвлений.	2	[6,7,14]
2.4	Операторы цикла и принципы их использования.	2	[2,6,7]
2.5	Составные типы. Массив, запись и файл.	2	[1,2,5,7]
2.6.	Подпрограммы. Процедуры и функции.	2	[1,6,14,15]
	6. Методы и технологии программирования	(4)	
3.1	Структурное программирование.	2	[1,2,6,16]
3.2	Модули и модульное программирование.	2	[2,6,16]
	Всего	20	

Практические занятия

№	Тема занятия	Часы	Литература
	1. Характеристики компьютера. Данные и системы счисления.	(6)	
1.1	Типы данных, кодирование и системы счисления.	2	[1,3,4,9,11]
1.3	Изучение и создание блок-схем.	2	[1,5,8]
1.3	Создание алгоритмов на базе алгоритмического языка.	2	[1,5,8,16]
	2. Алгоритмика и программирование.	(16)	
2.1	Изучение системы(среды) программирования. Простые операторы и создание линейных программ.	2	[1,6,7]
2.2	Составные операторы, описание разветвляющихся процессов.	2	[1,5,7,8]
2.3	Индексные переменные, их описание и обработка.	2	[5,6,7,8]
2.4	Изучение типов и принципов действия операторов цикла.	2	[5,6,7,8]
2.5	Проектирование вложенных циклов.	2	[1,5,6,8]
2.6	Организация и обработка записей.	2	[1,5,6,8]
2.7	Разработка процедур и функций.	2	[1,5,6,8]
2.8	Описание и обработка файлов.	2	[5,6,8,15]
	4. Методы программирования.	(2)	
3.1	Использование стандартных модулей, создание структурных и модульных программ.	2	[1,5,6,7,8]
	Всего	24	

Лабораторные работы

№	Тема работы	Часы	Литература
	Создание алгоритмов и программ		
	Лабораторная работа №1. Алгоритмы разветвляющихся и циклических процессов.	(6)	
1.1.	Постановка и анализ задачи.	2	[1,5,6,8]
1.2.	Проектирование алгоритма и программы.	2	[1,5,6,8]
1.3.	Отладка и демонстрация программы, обсуждение результатов.	2	[1,5,6,8]
	Лабораторная работа №2. Программирование циклов и матричных задач.	(6)	
2.1.	Постановка и анализ задачи.	2	[1,5,6,8]
2.2.	Проектирование алгоритма и программы.	2	[1,5,6,8]
2.3.	Отладка и демонстрация программы, обсуждение результатов.	2	[1,5,6,8]
	Лабораторная работа №3. Проектирование и реализация подпрограмм.	(6)	
3.1.	Постановка и анализ задачи.	2	[1,5,6,7,8]
3.2.	Проектирование алгоритма и программы.	2	[5,6,7,8]
3.3.	Отладка и демонстрация программы, обсуждение результатов.	2	[5,6,7,8]
	Лабораторная работа №4. Описание и обработка файлов.	(6)	
4.1.	Постановка и анализ задачи.	2	[1,2,7,8]
4.2.	Проектирование алгоритма и программы.	2	[1,2,7,8]
4.3.	Отладка и демонстрация программы, обсуждение результатов.	2	[1,2,7,8]
	Всего	24	

II-Семестр

Лекции

Раздел Б. Архитектура микропроцессора и язык ассемблер

№	Раздел (тема) лекции	Часы	Литература
	1. Микропроцессоры семейства Intel x86.	(10)	
1.1	Платформы и архитектуры ПК. Процессор, регистры, память и шины.	2	[1,3,4,9,11]
1.2	Ассемблер. Структура программы, сегменты и модели памяти.	2	[9,10,11,13]
1.3	Описание процессов на ассемблере. Условные переходы и ветвления.	2	[6,7,11,12]
1.4	Безусловные переходы. Организация циклов.	2	[6,7,11,12]
1.5	Процедуры на ассемблере. Параметры и общие переменные.	2	[6,7,11,12]
	2. Элементы современного программирования на базе C++.	(10)	
2.1	Введение. Тип данных, структура программы и ввод-вывод на C++.	2	[1,14,15,16]
2.2	Условные операторы **if** и **switch.**	2	[14,15,16]
2.3	Описание цикла. Цикла типа **While. Массивы.**	2	[14,15,16]
2.4	Циклов типа **for** и **do/while.**	2	[14,15,16]
2.5	Структура, объединение и описание функций.	2	[14,15,16]
	Всего	20	

Практические занятия

№	Тема занятия	Часы	Литература
	1. Базовая структура ПК и ассемблер-программы.	(12)	
1.1	Ассемблер. Применение команд и операндов.	2	[10,11,12,13]
1.2	Основы создания приложений на ассемблере.	2	[10,11,12]
1.3	Применение операторов встроенного ассемблера.	2	[6,7,15]
1.4	Описание условных и безусловных переходов.	2	[10,11,12]
1.5	Организация циклов.	2	[11,12,13]
1.6	Проектирование подпрограмм.	2	[11,12,13]
	2. Начало программирования на C++.	(12)	
2.1	Условные операторы и создание разветвляющихся программ.	2	[8,14,15]
2.2	Разработка циклических программ.	2	[8,15,16]
2.3	Описание и обработка массивов.	2	[8,14,15]
2.4	Динамические массивы, структуры и пользовательские типы.	2	[8,15,16]
2.5	Проектирование подпрограмм Параметры и их типы.	2	[14,15,16]
	3. Языковые программные интерфейсы.	(2)	

3.1	Организация интерфейсов. Взаимодействия процедур и программ.	2	[1,11,12]
	Всего	24	

Лабораторные работы

№	Тема работы	Часы	Литература
	Программирование на ассемблере и C++.		
	Лабораторная работа №1. Создание линейных программ на языке ассемблер.	(6)	
1.1	Постановка и анализ задачи.	2	[8,11,12,13]
1.2	Проектирование алгоритма и программы.	2	[8,12,13]
1.3	Отладка и демонстрация программы, обсуждение результатов.	2	[11,12,13]
	Лабораторная работа №2. Организация разветвлений и циклов.	(6)	
2.1	Постановка и анализ задачи.	2	[8,12,13]
2.2	Проектирование алгоритма и программы.	2	[8,12,13]
2.3	Отладка и демонстрация программы, обсуждение результатов.	2	[11,12,13]
	Лабораторная работа №3. Обработка массивов и структур на C++.	(6)	
3.1	Постановка и анализ задачи.	2	[8,15,16]
3.2	Проектирование алгоритма и программы.	2	[8,15,16]
3.3	Отладка и демонстрация программы, обсуждение результатов.	2	[8,14,15]
	Лабораторная работа №4. Разработка подпрограмм.	(6)	
4.1	Постановка и анализ задачи.	2	[14,15,16]
4.2	Проектирование алгоритма и программы.	2	[14,15,16]
4.3	Отладка и демонстрация программы, обсуждение результатов.	2	[14,15,16]
	Всего	24	

Тематика самостоятельных работ
I-семестр
Раздел А. Основы классического программирования

7. Платформы персональных компьютеров.
8. Классификация и структура языков программирования.
9. Способы описания синтаксиса языков программирования.
10. Специализированный алгоритмический язык и его конструкции.
11. Структура и компоненты интегрированных систем программирования.
12. Стандартные модули систем программирования, их структура и компоненты.

II-семестр
Раздел Б. Архитектура МП, языки ассемблер и С++

7. Процессоры Intel Pentium в современных разработках.
8. Особенности многоядерных микропроцессоров.
9. Операции со строками и массивами на ассемблере.
10. Структура и способы создания .exe-файлов.
11. Способы определения пользовательских типов в С++-программах.
12. Специализированные функции обработки файлов языка С++.

Литература

17. Симанович С. и др. Специальная информатика: универсальный курс. - М.: АСТ-ПРЕСС, 2000. – 480 с.
18. Вирт Н. Алгоритмы + структуры данных= программы. - М.: Мир, 1985. - 405 с.
19. Бройдо В., Ильина О. Архитектура ЭВМ и систем. – СПб.: Питер, 2006.-718 с.
20. Жмакин А. Архитектура ЭВМ. – СПб.: ВХБ, 2006. – 320 с.
21. Информатика. Учебник под ред. Н. Макаровой. –М.: ФиС, 2000. -284 с.
22. Фаронов В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. –М.: Нолидж, 1998. - 616 с.
23. Сурков Д. и др. Программирования в среде Borland Pascal для Windows. – Минск: Высшая школа, 1996. – 426 с.
24. Абрамов А. и др. Задачи по программированию. - Наука, 1988. – 224 с.
25. Нортон П. Программно-аппаратная организация IBM PC. – М.: Мир, 1996. – 327 с.
26. Абель П. Язык ассемблера для IBM PC и программирования. –М.: Высшая школа, 1992. – 447 с.
27. Магда Ю. Ассемблер для процессоров Intel Pentium. - СПб.: Питер, 2006. – 410 с.
28. Скляров В. Программирование на языке ассемблера. - Минск: Высшая школа, 1999. – 192 с.
29. Юров В. Assembler: практикум. – СПб.: Питер, 2002. – 400 с.
30. Страуструп Б. Язык программирования С++. Специальное издание. – М.: Бином–Пресс, 2006. – 1104 с.
31. Павловская Т. С++. Программирование на языке высокого уровня. – СПб.: Питер, 2005. – 461 с.
32. Культин Н. С++ Builder в задачах и примерах. – СПб.: ВХБ, 2005. - 336 с.

Электронные ресурсы

7. http://www.informatica.ru
8. http://www.intuit.ru

9. http://www.pascaldax.ru
10. http://www.cppstudio.ru
11. http://www.compteacher.ru/programming
12. http://www.citforum.ru/programming

Самаркандский государственный университет
Факультет прикладной математики и информатики
Кафедра информационных технологий
Оценочные меры

для предмета «Основы программирования» на основе рейтинговой системы.

Направление образования: 5130200 – Прикладная математика и информатика.

Учебный год: 2019-2020; Курс, семестр: 1,1

Лекции: 20 ч; Практические занятия: 24 ч.Лабораторные работы: 24 ч; Самостоятельная работа: 30 ч;Всего: 98 ч.

Оценка текущих контролей (ТК)

№	Оцениваемые виды работ (баллы)	1ТК	2ТК	Всего
		17	18	35
1	Активность на занятиях	2	2	4
2	Домашнее задание	3	3	6
3	Технологические работы на компьютере	2	3	5
4	Лабораторная работа	4	4	8
5	Контрольная работа	5	5	10
6	Самостоятельное образование	1	1	2

Оценка рубежных контролей(РК)

№	Оцениваемые виды работ (баллы)	1РК	2РК	Всего
		17	18	35
1	Письменная контрольная работа	8	8	16
2	Коллоквиум	8	9	17
3	Использование информационных ресурсов	1	1	2

Оценка итогового контроля (ИК)

№	Оцениваемые виды работ (баллы)	Всего 100	Перевод шкал	
1	Теоретический вопрос 1	20	100 балльная шкала	5 балльная шкала
2	Теоретический вопрос 2	20	90-100	Отлично
3	Теоретический вопрос 3	20	80-89,9	Хорошо
4	Теоретический вопрос 4	20	60-79,9	Удовлетворительно
5	Практическое задание	20	0-59,9	Неудовлетворительно

Самаркандский государственный университет
Факультет прикладной математики и информатики
Кафедра информационных технологий

График

рейтингового контроля по предмету «Основы программирования»
Направление образования: 5130200 – Прикладная математика и информатика
Учебный год: 2019–2020., Курс, семестр: 1,1

Номер темы в рабочей программе	Часы				Вид контроля	Балл		Сроки (неделя)
	Лекция	Практика	Лабора-торная	всего		Макс.	Мин.	
1.1. – 1.3.	4	6	6	16	1ТК	17		По графику
2.1. – 2.4.	4	6	6	16	1РК	17		По графику
2.5. – 2.7.	6	6	6	18	2ТК	18		По графику
2.8. – 3.3.	6	6	6	18	2РК	18		По графику
Всего	20	24	24	68		70	39	
					ИК	100	60	По графику деканата
Итого						170	99	

Самаркандский государственный университет
Факультет прикладной математики и информатики
Кафедра информационных технологий

Оценочные меры

для предмета «Основы программирования» на основе рейтинговой системы. Направление образования: 5130200 – Прикладная математика и информатика.
Учебный год: 2019-2020; Курс, семестр:1,2
Лекции: 20 ч., Практические занятия: 24 ч.
Лабораторные работы: 24 ч.,
Самостоятельная работа: 30 ч.,
Всего: 98 ч.

Оценка текущих контролей (ТК)

№	Оцениваемые виды работ(баллы)	1ТК	2ТК	Всего
		17	18	35
1	Активность на занятиях	2	2	4
2	Домашнее задание	3	3	6

3	Технологические работы на компьютере	2	3	5
4	Лабораторная работа	4	4	8
5	Контрольная работа	5	5	10
6	Самостоятельное образование	1	1	2

Оценка рубежных контролей(РК)

№	Оцениваемые виды работ(баллы)	1РК	2РК	Всего
		17	18	35
1	Письменная контрольная работа	8	8	16
2	Коллоквиум	8	9	17
3	Использование информационных ресурсов	1	1	2

Оценка итогового контроля (ИК)

№	Оцениваемые виды работ(баллы)	Всего	Перевод шкал	
		100		
1	Теоретический вопрос 1	20	100 балльная шкала	5 балльная шкала
2	Теоретический вопрос 2	20	90-100	Отлично
3	Теоретический вопрос 3	20	80-89,9	Хорошо
4	Теоретический вопрос 4	20	60-79,9	Удовлетворительно
5	Практическое задание	20	0-59,9	Неудовлетворительно

Самаркандский государственный университет
Факультет прикладной математики и информатики
Кафедра информационных технологий
График

рейтингового контроля по предмету «Основы программирования»
Направление образования: 5130200 – Прикладная математика и информатика
Учебный год: 2019–2020., Курс, семестр: 1,2

Номер темы в рабочей программе	Часы				Вид контроля	Балл		Сроки (неделя)
	Лекция	Практика	Лабора-торная	Всего		Макс.	Мин.	
1.1. – 1.4.	4	6	6	16	1ТК	17		По графику
1.5. – 1.9.	4	6	6	16	1РК	17		По графику
2.1. – 2.4.	6	6	6	18	2ТК	18		По графику
2.5. – 3.2.	6	6	6	18	2РК	18		По графику
Всего	20	24	24	68		70	39	

					ИК	100	60	По графику деканата
Итого						179	99	

Рабочую программу составил: доц. Кобилов С. С.

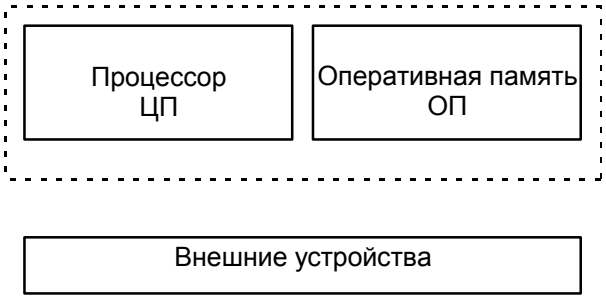
Зав. кафедрой: проф. Жуманов И. И.

ТЕКСТ ЛЕКЦИИ

Лекции 1. Архитектура компьютера и его программного обеспечения. Представление данных и принципы фон Неймана.

Структура компьютера.

Вычислительная машина состоит из следующих компонент



Назначение компонент.

Процессор – управляет работой ЭВМ, обеспечивает выполнение программ.

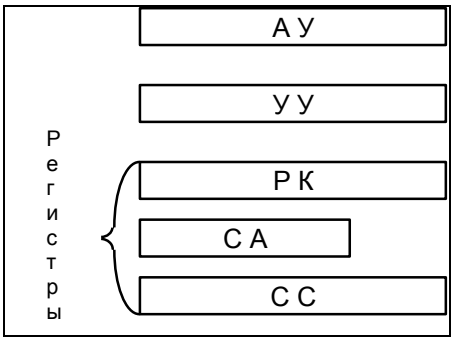
Оперативная память – используется для хранения данных и программ во время работы ЭВМ

Внешние устройства – служат для связи ЭВМ с внешним миром

Рассмотрим структуру и работу каждой из компонент.

Процессор. Такт работы процессора.

Процессор включает в себя следующие устройства.



ЦП

АЛУ (арифметическое устройство) – оно выполняет арифметические и логические операции (например, сложение, вычитание, умножение)

УУ (устройство управления) – управляет работой процессора

Регистры - специальные ячейки, которые находятся в ЦП.

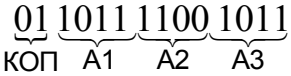
РК (регистр команды) содержит машинную команду, которую выполняет в данный момент процессор.

СА (счетчик адреса) содержит адрес следующей команды.

СС (слово-состояние) содержит информацию о результате выполнения команды.

Выполнение процессором одной машинной команды назовём тактом работы процессора.

Рассмотрим, как процессор выполняет машинную команду на примере команды



Пусть 01 изображает код операции (КОП) "сложение"; А1, А2, А3 - адреса первого операнда, второго операнда и результата соответственно.

1. В РК считывается из ОП команда, адрес которой записан в СА.

2. Содержание СА увеличивается на 1, так что теперь в СА получился адрес следующей команды программы.

3.УУ анализирует содержимое РК и организует выполнение команды.

Выделяется КОП. Определяется, что надо выполнить операцию "сложение". Определяются А1, А2, А3. Содержимое ячеек ОП с адресами А1, А2 пересылаются в АЛУ. Далее АЛУ выполняет действие сложение. Результат сложения из АЛУ пересылается в ячейку памяти с адресом А3. В регистр СС записывается информация об удачном (или неудачном) окончании выполнения сложения.

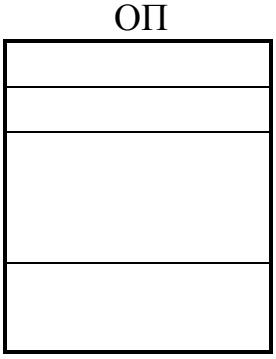
Далее работа повторяется с первого шага.

Ранее мы отмечали, что ячейка ОП может содержать данное или команду. Теперь понятно, как процессор отличает данное от команды: если адрес ячейки встретился в команде как адрес операнда, процессор обрабатывает содержимое ячейки как данное; если адрес ячейки получился в счетчике адреса, процессор обрабатывает содержимое ячейки как команду. Содержимое одной и той же ячейки в один момент работы процессора может трактоваться как данное, а в другой - как команда.

Перед началом работы процессора в регистр СА записывается аппаратно всегда один и тот же адрес, и первая команда программы должна располагаться в ОП в ячейке именно с этим адресом.

Оперативная память.

Мы касались устройства оперативной памяти. Напомним основные сведения.



Оперативная память (ОП) состоит из ячеек. Каждая ячейка имеет свой адрес - число от 0 до N-1. Количество ячеек (N) называется объемом ОП. Заметим, что объем ОП и размер регистра СА взаимосвязаны: количество разрядов в СА должно быть достаточно для хранения любого возможного адреса. (Наибольший возможный адрес в нашем предположении N-1.)

Ячейки состоят из разрядов. Количество разрядов во всех ячейках одинаково и называется разрядностью машины. Каждый разряд содержит одну двоичную цифру. Иногда разряды называются битами. В программировании слово "бит" используют в двух смыслах:

Бит - один двоичный разряд ячейки.

- содержимое одного двоичного разряда.

Содержимое ячейки называют **словом** или **машинным словом**.

Содержимое ОП. Ячейки могут хранить данные и команды. Команды, составляющие программу, обычно располагаются в ОП последовательно, друг за другом. Что именно содержит ячейка - данное или команду - определяется в момент использования содержимого ячейки.

Работа ОП. Заметим, что ячейка ОП всегда имеет некоторое содержимое. В самом деле, электронное устройство, являющееся разрядом в ячейке, обязательно находится в каком-нибудь состоянии; это состояние изображает "0" или "1". Таким образом, ячейка заполнена нулями и единицами. Однако это содержимое не имеет никакого смысла. Для того, чтобы можно было обрабатывать данное, содержащееся в ячейке, надо сначала это данное записать в ячейку.

Итак, есть две основные операции работы с ОП:

1. Запись данного в ячейку.

ЦП сообщает ОП, чтó именно надо записать и по какому адресу. При записи в ячейку ее старое содержимое теряется, становится недоступным.

2. Чтение содержимого ячейки.

ЦП передает ОП нужный адрес. Содержимое ячейки с этим адресом считывается и пересылается в ЦП. При чтении содержимое ячейки не изменяется.

Чтение и запись в ОП производится специальными электронными схемами. При выключении вычислительной машины содержимое ОП теряется.

Принципы фон-Неймана.

1. Линейная организация памяти.

Ячейки памяти располагаются последовательно по возрастанию номеров.

2. Прямой доступ к элементам памяти.

Доступ к ячейке осуществляется по её адресу, в каждый момент работы компьютера можно обратиться к любой ячейке памяти.

Этот принцип обеспечивает облегчение программирования, удобство и надежность использования ЭВМ. (Вспомните машину Тьюринга. Для доступа к ячейке, отстоящей, например, на три ячейки правее данной, требовалось вводить три дополнительных состояния.)

3. Использование двоичной системы для хранения и обработки информации.

Этот принцип следует прежде всего из практических соображений: довольно легко с помощью электронных устройств реализовать два возможных состояния - 0 и 1.

4. Принцип хранимой программы.

Программа, управляющая процессом вычислений, хранится в памяти машины.

Этот принцип обеспечивает универсальность ЭВМ. (Сравним с машиной Тьюринга: каждая машина Тьюринга имела одну программу и могла решать только одну задачу!)

5. Машинные операции.

Существует набор действий по обработке данных, выполняемых аппаратно (реализованных в виде электронных схем). Эти действия называются машинными операциями.

Чем больше машинных операций, тем легче программировать для ЭВМ и тем выше ее быстродействие. (Вспомните, чтобы прибавить 1 к числу с помощью МТ, требовалось написать достаточно объемную программу.)

Каждой машинной операции соответствует машинная команда - последовательность нулей и единиц, которую может понять и выполнить процессор.

Таким образом, содержащиеся в ячейке памяти нули и единицы могут изображать данное, а могут являться командой. Что же именно записано в ячейке - данное или команда - определяется во время работы ЭВМ. В дальнейшем мы обсудим подробнее этот вопрос.

Итак, команда - это приказ процессору выполнить машинную операцию. Последовательность команд называется программой.

6. Последовательное исполнение команд.

Команды, записанные в памяти компьютера, выполняются последовательно, друг за другом.

Лекции 2. Алгоритмы, их свойства и способы описания.

В 1983 году отмечалось 1200-летие со дня рождения величайшего средневекового ученого Средней Азии Мухамеда ибн Мусы аль-Хорезми. С именем этого ученого связано понятие алгоритма.

Алгоритм – *точное, понятное предписание исполнителю совершить последовательность действий, направленных на решение поставленной задачи.*

Итак, алгоритм - это определённая последовательность действий, которые необходимо выполнить, чтобы получить результат. Алгоритм может

представлять собой некоторую последовательность вычислений, а может - последовательность действий нематематического характера. Для любого алгоритма справедливы общие закономерности - свойства алгоритма.

Свойства алгоритма

1. *Дискретность.*
2. *Конечность и понятность*
3. *Детерминированность*
4. *Массовость*
5. *Результативность*

Дискретность - это свойство алгоритма, когда алгоритм разбивается на конечное число элементарных действий (шагов).

Конечность и понятность - свойство алгоритма, при котором каждое из этих элементарных действий (шагов) являются законченными и понятными.

Детерминированность (определенность) - свойство, когда каждое действие (операция, указание, шаг, требование) должно пониматься в строго определённом смысле, чтобы не оставалась места произвольному толкованию, чтобы каждый, прочитавший указание, понимал его однозначно.

Массовость - свойство, когда по данному алгоритму должна решаться не одна, а целый класс подобных задач.

Результативность – свойство, при котором любой алгоритм в процессе выполнения должен приводить к определённому результату. Отрицательный результат также является результатом.

Способы задания алгоритмов

1. ***Словесное описание*** (на языке исполнителя).

Словесная запись алгоритма представляет собой последовательность этапов обработки данных и задается в произвольном изложении на естественном языке. Ориентирована на исполнителя-человека

2. ***Табличное описание***

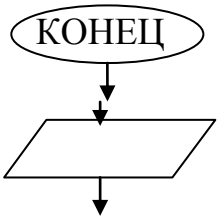
ФИО	1.Кол-во дней	2.Дневная тарифная ставка	3. З/пл= 1*2
Иванов	25	100	2500

3. ***Графическое описание или блок – схема алгоритм***

Алгоритм изображается в виде последовательности, связанных между собой функциональных блоков, каждому из которых соответствует выполнение одного или нескольких действий (операторов).

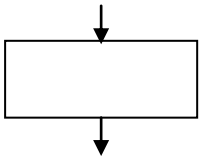
Приняты определенные стандарты графических изображений функциональных блоков.

- начало алгоритма

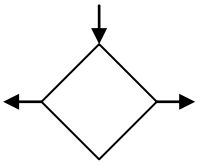


- конец алгоритма

- ввод/вывод



- оператор действия



- проверка условия

Внутри блока запись не формализована

4. ***Псевдокод*** - представляет собой систему обозначений и правил для единообразной записи алгоритма. Псевдокод занимает промежуточное положение между словесной записью алгоритма и алгоритмическим языком. В псевдокоде не приняты строгие синтаксические правила для записи команд, присущих формальным языкам. Вводятся понятия служебных слов, смысл которых определён раз и навсегда. Служебные слова выделяются в печатном тексте жирным шрифтом.

Рассмотрим запись алгоритма Евклида на псевдокоде:

Алг алгоритм Евклида
Арг N,M
Рез НОД
Нач
Пока N ≠M
  Н.ц.
    Если M > N
      То M: = M-N
      Иначе N: = N-M
        Всё
К.ц.
НОД: = M
Кон

5. ***На алгоритмическом языке*** (программа).

Алгоритмический язык ориентирован на исполнителя ЭВМ, полностью формализованная запись алгоритма.

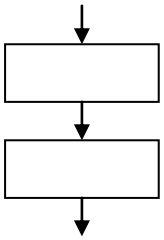
Рассмотрим запись алгоритма Евклида на языке программирования Pascal:

```
Program NOD;
Var n,m:word;
Begin
    Writeln(‘Input m,n);
    Readln(m,n);
    While m<>n do
       If m>n then m:=m-n
              else n:=n-m;
    Writeln(‘nod=’,m)
End.
```

Типы алгоритмических структур.

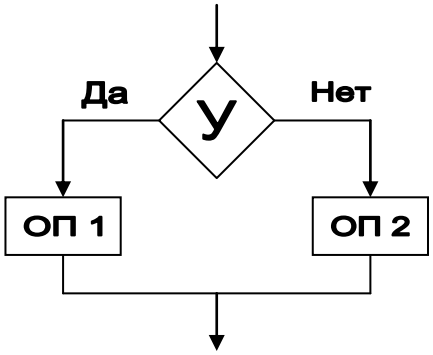
Существует три алгоритмические структуры. Из них составляются алгоритмы любой сложности.

1) СЛЕДОВАНИЕ



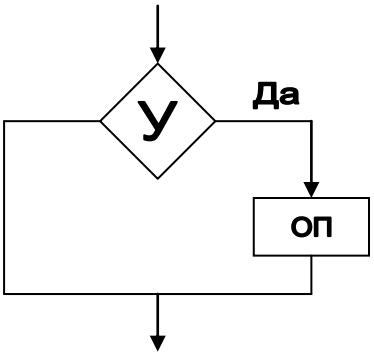
2) ВЕТВЛЕНИЕ

a) полное



Если условие выполняется, то оператор 1, а если оно не выполняется, то оператор 2.

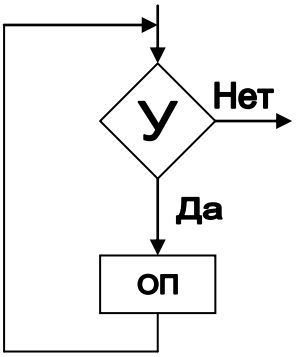
b) неполное



Если условие выполняется, то оператор выполняется, а если нет, то ни чего не происходит (просто пролетает).

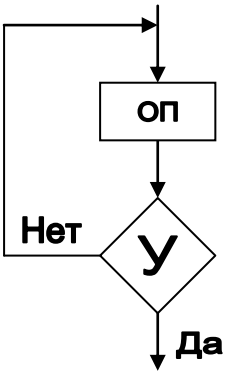
3) ЦИКЛ

a) с предусловием



Если условие истинно, то выполняется оператор, а если ложно, то выходит из цикла.

b) с постусловием



Пока условие ложно, мы в цепи, а когда истинно, то мы вне цепи.

Каждая последовательность имеет вход и выход.

Лекции 3. Структуры ЯП и программы. Типы данных, переменные, стандартные типы и функции.

Парадигма - это совокупность научных методов и решений, определяющих какое-либо направление в науке.

В процессе развития компьютеров и методов обработки информации появилось большое количество языков программирования: Фортран, Пролог, Алгол, Ada, Basic, Modula, Лисп, Паскаль, PL/1, Си, Си++, Java и другие. Все языки условно можно поделить на группы, которые отличаются друг от друга походом к решению задачи на компьютере и, соответственно, методами построения программ для реализации их решения. Так

сформировалось понятие ***парадигмы*** программирования. Понятие это еще не устоявшееся, и в литературе можно встретить различные описания парадигм программирования. Мы выделим три наиболее часто встречающихся: директивное программирование, декларативное программирование и объектно-ориентированное программирование

Директивное программирование (которое в литературе так же определяется как императивное, процедурное, процедурно-ориентированное, модульное, структурное) предполагает такое решение задачи, где программа представляет собой последовательность команд, которую компьютер затем транслирует в машинный код. Представителями таких языков программирования из перчисленных яяются Фортран, Алгол, Ada, Basic, Modula, Паскаль, PL/1, Си.

При **декларативном** программировании на первом месте выступает не собственно последовательность действий, приводящих к результату, а сама задача, которую надо решить. Программа представляет собой детальное формализованное описание постановки задачи, которое затем транслируется в машинную программу (Пролог, Лисп). Эта парадигма получила также название **функционально-логического программирования.**

Объектно-ориентированное программирование в настоящий момент используют высоко квалифицированные программисты для создания больших программных комплексов. Оно предполагает жесткую организацию данных и тесную связь с методами их обработки. Для изучения программирования это наиболее сложная парадигма.

В настоящем курсе мы будем ориентированы на первую парадигму программирования.

Этапы решения задач на компьютере.

1) **Постановка задачи** в виде текста, или информационных таблиц, или математических формул, или в любой другой форме, корректно описывающей задачу данной предметной области, например:

Задача: Посчитать объем выпускаемой продукции за некоторый промежуток времени, если известна производительность труда в каждый момент этого промежутка.

Задача: Рассчитать демографию населения на ближайшие 10 лет.

Задача: Решить квадратное уравнение.

2) Составление **математической модели** задачи. Задача должна быть максимально формализована и описана в виде математических формул, например:

W – объем продукции;

t0,t1 – начальное и конечное значения времени t;

T = t1 – t0;

Z(t) – производительность труда.

Производительность можно вычислить при помощи интеграла на промежутке от t до t1. интеграл от Z(t) по dt:

$$W = \int_{t_0}^{t_1} z(t)dt$$

3)Выбор **численного метода** решения задач. Для численного нахождения интеграла существуют различные численные методы, например, метод прямоугольников или трапеций.

4)Составление **алгоритма** решения задач – четко обозначенная последовательность действий по решению конкретной задачи. Каждый алгоритм обладает следующими свойствами:

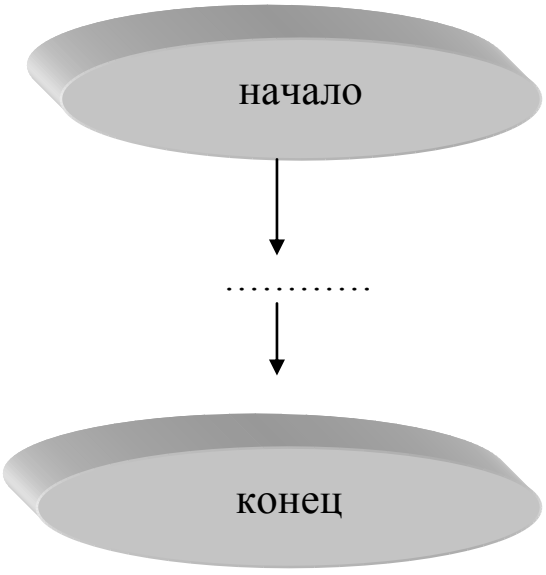
a. Конечность алгоритма, то есть решение задачи должно быть получено за конечное количество операций;

b. Определенность алгоритма (на каждом этапе алгоритма должно быть четко известно, какую операцию надо выполнить и какую операцию надо выполнить дальше.).

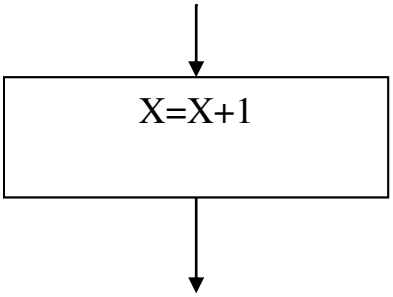
c. Массовость алгоритма (алгоритм должен предложить решение не данной конкретной задачи, а класса подобных задач).

Одним из наиболее распространенных языков записи алгоритмов являются **блок-схемы.** Блок-схема это последовательность геометрических фигур, внутри которых записана операция, а сами фигуры соединены стрелками, показывающими порядок выполнения операций. Простейшими из геометрических фигур являются следующие.

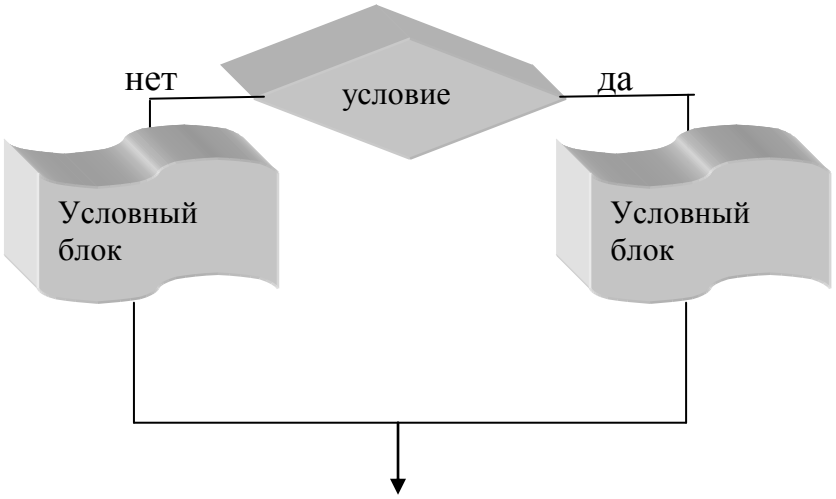
Порядок выполнения операций:



Арифметический блок (или блок присваивания):



Условный блок:



Остановимся подробней на этапе составления алгоритма. Пусть дана задача: решить квадратное уравнение. Очевидно, что её математической моделью будет уравнение вида $ax^2 + bx + c = 0$. Анализ задачи позволяет выявить следующие численные операции для нахождения решения:

a) $x1,x2 = (-b \pm (b^2-4ac)^{1/2})/2a$, $D=b^2-4ac>0$
$a \neq 0$

b) $a \neq 0$, $D<0$ – нет решений

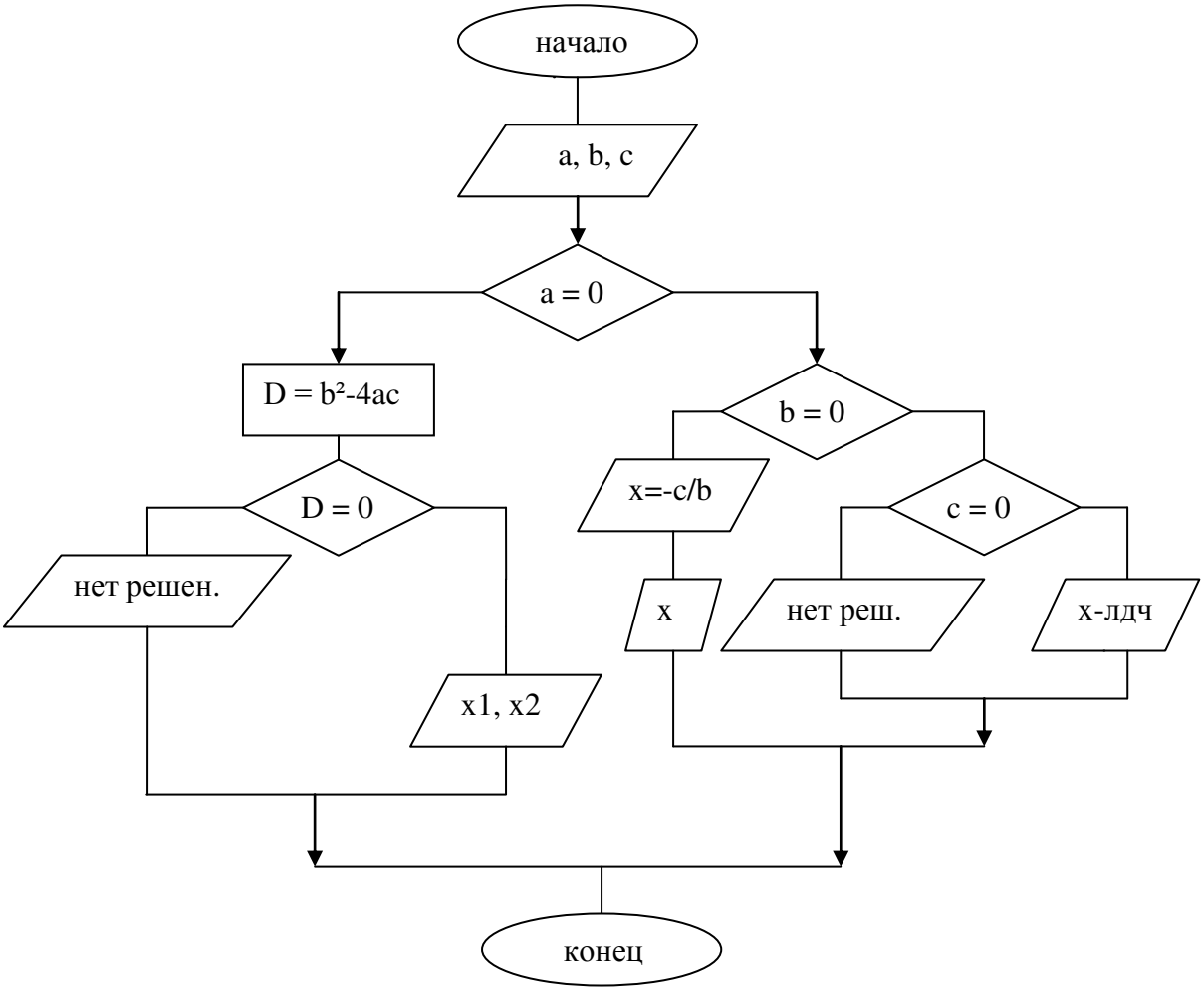
c) $a=0 \Rightarrow bx+c=0 \Rightarrow x= (-c)/b$
$b \neq 0$

d) $a=0$ и $b=0 \Rightarrow c=0$

e) $a=0$, $b=0$ и $c=0$ – x- любое действительное число

f) $a=0$, $b=0$, $c \neq 0$ – нет решений.

Тогда алгоритм решения задачи можно представить в виде следующей блок-схемы:



5) **Программирование** – это запись алгоритма решения задачи на каком-либо языке программирования.

Языки программирования:

1-ЯВУ-языки высокого уровня; приближены к естественным языкам (Бейсик, Паскаль). Их задача: как можно быстрее запрограммировать.

2-ЯНУ-языки низкого уровня; приближены к машинным языкам. Они делятся на: a) МЯ-машинные языки;

b) МОЯ-машинно-ориентированные языки (выполняют машинные команды);

c) ЯСК-языки символического кодирования (это машинные языки, но записывающие с помощью символов).

В настоящее время ЯНУ получили общее название – ассемблеры.

6) **Трансляция**- это перевод с символического на машинный.

символическая команда -> машинная команда

Трансляция ЯВУ: символическую программу заменяют последовательностью машинных команд.

Существует 2 типа трансляторов:

1) интерпретатор: процесс перевода программы по шаговым методам (по одной команде) с непосредственным исполнением только что переведенной

команды. Интерпретаторы совмещают в себе последовательные этапы решения задачи: компоновка и выполнение. После получения результата исходный текст программы остается без изменения. Режим интерпретации используется в работе с учебными программами, так как в этом режиме удобно выполнять еще один этап: отладка.

2) Компилятор. При компиляции исходная символьная программа рассматривается как единое исходное данное для компилятора, а результатом работы компилятора является программа пользователя, записанная в машинном коде (файл с расширеием .obj-объектная программа пользователя)

При трансляции осуществляется поиск синтаксических ошибок в операторах программ. При обнаружении ошибки трансляция прекращается и выдается сообщение об ошибке.

7) Компоновка-это процесс преобразования объектной программы в программу, способную работать в данной вычислительной системе. На этапе компоновки к объектной программе пользователя (obj) присоединяются программы операционной системы, необходимые для работы программного пользователя в данной вычислительной среде. На этом этапе возможно выявление ошибок, связанных, например, с обращением программного пользователя к устройствам или программам, не существующих в данной вычислительной системе.

8) Отладка-это процесс поиска ошибок во время исполнения программы. Для отладки программы обычно используют следующие приемы:

a) создание контрольного примера, просчитанного вручную и сравнение результата работы;

b) если результаты не равны, то поиск ошибки осуществляется методом трассировки;

c) просмотр промежуточных результатов вычисления.

Трассировка-это отслеживание порядка выполнения операторов программ. Этот способ реализуется путем вставки в программу промежуточных операторов печати. Существуют автоматические способы трассировки.

При отладке могут быть обнаружены ошибки 2-х типов:

-ошибки алгоритма;

-ошибки вычисления.

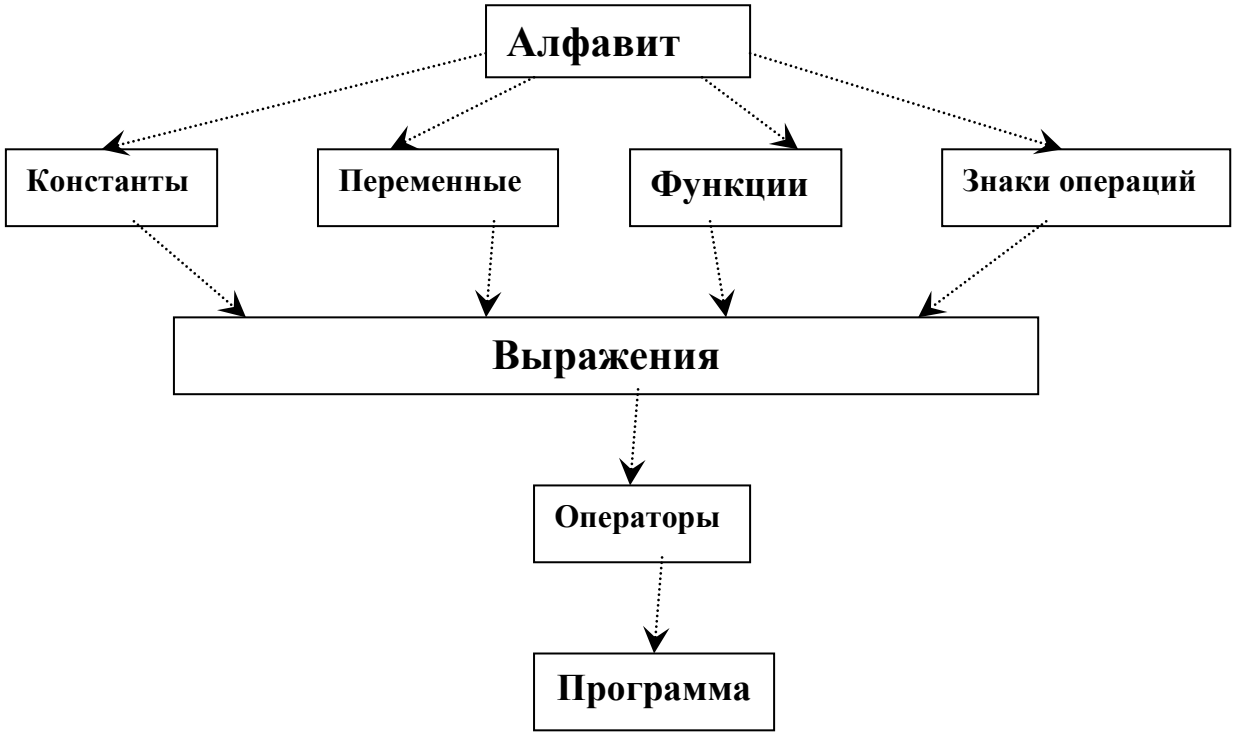
При обнаружении ошибки возможно вернуться на любой из предыдущих этапов.

9) Выполнение.

На этом этапе так же могут быть обнаружены ошибки. Как правило, существует договоренность между программистом и заказчиком по обслуживанию программы и устранению ошибок.

Общее описание языков программирования.

Содержание этого параграфа будет излагаться в соответствии со следующей схемой:



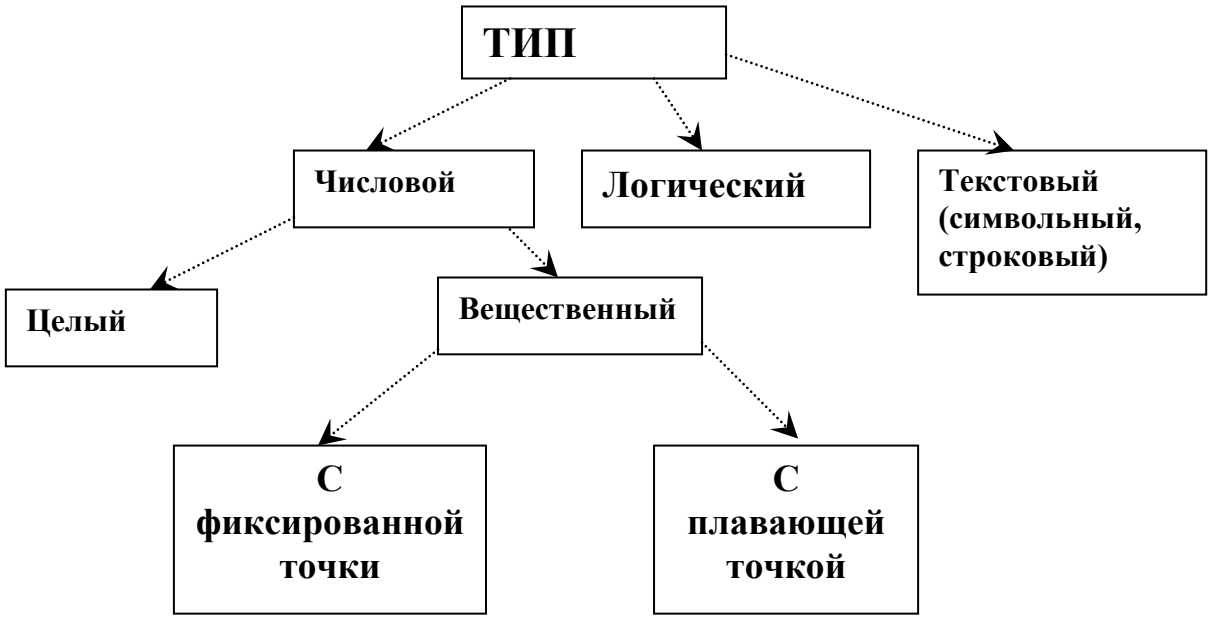
Алфавит

Алфавит- это совокупность объектов, являющих собой наименьшую единицу информации в программе.

1. Буквенно-цифровые символы (рус., лат.).
2. Знаки препинания
3. Ключевые слова (зарезервированные слова - это те неделимые конструкции, из которых строятся программные объекты), например, if, for, next и другие. Эти слова не могут служить никакими другими объектами

Описание данных

Типы данных



Это величина, которая не изменяет своего значения во время работы программы. Каждая константа обладает двумя характеристиками: значение и тип.

Типы констант можно разделить на три части: числовой логический и текстовый. В свою очередь числовой тип делится на целый, вещественный с фиксированной точкой и вещественный с плавающей точкой.

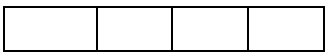
Значения константы определены формой её записи в программе. Например,

1 1.
целый вещественный
тип тип

Правила записи целых и вещественных констант.

[+-] <Последовательность цифр>

Целый тип определяет способ представления целого числа.

 - 32 бита, 4 байта

 - 16 битов, 2 байта

Диапазон для 2-х байтов от -2^{15} до $+2^{15}-1$ (-32768 до 32767)

32 32

Способ программирования создаёт диапазон от -2 до 2 - 1 - диапазон для 32 бит.

1. - число с фиксированной точкой.

Общий формат записи числа с фиксированной точкой.

[+-] [<Последовательность цифр>].[<цифры>]

-.1= - 0.1

Константы с плавающей точкой.

27

1.32*10

-12

0.1 * 10 -**порядок**

манн-

тисса

Общая форма записи числа с плавающей точкой.

<число с фиксированной точкой> E <целое число>

-12

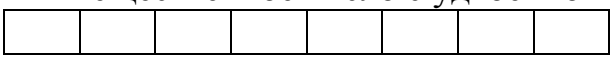
+.1 E - 12 + 0.1*10

E - экспонента

Представление этих чисел в компьютере.

Вещественные числа:

- вещественное число
- вещественное число с удвоенной точностью(8 байт)



порядок м а н т и с с а 32

Диапазон записи в компьютере до **|10|**

Логический тип

Логика работает с высказываниями, утверждениями, о которых можно говорить, истинны они или ложны. В Basic в качестве логического используется числовой тип: «истина» =1 , «ложь» = 2. В Pascal константы логического типа определены как «истина» = **true**, «ложь» = false.

Текстовый тип

Это последовательность любых символов алфавита, заключённая в Basic в двойные кавычки, в Pascal в апострофы. Пустая строка не содержит ни одного символа и записывается как две двойные кавычки в Бейсике и два апострофа в Паскале.

Переменные

Это величины, которые во время работы программы могут изменять своё значение. Всякая переменная обладает 3 характеристиками:

- Имя (идентификатор)

через имя мы наблюдаем переменную в программе. Имя переменных - это последовательность букв и цифр, начинающихся с буквы - правило записи имени.

В программе не может быть двух разных переменных с одним именем.

- Значение. Получает во время работы программы. Только 2 оператора, которые позволяют придавать значение: оператор ввода и оператор присваивания.
- Тип

Способы задания типа в Basic:

- по умолчанию

% в конце имени - переменная целого типа

$ в конце имени - переменная текстового типа

Числовая вещественная, если в конце ничего не стоит

Существует свой набор операций и при попытке выполнения операций над переменными не собственного типа транслятор выдаёт сообщение об ошибке.

- Помимо задания типа по умолчанию в Basic существуют другие операторы явного задания.

В языках программирования существует сложный тип переменной - переменная с индексом, являющая собой отдельный элемент сложного типа данных под названием массив.

Массив – это конечная упорядоченная совокупность переменных одного типа.

Что значит ***упорядоченная***? Это значит, что каждый элемент массива находится на своём месте, т.е. если 2 элемента массива поменять местами, то это будет уже другой массив. Величина, определяющая местоположение переменной в массиве, называется **индексом.** Иначе говоря, индекс – это номер элемента (переменной) в массиве.

Массив считается данным сложного типа и требует обязательного **описания.** В Basic **описание** массива выполняется с помощью оператора **DIM** (dimension), который относится к т.н. *описательным* или *невыполняемым* операторам. Формат записи такого оператора:

DIM <имя массива**> (<**максимальные размеры индексов>**)**

При выполнении оператора DIM в памяти компьютера выделяется место для хранения элементов массива в указанном объёме. На выделение объёма памяти для массивов существуют ограничения, заданные системой программирования.

Например: DIM A(10) , DIM Z(5,3,7)

Массивы могут иметь несколько индексов и бывают **одномерные** (или вектора)**, двумерные** (или матрицы)**, трёхмерные и т.д. – многомерные.**

В Бэйсике, если массив **одномерный,** то в нем максимальное значение индекса равно кол-ву элементов в массиве. В **двумерном** массиве кол-во элементов равно произведению максимальных значений индексов. Например в матрице **DIM** B(4,5) кол-во элементов равно 4*5 = 20 элементов. В памяти компьютера все элементы всех массивов расположены линейно. Это значит, что элементы матрицы $\begin{pmatrix} 5 & 0 & 2 \\ 1 & 6 & 3 \end{pmatrix}$ будут расположены в порядке 5, 0, 2, 1, 6, 3, то есть по строчкам. Говорят, что элементы многомерных массивов расположены *в порядке наибыстрейшего изменения последнего индекса*. Индекс может изменяться от начального значения до максимального значения, указанного в операторе DIM. В качестве индекса может выступать **переменная, например,** A(i) или даже выражение - A(k+1). Использование переменных в качестве индексов обеспечило максимальную эффективность в обработке массивов.

В Pascalе описание массива задается в специальном разделе программы и имеет следующий формат:

<имя массива>**: array** <мин. индекс>**..**<макс. индекс> **of** <тип>**;**

Количество элементов в массивах в Pascal определяется как произведение разностей макс. и мин. значений индексов плюс 1 .

В качестве значений индексов мы будем использовать величины **целого типа** (хотя в Pascal возможны и другие типы индексов).

Функции.

Функция – это некоторая программа (или модуль), составленная в соответствии с правилами системы программирования. Результатом работы функции является некоторая величина. Функции бывают стандартными и нестандартными

Стандартной функцией называется программа (модуль), разработанная в рамках системы программирования и поставляемая вместе с ней. Они, как правило, реализуют некоторые из наиболее часто встречающихся математические функции. Стандартная функция имеет имя, по которому пользователь может обратиться к соответствующей функции для получения результата, указав при этом, если требуется, аргумент функции.

Таблица основных стандартных функций		
Y=f(x)	**Basic**	**Pascal**
$y = \|x\|$	ABS(x)	Abs(x)
$y = x^2$	----	Sqr(x)
$y = \sqrt{x}$	SQR(x)	Sqrt(x)
y = e^x	EXP(X)	Exp(x)
y=ln x	LOG(x)	Ln(x)
y = sin x	sin(x)	sin(x)
y = cos x	cos(x)	cos(x)
y = arctg x	atn(x)	arctan(x)
Случайное число от 0 до 1	RND	Random
Ближайшее целое, меньшее x	INT(X)	---
Целая часть x	---	Int(x)

Это далеко не полный список стандартных функций. В книгах всегда есть приложения, в которых приводятся таблицы стандартных функций, содержащие их полный список.

Системы программирования и для Бэйсика, и для Паскаля содержат средства для создания собственных, пользовательских или нестандартных функций, о чем речь пойдет ниже.

Выражения.

Выражение – это последовательность констант, переменных и функций соединённых знаками операций. Тип и значение данных, используемых в выражении, определяют тип выражения в целом.

Арифметические выражения

АВ формируется из данных числового типа и соответствующих знаков операций. Так как числовой тип может быть как вещественным, так и целым, то знаки операций так же делятся на вещественные и целые. К вещественным операциям и в Бэйсике, и в Паскале относятся "+" - сложение, "-" - вычитание, "*" – умножение и "/" – деление. Кроме того, в Бэйсике имеется операция возведения в степень – "^". К операциям целого типа относятся те же операции, кроме деления, а так же имеются операция получения частного от деления двух целых чисел (в Бэйсике - “\”, в Паскале – "div") и операция получения остатка от деления двух целых чисел – "mod" и в Бэйсике, и в Паскале (например, 17 mod 7, результат равен 3).

При написании выражений нужно следовать определенным правилам системы программирования. Так в некоторых случаях запрещено смешивать вещественные операнды и целые операции и наоборот. Особенно строг в этом отношении Паскаль. Такие ошибки легко исправляются, поскольку сообщения о них выдаются на этапе трансляции программы.

Тип выражения в целом определяется полученным значением, причём если хотя бы один из **операндов** (величины, участвующие в операциях) имеет вещественный тип, то результат так же имеет вещественный тип.

При написании арифметических выражений следует использовать скобки, которые наилучшим образом определяют порядок выполнения операций в выражении:

1. Вычисления начинаются с самых внутренних скобок.
2. Вычисляются значения функций.
3. Возведение в степень.
4. Операции умножения и деления в порядке записи.
5. Сложение и вычитание в порядке записи

Лекции 4. Определение типов и описание переменных. Ввод-вывод данных.

1. ***Общие сведения о языке.***

Язык Pascal (Паскаль) был разработан в 70-х годах 20-го века профессором Швейцарского федерального института технологии в Цюрихе Никалаусом Виртом. Этот язык получил свое название в честь великого французского математика и физика Блеза Паскаля, который в 1642 году изобрел счетную машину для арифметических операций – паскалево колесо. Язык разрабатывался, прежде всего, для целей

обучения программированию, но этим не ограничена сфера его применения. Основными достоинствами Паскаля являются легкость при изучении и наглядность программ. Кроме того, в языке Паскаль отражена концепция структурного программирования, имеется богатый набор различных типов данных и программные средства, позволяющие доказывать правильность написанных программ.
В 1984 году на рынке программных продуктов появляется интегрированная система программирования Turbo Pascal. До этих пор предпочтение отдавалось языку программирования Basic – простому, дешевому и легко усваиваемому. Паскаль же был языком для избранных – аппаратно зависимым, дорогим и сложным в обращении. С появлением среды Turbo Pascal положение резко изменилось. Начиная с 6-ой версии, система Turbo Pascal обладает такими качествами, как:

- многооконный и многофайловый режим работы;
- встроенные средства верификации данных и программ;
- режимы трассировки;
- поддержка работы манипулятора мышь;
- применение объектно-ориентированного программирования;
- наличие большой библиотеки модулей и т.п.

В 1992 году фирма Borland International выпустила 2 пакета программирования на языке Паскаль: Borland Pascal 7.0 и Turbo Pascal 7.0.

2. ***Общие правила записи программы***

При записи программы на языке программирования Паскаль необходимо придерживаться некоторых общепринятых соглашений:

1) служебные слова, значение которых зарезервировано раз и навсегда, нельзя использовать ни для каких других целей (для обозначения имен переменных, для заголовка программы, для имени процедуры или функции, определенной пользователем и т.п.);
2) заглавные и прописные буквы не различаются;
3) буквы русского алфавита могут быть использованы только в комментариях или в строковых выражениях.

Общая структура паскаль-программы может быть представлена в виде следующей схемы:

```
Program <имя>
  var <раздел описания переменных>;
Begin
  <раздел операторов>
```

End.

3. ***Основные элементы языка.***

Любой язык программирования есть совокупность трех составляющих: набора символов (*алфавита*), правил образования (*синтаксис*) и истолкования (*семантика*) конструкций из символов для задания алгоритмов.

Алфавит.

Основные элементы, составляющие алфавит языка Паскаль, можно разбить на несколько групп в соответствии с их смысловыми значениями:

- *основные символы* (буквы латинского и русского алфавитов, цифры, знаки арифметических операций, знаки отношений, кавычки, точка, запятая, точка с запятой, двоеточие, апостроф, скобки и т.п.);
- *служебные слова* (and, array, begin, case и т.п.);
- *стандартные идентификаторы* (true, false, integer, sin, abs и т.п.).

Данные.

В разделе операторов программы команды определяют, какие действия должны быть выполнены над данными. С каждым элементом данных обычно связаны *имя* и *значение.*

Имя используется для обозначения элемента данных. В качестве имени в языке Паскаль используется идентификатор – последовательность букв латинского алфавита и цифр, начинающаяся с буквы. Все идентификаторы, используемые в программе, обязательно должны быть определены, т.е. описаны в соответствующем разделе программы.

В качестве данных в паскаль-программе могут использоваться постоянные и *переменные* величины. Значение постоянной величины (*константы*) не изменяется в процессе выполнения программы.

Значение элемента данных определяется типом данных. В языке Паскаль существует 4 стандартных типа данных:

1) *integer – целый*

значения этого типа данных лежат в диапазоне от -32768 до 32767.

Целые числа в языке Паскаль записываются в обычном виде. Над ними определены следующие операции: сложение (+), вычитание (-), умножение (*), деление (/), нахождение целой части от деления (div) и остатка от деления (mod). Результат всех операций, кроме деления, целое число.

К типу integer относятся некоторые стандартные функции: нахождение

предыдущего элемента в множестве (pred), нахождение последующего элемента (succ), модуль числа (abs), квадрат числа (sqr) и др.
Существует несколько ограничений и расширений целого типа:

shortint (-128..127); longint (- 2 млрд. .. 2 млрд.); byte (0.. 255); word (0..65535).

2) *real – вещественный*

В языке Паскаль существует два способа записи вещественный чисел:

- с фиксированной десятичной точкой;
- с плавающей десятичной точкой (в экспоненциальной форме) вида mEn, т.е. $m*10^n$.

Например: 314E-2; 2.76E11; .5E-3
Операции +, -, *. / выполняются с принятой в данном компьютере точностью и не всегда дают точный результат.

3) *Boolean – логический или булевский*

Множество значений данного типа – это {true, false}.
В языке паскаль определены 6 операций отношения (<, >, <=, >=, =, <>) и логические операции: отрицание (not), логическое умножение – конъюнкция (and), логическое сложение – дизъюнкция (or). С помощью этих операций строятся выражения отношения, значения которых относятся к логическому типу данных.

4) *char – символьный*

Значением переменной символьного типа является символ. Множество символов отвечает следующим требованиям:

- любой символ имеет свой порядковый номер;
- порядковые номера цифр 0..9 упорядочены по возрастанию и следуют друг за другом;
- порядковые номера букв также упорядочены по возрастанию, но не обязательно следуют друг за другом.

Константы символьного типа записываются в языке Паскаль в виде символов, заключенных в апострофы. Например: 'a', '1', 'z', 'ш'.
Над данными символьного типа определены только операции отношения.
К символьному типу относятся некоторые стандартные функции языка Паскаль: нахождение порядкового номера символа (ord), нахождение символа по его порядковому номеру (chr).

4. Структура программы на языке Паскаль.

Program <имя программы>
  uses <раздел подключения внешних модулей>;
  label <раздел описания меток>;
  const <раздел описания констант>;
  type <раздел описания типов>;
  var <раздел описания переменных>;
  procedure <раздел описания пользовательских процедур>;
  function <раздел описания пользовательских функций>;
Begin
  <раздел операторов или основной блок программы>
End.

Метки перечисляются обычным списком, константы и типы описываются в стиле: <идентификатор>=<значение>, переменные перечисляются с указанием своих типов: <имя>:<тип>.

Комментарии к программе служат для пояснения и представляют собой обычный текст, заключенный в фигурные скобки.

Ввод - вывод. Операторы Read (Readln), Write (Writeln). Простейшие линейные программы

Решим задачу, прокомментировав каждое свое действие в фигурных скобках. Напомним, что комментарий не воспринимается компьютером, а нам он нужен для того, чтобы лучше понять как работает программа.

Задача. Напишите программу, которая бы очищала экран и вычисляла произведение двух чисел, вводимых пользователем.

```
Program Proizv2;
Uses
  Crt;{Подключаем модуль Crt}
Var
  number1, {переменная, в которой будет содержаться первое число}
  number2, {переменная, в которой будет содержаться второе число}
  rezult {переменная, в которой будет содержаться результат}
    : integer;
Begin
  ClrScr;{Используем процедуру очистки экрана из модуля Crt}
  Write ('Введите первое число ');
  {Выводим на экран символы, записанные между апострофами}
  Readln (number1);
  {Введенное пользователем число считываем в переменную number1}
  Write ('Введите второе число ');
  {Выводим на экран символы, записанные между апострофами}
  Readln (number2);
```

```
    {Введенное пользователем число считываем в переменную number2}
    rezult := number1 * number2;
    {Находим произведение введенных чисел и присваиваем переменной rezult}
    Write ('Произведение чисел ', number1, ' и ', number2, ' равно ', rezult);
    {Выводим на экран строчку, содержащую ответ задачи}
    Readln;{Процедура задержки экрана}
End.
```

Чтобы лучше понять действие программы, наберите ее на компьютере и проверьте ее действие. Ответьте на вопросы:

- почему программу назвали Proizv2?
- зачем в раздел Uses поместили модуль Crt?
- какое назначение переменных number1, number2, rezult?
- какой тип у этих переменных? что это значит?
- если присвоить переменным number1 и number2 соответственно значение 5 и 7, то какую строчку выдаст компьютер при исполнении последней процедуры Write? Запишите ее в тетрадь.
- в каких строчках у пользователя запрашиваются значения переменных?
- в какой строчке происходит умножение чисел?
- что делает оператор присваивания в этой программе?

Задание. Измените программу так, чтобы она запрашивала у пользователя еще одну переменную и выводила результат произведения трех чисел.

Операторы Write и WriteLn

Мы уже использовали операторы Write и WriteLn, но нам необходимо подробнее остановиться на правилах применения этих операторов.

Write (англ. писать) – оператор, который используется для вывода информации на экран. Оператор WriteLn выполняет то же самое действие, но так как у него есть еще окончание Ln (line - англ. линия, строка), то после вывода на экран нужного сообщения, он дополнительно переводит курсор на следующую строчку.

Общий вид:

```
Write (список выражений)
WriteLn (список выражений)
```

Процедуры Write и WriteLn используются не только для вывода результата, но и для вывода различных сообщений или запросов. Это позволяет вести диалог с пользователем, сообщать ему, когда ему нужно ввести значения , когда он получает результат, когда он ошибся и др.

Например, при выполнении процедуры WriteLn(‘Найденное число ‘,a), будет напечатана строчка, заключенная в апострофы, а затем выведено значение переменной a.

Оператор WriteLn можно применить и без параметров. В этом случае напечатается строка, состоящая из пробелов, и курсор будет переведен на

другую строку. Это иногда нам нужно для лучшего восприятия ввода данных.

Операторы Read и ReadLn

Вспомним, что основное назначение ЭВМ – сэкономить человеческий труд. Поэтому необходимо обеспечить возможность, однажды написав программу, многократно ее использовать, вводя каждый раз другие данные. Такая гибкость в языке обеспечивается операторами Read и ReadLn. Этими операторами вводится информация с клавиатуры.

Общий вид:

Read(переменная, переменная...)

ReadLn(переменная, переменная...)

При выполнении процедуры Read ожидается ввод перечисленных в скобках значений. Вводимые данные нужно отделить друг от друга пробелами. Присваивание значений идет по очереди.

Например, если вводятся значения 53 и X, то при выполнении оператора Read(a,b) переменной a будет присвоено число 53, а переменной X – буква X. Причем, отметим, чтобы не было аварийной ситуации, нужно правильно определить тип данных в разделе Var; в нашем случае a:integer, a b:char.

Особых различий при чтении и записи в использовании операторов Read и ReadLn нет. Часто процедуру ReadLn без параметров применяют в конце программы для задержки: до нажатия на клавишу <Enter> результат выполнения программы остается на экране. Это очень полезно делать для анализа результатов.

Лекции 5. Сложные операторы. Организация ветвлений.

1. Простые и составные операторы.

Простые операторы.

1) Оператор присваивания.

Синтаксис:

<имя переменной>:=<выражение>

Выполнение: вычисляется значение выражения, стоящего в правой части оператора, и полученное значение присваивается переменной, имя которой указано в левой части оператора.

При вычислении значения выражения необходимо учитывать тип данных и операции, которые над ними выполняются. Если хотя бы один элемент данных, входящих в выражение, относится к вещественному типу или же в выражении встречается операция деления, то результат будет принадлежать вещественному типу данных.

2) *Вызов процедуры.*

Синтаксис:
<имя процедуры>[(<список значений параметров>)];
Выполнение: происходит переход к стандартной или определенной пользователем процедуре с неявным присваиванием фактических значений формальным параметрам процедуры.
Организовать ввод и вывод данных в языке Паскаль можно с помощью стандартных процедур read, readln, write, writeln.
Ввод/вывод данных всегда связан с обменом информацией между оперативной памятью и внешними носителями информации, в качестве которых могут выступать как файлы так и консольные устройства ввода/вывода (клавиатура, дисплей, принтер и т.д.).
Синтаксис:
read([<имя устройства ввода>,]<список имен переменных>);
readln([<имя устройства ввода>,]<список имен переменных>);
write([<имя устройства вывода>,]<список значений>);
writeln([<имя устройства вывода>,]<список значений>);
Выполнение:
ввод данных: с устройства ввода последовательно считываются значения и присваиваются переменным, имена которых указаны в списке. При этом, необходимо следить за тем, чтобы совпадали типы у переменных и присваиваемых им значений (нельзя присвоить символьное значение вещественной переменной и т.п.);
вывод данных: значения, указанные в списке, последовательно выводятся на устройство вывода данных.
Отличие процедур read и readln, write и writeln состоит в том, что при выполнении процедур writeln и readln к последнему значению автоматически дописывается управляющий символ конца строки, что означает переход на новую строку экрана или файла.

Составные операторы.

Составной оператор представляет собой последовательность операторов, заключенных в операторные скобки begin и end.

Выполнение составного оператора заключается в том, что один за другим выполняются операторы, указанные внутри операторных скобок, в той последовательности, как они записаны, до тех пор, пока не будет полностью исчерпана вся последовательность.

2. ***Структурные операторы.***

1) *Условный оператор*

Синтаксис:
if <условие>
then <оператор>
[else <оператор>];

где <условие> - выражение булевского типа;
<оператор> - любой оператор языка Паскаль (в том числе и составной).

Выполнение: вычисляется значение выражения, стоящего в условии. Если это значение истинно, то выполняется оператор, следующий за служебным словом then. Если условие ложно, то выполняется оператор, стоящий за служебным словом else.
Условный оператор может использоваться и в сокращенной форме, когда отсутствует часть оператора, начиная со служебного слова else. В этом случае оператор выполняется при истинном условии, а если условие ложно, то происходит переход к оператору, следующему за условным. Например:

```
if x>y        |  if x>y
then          |  then
  begin       |    x:=0;
    x:=0;     |    y:=0;
    y:=0;
  end;
```

В условном операторе за служебными словами then и else могут следовать любые операторы языка Паскаль, в том числе и условные. Поэтому возможны «вложенные» условные операторы. Например,

```
if x>y
then x:=x-y
else if x=y
  then x:=0
  else y:=y-x;
```

Условие может быть составным. В этом случае каждая его часть заключается в скобки. Например:

```
if (x>5) and (x<15)
if (a=b) or (not(t))
```

2) Оператор варианта

С помощью условного оператора осуществляется выбор одного из двух возможных действий в зависимости от значения булевского выражения. Однако, часто бывает необходимо выбрать одно из нескольких (больше двух) действий. Для реализации такого рода ситуаций в языке Паскаль предусмотрен специальный оператор варианта.

Синтаксис:
case <выражение> of

<значение 1>: <оператор>;
…
<значение n>: <оператор>;
[else <оператор>]

end;

где <выражение> - выражение любого скалярного типа (кроме real);
<оператор> - любой оператор языка Паскаль.

Выполнение: вычисляется значение выражения, стоящего после case. Это значение используется для выбора одного из возможных действий. Если полученное значение выражения совпадает с одним из перечисленных после of, то выполняется соответствующий оператор. Если один и тот же оператор должен выполняться при различных значениях выражения, то эти значения могут быть указаны через запятые или в виде интервала. Если в операторе присутствует часть else, то оператор, указанный за данным служебным словом, выполняется тогда, когда значение выражения не совпадает ни с одним из значений, перечисленных после of. Например:

Пусть m – номер месяца, y – порядковый номер года. Определить d – количество дней в соответствующем месяце.

```
case m of

  1,3,5,7,8,10,12: d:=31;
  4,6,9,11: d:=30;
  2: if (y mod 4=0) and (y mod 100 <>0)
     then d:=29
     else d:=28
end;
```

Лекции 6. Операторы цикла и принципы их использования.

Структурные операторы.

Оператор цикла

Оператор цикла используется для организации многократного повторения выполнения одних и тех же операторов. В языке Паскаль существует три типа оператора цикла.

а) оператор цикла с предусловием

Синтаксис:

while <условие> do

<оператор>;

где <условие> - булевское выражение;
<оператор> - любой оператор языка Паскаль, называемый «телом цикла».
Выполнение: вычисляется значение выражения, стоящего в условии. Если это значение истинно, то выполняется оператор и снова происходит возврат к вычислению значения булевского выражения. Как только условие станет ложным, выполнение цикла завершается и выполняется оператор, следующий за телом цикла.
Оператор цикла с предусловием может быть не выполнен ни разу в том случае, если при первом же вычислении значения условия оно будет ложным.
Замечание.
Оператор, стоящий в теле цикла, обязательно должен изменять значение условия, иначе цикл будет выполняться бесконечное число раз.
б) оператор цикла с постусловием
Синтаксис:
repeat

<оператор>

until <условие>;

где <условие> - булевское выражение;
<оператор> - любой оператор языка Паскаль.
Выполнение: выполняется оператор, стоящий в теле цикла, затем вычисляется значение выражения, стоящего в условии. Если это значение ложно, то происходит переход к выполнению оператора, стоящего в теле цикла. Как только условие станет истинным, выполнение цикла завершается.
Для оператора цикла с постусловием справедливо замечание, сделанное выше.
Отличие циклов с предусловием и с постусловием:

- при выполнении цикла с предусловием тело цикла выполняется, если условие истинно, а в цикле с постусловием повторение осуществляется, если значение условия ложно;
- тело цикла с постусловием обязательно будет выполнено хотя бы один раз независимо от значения условия, тогда как тело цикла с предусловием может быть не выполнено ни разу;
- если в теле цикла с предусловием используется составной оператор, то

он заключается в операторные скобки. При использовании составного оператора в теле цикла с постусловием роль операторных скобок выполняют служебные слова repeat и until.

в) оператор цикла с параметром

Оператор цикла с параметром целесообразно применять тогда, когда заранее известно количество повторений цикла. Оператор цикла с параметром используется в двух видах:

1. *Синтаксис*:

for <имя перем.>:= <выражение 1> to <выражение 2> do

<оператор>;

где <имя переменной> - идентификатор переменной целого типа, называемой параметром цикла;

<выражение 1> и <выражение 2> - выражения, задающие начальное и конечное значения параметра цикла;

<оператор> - любой оператор языка Паскаль.

Для объяснения выполнения обозначим:

i – имя переменной, z1 и z2 – выражения 1 и 2 соответственно, s – оператор, образующий тело цикла.

Тогда цикл имеет вид:

for i:= z1 to z2 do s;

Выполнение: вычисляются значения выражений z1 и z2. Если z1>z2, то выполнение оператора цикла завершается. Если z1<=z2, то переменной i присваивается значение z1 и выполняется оператор s. Затем значение переменной I увеличивается на 1 и вычисляется значение булевского выражения i<=z2. Если это значение истинно, то вновь выполняется оператор s, берется следующее значение параметра цикла и т.д. Выполнение оператора цикла с параметром завершается тогда, когда станет ложным значение булевского выражения i<=z2.

2. *Синтаксис*:

for <имя перем.>:= <выражение 1> downto <выражение 2> do

<оператор>;

В обозначениях из предыдущего раздела оператор цикла принимает следующий вид:

for i:= z1 downto z2 do s;

В этом случае z1>z2 и значение параметра цикла при каждом повторении цикла не увеличивается, а уменьшается на 1.

Выполнение: вычисляются значения выражений z1 и z2. Если z1<z2, то выполнение оператора цикла завершается. Если z1>=z2, то переменной i присваивается значение z1 и выполняется оператор s. Затем значение переменной i уменьшается на 1 и вычисляется значение булевского

выражения i>=z2. Если это значение истинно, то вновь выполняется оператор s, берется следующее значение параметра цикла и т.д. Выполнение оператора цикла с параметром завершается тогда, когда станет ложным значение булевского выражения i>=z2.

Замечание.

После завершения выполнения оператора цикла значение параметра цикла не определено.

Примеры использования структурных операторов.

1. Вычислить значение факториала числа.

```
Program factorial;
  uses crt;
  var n, i:integer; p:longint;
Begin
  clrscr;
  readln(n);
  p:=1;
  for i:=1 to n do
    p:=p*i;
  write(n,'! = ', p);
End.
```

2. Вычислить наибольший общий делитель двух целых чисел a и b.

```
Program NOD;
  uses crt;
  var a,b: integer;
Begin
  clrscr;
  readln(a,b);
  a1:=a;
  b1:=b;
  while a<>b do
    if a>b then a:=a-b
    else b:=b-a;
  writeln('НОД(',a1,',',b1,') = ',a);
  readln;
End.
```

3. Вычислить наименьшее из 20 целых чисел.

```
Program min_of_20;
  uses crt;
  var n, min, i: integer;
Begin
```

```
  clrscr;
  writeln('Введите первое число');
  readln(n);
  min:=n;
  for i:=2 to 20 do
  begin
    writeln('Введите ',i , '-е число');
    readln(n);
    if n<min then min:=n
  end;
  writeln('min = ', min);
  readln;
End.
```

Лекции 7. Составные типы. Массив, запись и файл.

Определение массива.

В ряде случаев приходится сталкиваться с ситуацией, когда должно использоваться относительно много переменных одного типа.

Представим себе программу, которая вычисляет метеорологические данные (температура, влажность, давление, осадки и т.п.) за год. Т.е. в этой программе всегда будет не меньше 365 переменных. Можно описать эти 365 (366) переменных следующим образом:

```
var day1, day2, day3, …, day365: real;
```

но уже только одна такая запись будет занимать больше страницы, не говоря о том, чтобы посчитать, например, среднюю температуру за год.

Для таких и подобных этому случаев в языке Паскаль предусмотрена возможность введения большого числа переменных одного и того же типа за счет использования структурированного типа массив. Следует отметить, что подобный тип данных есть практически во всех языках программирования.

Массив – это упорядоченный набор перенумерованных элементов одного типа.

Элементы массива характеризуются следующими *свойствами*:

1) все элементы упорядочены;
2) обращение к любому элементу массива осуществляется по его индексу;
3) количество элементов определяется один раз при описании массива и далее не может быть превышено;
4) все элементы массива принадлежат к одному типу данных, называемому ***базовым типом массива***.

Все элементы массива имеют общее имя – ***имя массива***, а указание на конкретный элемент осуществляется с помощью ***индекса*** – порядкового номера элемента.

Для описания объекта типа массив в языке Паскаль используется следующая конструкция:

array [<тип1>] of <тип2>;

где тип1 – тип индекса;

тип2 – тип элементов (базовый тип).

Типы индексов.

В качестве типа индекса могут быть использованы такие типы, как:

Ограниченный.

Пример.

array [1..n] of integer;

array [1146..2004] of real;

array [-754..-1] of integer;

В этом случае обращение к элементам массива осуществляется через указание имени массива и в квадратных скобках его индекса – любого числа из представленного интервала.

Пример.

При описании следующего вида

type tula = array [1146..2004] of real;

var stat: tula;

возможны обращения к элементам массива:

stat [1675] stat [1980] stat [2000]

Описать массив можно и другим способом:

var stat1: array [2000..2004] of real;

Перечислимый.

Пример.

type m = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);

var t: array [m] of integer;

Тогда обращение к элементам массива будет следующим:

t[Jan] t[Oct]

т.е. в качестве индекса выступает любое значение из определенного ранее набора.

3. Булевский.

В этом случае индексные выражения могут представлять собой логические выражения.

Пример.

var k: array [boolean] of integer; a,b: boolean;

Обращение к элементам массива может быть следующим:

k[false] k[true] k[a or b]

4. Символьный.

Пример.

type s = array [char] of integer;

var m: s; c: char;

К элементам массива можно обратиться следующим образом:

m['d'] m['z'] m[pred(c)]

В качестве типа индекса не могут быть использованы типы integer и real, т.к. множества значений этих типов в языке Паскаль не ограничены (тип integer ограничен лишь возможностями техники), а real еще и не упорядочено.

При объявлении типа индексов массива удобно использовать константы:

const n = 10;

type m = array[1..n] of integer;

Размерность массива.

Количество индексов в обозначении элемента массива определяет ***размерность*** массива. Массив может быть одномерным, двумерным, трехмерным и т.д. При описании таких массивов должен быть описан тип каждого индекса.

Пример.

var m: array[1..10, 1..15] of real;

m[5,8] m[1,10]

В этом случае мы имеем дело с прямоугольной таблицей, состоящей из 15 столбцов и 10 строк.

В языке Паскаль число размерностей неограниченно. Например, выражение

var a: array[1..1000, 1..35, 1..20, 1..70] of real;

совершенно правильно с точки зрения синтаксиса языка, хотя можно сомневаться, хватит ли у компьютера памяти для такого количества переменных.

Замечание.

Размерность массивов на практике ограничена размером памяти компьютера.

4. Организация работы с массивами.

Для организации работы с массивом необходимо:

1) *объявить* (или описать) массив, т.е. задать имя, типы индексов и элементов;

2) *сгенерировать* (или сформировать) массив, т.е. задать значения элементов массива. Среди способов генерации массивов можно

выделить следующие: с клавиатуры, из памяти, с помощью датчика случайных чисел, по какому-либо закону (формуле) и др.;
3) *визуализировать* массив, т.е. вывести на экран первоначальные значения элементов массива;
4) *обработать* элементы массива в соответствии с задачей;
5) вывести на экран *результат* выполненных действий.
Действия по формированию и визуализации массивов удобнее оформить в виде отдельных *процедур*.

5. Массивы констант.

В объявлении массива констант указываются значения его элементов, заключенные в круглые скобки и отделены друг от друга запятыми.
Пример.
type color = (Red, Blue, Green);

```
palette = array [color] of string[10];
const m: palette = ('Красный', 'Синий', 'Зеленый');
```

В результате элементы массива m приобретают следующие значения:
m[Red] = 'Красный';
m[Blue] = 'Синий';
m[Green] = 'Зеленый';
Элементы такого массива удобно задавать не как одиночные символы, а в виде строковой константы. Например, объявление константы
const digits: array [0..9] of char = ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')

можно представить в более компактной форме:
const digits: array [0..9] of char = '0123456789';
Чтобы обратиться к элементу константы типа массив необходимо указать ее имя и затем в квадратных скобках – требуемое число индексов, которые могут задаваться выражениями соответствующих типов.

Файловый тип данных

Общее понятие о файле.

Типы данных, с которыми мы до сих пор имели дело, предназначены для манипулирования информацией, содержащейся в оперативной памяти компьютера. Однако, часто возникает необходимость сохранения этих результатов с целью их постоянного использования. Для долговременного хранения информация из оперативной памяти переносится в файлы.

Файл представляет собой некоторое поименованное место на внешнем носителе**.**

Кроме "долговременности" хранения у файлов имеется еще одна отличительная особенность: их неопределенный объем (или длина). Если для каждого из прочих структурированных типов (например, массивов) всегда точно определено, сколько элементов содержит та или иная структура, то, сколько элементов должно быть в файле, при объявлении файлового типа не указывается. *Максимальная длина файла ограничивается только свободным местом на диске*, и это является основным отличием файлов от массивов.

Для хранения информации в Turbo Pascal предусмотрена возможность определения *файловых типов* и *файловых переменных*. После этого информацию, которая может потребоваться впоследствии, можно перенести в файл на диске. Существуют три возможности определения файлового типа:

Type f = file of <базовый тип элементов файла>;

ft = text;
ff = file;

Данный пример демонстрирует наличие трех видов файлов, которыми оперирует Turbo Pascal. Речь идет (по порядку) о *типизированных*, *текстовых* и *нетипизированных* файлах.

file и **of** – зарезервированные слова
text – идентификатор стандартного типа данных (такого, как, например, integer или real);

<базовый тип элементов в файле> – любой тип, кроме файлового.
В разделе описания переменных описываются файловые переменные указанных типов.

```
var
  f1 : file;
  f2 : text;
  f3 : file of integer;
  f4 : file of mass.
```

Среди объявленных файловых переменных f1 – это нетипизированный файл, f2 – текстовый, а f3, f4 – типизированные файлы. Причем если

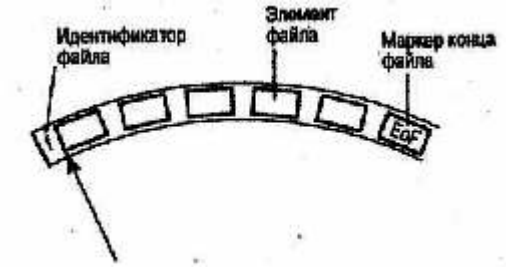
среди типизированных файлов элементы файла f3 относятся к стандартному типу (integer), то элементы файла f4 – к типу, объявленному пользователем (тип mass представляет собой массив целых чисел).

Файлы различных видов имеют нечто общее:

- в файле в каждый момент может быть доступен только один элемент. Например, в файле f3 можно иметь доступ к одному из целых чисел, из которых состоит файл; в файле f4 – к единственному массиву;
- файлы всех видов завершаются маркером конца файла (EoF – End of File).

Доступ к элементам файла обычно осуществляется последовательно, путем их поочередного перебора. Для того чтобы "добраться" до последнего элемента, необходимо обработать (считать или записать) все предыдущие. Кроме того, для типизированных и нетипизированных файлов возможен переход к определенному элементу (минуя предыдущие).

Число элементов файла определяет его *объем*, или *длину*. Как уже отмечалось, длина файла при его объявлении не фиксируется.



Попробуем представить себе все это наглядно. Информация на диске записывается на дорожках, располагающихся по концентрическим окружностям. Поэтому логично представить файл в виде части дорожки – сектора. Здесь изображен файл, содержащий пять элементов. Начинается файл идентификатором, который обозначен буквой **f**. Далее следуют несколько элементов, и завершается файл маркером конца файла (**EoF**). Элементы, образующие файл, представляют собой значения, принадлежащие любому простому или структурированному типу, за исключением файлового. Чтобы подчеркнуть, что длина файла не ограничена, изображенный здесь сектор справа не замкнут.
Стрелка на рисунке представляет собой *указатель текущей позиции* файла (*указатель считывания*). Для чего необходим указатель текущей позиции? Если инициировать запись в файл или считывание

из файла, запись будет проводиться, начиная с элемента, перед которым расположен указатель текущей позиции. Это же касается и считывания.

Разумеется, данное визуальное представление файла очень условно. Часто файл занимает не часть дорожки, а несколько дорожек. Кроме того, никакой стрелки в реальности не существует. Это визуальное представление файла (вернее, его вариации) мы будем использовать в дальнейшем при изучении различных операций над файлами.

Операции над файлами.

В отличие от других типов данных, в Turbo Pascal нет встроенных операторов, предназначенных для манипулирования файлами. Например, не удастся с помощью оператора присваивания присвоить файловой переменной некоторое значение. В силу этого соответствующие операции реализованы в виде процедур и функций.

Файловые процедуры и функции условно можно разделить на пять групп:

1) организующие доступ к файлам;
2) осуществляющие ввод/вывод;
3) предназначенные для ориентирования в файле;
4) специальные операции;
5) завершающие операции.

Подготовка файлов к работе.
Организация доступа к файлам.

Поскольку файлы, в отличие от данных других типов, содержатся в так называемой вторичной памяти (т.е. на дисках), для доступа к файлам недостаточно только объявить файловую переменную. При объявлении файловой переменной не указывается ни имя файла, ни в каком каталоге он содержится. Для того чтобы конкретный файл стал доступным, его необходимо как-то связать (или ассоциировать) с ранее объявленной файловой переменной. Такое связывание осуществляется с помощью процедуры **Assign**, которая является стандартной процедурой Turbo Pascal.
Заголовок процедуры **Assign** выглядит следующим образом:

Assign (<имя файловой переменной>, <имя файла>);
где <имя файловой переменной> – имя файловой переменной любого вида, с которой ассоциируется файл с диска, имеющий имя <имя файла> .
Обобщенный вид имени файла, которое строится по правилам имен

операционной системы MS DOS, выглядит так:
<диск>:\<путь>\<имя_файла>

В качестве параметра процедуры Assign вместо имени файла с диска можно использовать зарезервированное за некоторым устройством имя. В MS DOS основным аппаратным средствам ПК присвоены символические имена, которые можно использовать в операциях ввода/вывода вместо имен файлов.

Символическое имя	Устройство
CON	Консоль
PRN, LPT1, LPT2, LPT3	Принтеры
AUX, COM1, COM2	Коммуникационные каналы
NUL	Фиктивное устройство

CON – символическое имя консоли. Под консолью подразумеваются одновременно и клавиатура, и монитор компьютера. Логическое имя CON можно указать для процедуры **Assign** вместо имени файла; когда информацию требуется ввести с клавиатуры или вывести на экран. Система по выполняемой операции определяет, с клавиатурой или монитором ей следует иметь дело в том или ином случае. Вывести данные можно на экран, но не на клавиатуру, а ввести только с клавиатуры.

PRN – символическое имя принтера. Его можно указать в качестве параметра процедуры Assign (вместо имени файла), когда требуется распечатать информацию на принтере. Если к одному компьютеру подключено несколько принтеров, в MS DOS для принтеров предусмотрены следующие символические имена: LPT1, LPT2, LPT3, (причем имена PRN и LPT1 взаимозаменяемые или синонимы).

До выполнения над файлом каких-либо действий всегда необходимо обратиться к процедуре Assign. После вызова процедуры Assign, связь между указанными файлом и файловой переменной существует вплоть до завершения работы программы, либо пока к этой файловой переменной снова не будет применена процедура Assign.

2.1.2.Открытие файлов.

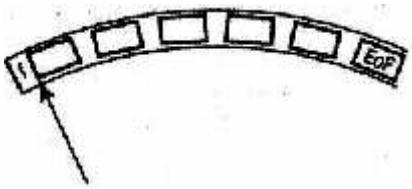
Для открытия файлов предназначены процедуры: **Reset, Rewrite** и **Append**. Причем если процедуры Reset и Rewrite подходят для открытия файла любого вида, то с помощью процедуры Append можно открыть только текстовый файл.

1)

Reset(<имя файловой переменной>);

Процедура **Reset** , примененная к текстовому файлу, открывает его только для чтения, а к типизированному или нетипизированному файлу, открывает его как для чтения, так и для записи.

Файл, открываемый с помощью процедуры Reset, должен уже существовать. Если файла с таким именем и в указанном каталоге не окажется, будет выдано сообщение об ошибке.



После открытия файла с помощью процедуры Reset указатель текущей позиции расположен перед первым элементом.

2)

Rewrite(<имя файловой переменной>);

Процедура **Rewrite,** примененная к текстовому файлу, открывает его только для записи, а к типизированному или нетипизированному файлу, открывает его как для чтения, так и для записи.

Замечание. Если процедуру Reset применить к несуществующему файлу, будет выдано сообщение об ошибке, а процедура Rewrite в этом случае создаст новый файл. Если же процедуру Reset применить к существующему на диске файлу, файл будет открыт, а процедура Rewrite при этом удалит содержимое старого файла и создаст **новый** файл с тем же именем.

Файл, открытый с помощью процедуры Rewrite, выглядит так:

После применения процедуры Rewrite открывается пустой файл, в котором указатель текущей позиции расположен перед маркером конца файла.

2.2. Организация ввода/вывода в файле.

Для осуществления ввода информации в файл и вывода из файла в Turbo Pascal для текстовых и типизированных файлов предназначены процедуры **Read** и **Write**. Использование процедур Read и Write имеет особенности, зависящие от вида файла (текстовый это файл или типизированный).

1)

Read (f,v1[,v2,…,vn]);

f - файловая переменная, имеющая тип **file of …**
v1,v2,…,vn – одна или несколько переменных, принадлежащих любому типу, за исключением файлового.
Процедура **Read** для типизированных файлов обеспечивает считывание одного или нескольких компонентов файла и присвоение считанного значения (значений) некоторой переменной (переменным).

2)

Write (f,v1[,v2,…,vn]);

f - файловая переменная, имеющая тип **file of …**
v1,v2,…,vn – одна или несколько переменных, принадлежащих любому типу, за исключением файлового. Эти переменные должны иметь тот же тип, что и тип файла.
Процедура **Write** для типизированных файлов обеспечивает присвоение значения (значений) некоторой переменной (переменных) компоненту (компонентам) файла.

После записи переменной очередному элементу файла присваивается значение переменной, т.е. старое значение элемента заменяется новым, указатель текущей позиции файла перемещается к следующему элементу. Если указатель находится в конце файла, то при записи очередного элемента этот элемент дополняет файл.

3)

Текстовый файл представляет собой последовательность строк разной длины, состоящих из символов. Каждая строка текстового файла оканчивается маркером конца строки **EoLN,** а завершает текстовый файл, как и любой другой, маркер конца файла **EoF.** К элементам текстового файла возможен только последовательный доступ, начиная с первого. Для того чтобы файл открыть как текстовый, его необходимо ассоциировать с файловой переменной, имеющий тип Text.
Замечание. Помимо Read и Write, для текстовых файлов применимы процедуры Readln и Writeln.

При осуществлении стандартного ввода-вывода операторы выглядят так:
Read(a,b,c); или **Write(x,y,'Вывод на экран');**

В Turbo Pascal существует два стандартных идентификатора, играющие роль текстовых файловых переменных, ассоциируемых вместо файлов с конкретными физическими устройствами компьютера. Это идентификаторы **INPUT** и **OUTPUT**, которые ассоциируются соответственно с клавиатурой и экраном компьютера. Указанные файлы считаются постоянно открытыми (т.е. для них не нужно использовать процедуры Rewrite и Reset) и эти идентификаторы можно использовать в программах. Так, операторы ввода-вывода, о которых шла речь выше, можно представить и иначе:

Read(Input, a,b,c); или **Write(Output, x,y,'Вывод на экран');** Первый оператор считывает из файла INPUT значения (т.е. позволяет ввести их с клавиатуры), которые будут присвоены переменным **a,b,c**. Второй оператор записывает в файл OUTPUT (т.е. выводит на экран) значения переменных **x,y,** а также строку текста. Первая и вторая пары операторов эквивалентны.

Иными словами, в процедурах **Read** и **Write**, когда требуется осуществить ввод с клавиатуры или вывод на экран, указывать файл INPUT или OUTPUT |необязательно. Если в процедуре Read или Write файл не указан, по умолчанию подразумевается файл INPUT или OUTPUT, в зависимости от того, ввод или вывод инициируется.
Упомянутые стандартные файлы можно "переассоциировать". Для этого достаточно воспользоваться процедурой Assign, например:

Assign (Output,'text.txt');

Если после этого в данной программе вызвать процедуру Write без указания файла, соответствующая информация окажется записана в файл TEXT.TXT.

Использование процедуры **Read** для текстовых файлов похоже на ситуацию с типизированными файлами, отличие заключается в том, что переменные **V1, V2, ..., Vn**, записанные в одной команде, могут принадлежать различным типам данных.

Считывание значений типа char.

Если это переменная (переменные) типа char, то из файла считывается символ и присваивается переменной, затем считывается следующий символ и присваивается следующей переменной – и так до тех пор, пока всем переменным типа char, указанным при вызове процедуры Read, будут присвоены считанные из файла значения. (При вводе с клавиатуры между вводимыми значениями требуется вводить некоторый разделитель – пробел, символ табуляции (клавиша <Tab>) или конец строки (клавиша <Enter>). Если очередной

считанный символ окажется маркером конца строки EoLn, считывание будет продолжено из новой строки. Если очередной считанный символ окажется маркером конца файла, то выполнение процедуры будет прекращено.

Считывание значений типа string.

Если это строковая переменная (переменные), то из файла будут считываться все символы до ближайшего маркера конца строки. Если длина считанной строки превзойдет допустимую для значений типа string величину (255 символов), все оставшиеся до конца строки байты отбрасываются. Считывание нескольких строк подряд следует производить неоднократным обращением к процедуре **Readln**.

Считывание числовых значений

Если это значение типа integer или real, процедура **Read** будет пропускать любые пробелы, символы табуляции или маркеры конца строки, предшествующие числовой строке. Когда будет обнаружена первая значащая цифра, процедура Read начнет формировать числовое значение, пока не встретится один из перечисленных символов (пробел, символ табуляции или маркер конца строки). Считанная таким образом последовательность цифр рассматривается как символьное представление соответствующего числа, и полученное значение присваивается соответствующей переменной. Если числовая строка не соответствует ожидаемому формату, будет выдано сообщение об ошибке ввода/вывода. Следующая процедура Read начнет считывание с пробела, символа табуляции или маркера конца строки, которым была завершена предыдущая числовая строка.

4)
Кроме подпрограмм, осуществляющих ввод/вывод, имеется функция, которая осуществляет контроль за вводом/выводом.

IOResult()
возвращает код ошибки, возникшей при выполнении последней операции ввода/вывода. Если ошибки не было, функция возвращает нуль.
Для того чтобы контролировать ошибки ввода/вывода с помощью функции IOResult, должен быть выключен автоконтроль (директива компилятора {$1-}).

Записи в языке Паскаль

Классификация типов данных в Паскале.

Все типы данных языка Паскаль условно можно разделить на *стандартные* и *пользовательские* в соответствии с тем, как они определены. Стандартные типы являются встроенными, зависят от реализации языка и не требуют описания в разделе типов. Пользовательские определяются программистом на основе стандартных типов и предназначаются для решения какой-либо конкретной задачи.

Кроме того, все типы можно разделить на *простые* и *структурированные.*

Переменные простых типов могут в каждый момент времени принимать только одно значение, определенное множеством значений типа. К простым типам относятся *integer(+ все разновидности), real, boolean, char*.

Структурированные типы характеризуются тем, что переменные, относящиеся к ним, принимают сложное, составное значение, представляющее собой набор значений одного или нескольких типов. К сложным типам относятся *string (набор значений типа char), array (набор значений базового типа), file* или *text (набор значений одного или разных типов)* и *record.*

Определение записи. Вариантная часть.

Запись – это сложный, структурированный тип, который представляет собой совокупность данных различного типа, имеющих прямое и характеризующее отношение к какому-либо объекту. Запись есть единица информации о предмете с точки зрения его существенных свойств.

Количество элементов записи строго фиксировано. Каждый элемент записи называется ***полем***. Чаще всего при задании типа запись поля принадлежат разным типам (в противном случае использование записи теряет смысл, т.к. можно вполне обойтись массивом). Названия полей в одной записи должны быть различны, но имена полей внутри различных записей могут совпадать.

В языке Паскаль для обозначения типа записей служит идентификатор *record.*

```
<идентификатор типа> = record
  <поле 1> : <тип значений поля 1>;
  . . .
  <поле k> : <тип значений поля k>;
  [вариантная часть]
  end;
```

Пример.

1. Составить описание книг по следующим характеристикам: код классификации, автор, название, аннотация, год издания, стоимость, количество страниц.

```
type book = record
kod : string[15];
avtor, nazvanie, annot : string;
god : word;
stoim : real;
stranic ; integer;
end;
```

2. Составить описание а) любой даты календаря; б) личных данных человека (имя, фамилия, дата рождения, место рождения); в) личных данных гражданина иностранного государства.

```
type date = record
year : word;
month : 1..12;
day : 1..31;
end;
person_ros = record
imya, fam : string[40];
data_r : date;
mesto_r : string;
end;
person_ino = record
imya, fam : string[40];
data_r : date;
strana : string[30];
data_entry : date;
data_exit : date;
end;
```

Если затем в программе объявить следующие переменные:
var x : date; y: person_ros; z : person_ino;

то в теле программы в разделе операторов будут доступны следующие переменные:

x. year; y. imya; z. imya;
x. month; y. fam; z. fam;
x. day; y. data_r; z. data_r;
y. mesto_r; z. strana;
z. data_entry;
z. data_exit;

Система никогда не перепутает переменные y.data_r и z.data_r, т.к. переменные y и z принадлежат к разным типам.
Кроме того, в программе будут известны переменные: y.data_r.year; y.data_r.month и т.д.

Идентификаторы полей внутри записи могут быть использованы в качестве имен других переменных, т.е. в программе могут быть

объявлены переменные data_r, fam, day и т.п., которые будут абсолютно самостоятельными.

3. Во втором примере можно существенно упростить описание второго и третьего типов, если использовать записи с вариантной частью:

```
person = record
  imya, fam : string[40];   описание фиксированной части (всегда
располагается вначале)
  data_r : date;
  case citizen : boolean of   | |вариантная часть в каждой записи |может быть
только одна. Обычно |задается оператором варианта case. |
     true : mesto_r : string;
     false : strana : string[30];
     data_entry : date;
     data_exit : date;
  end;   однозначно действует на окончание описания типа и на окончание
case.
```

Массив записей. Оператор присоединения.

Запись, как единица информации, описывает один объект. Решение конкретных задач, как правило, требует объединения информации в более крупные множества. В таких случаях используются более сложные структуры типа массив записей или файл записей.

Массив записей представляет собой массив, базовым типом элементов которого является запись.

Пример.

Описание студентов одной группы может выглядеть следующим образом:

```
type gruppa = array[1..30] of person_ros;
var student: gruppa; k : integer;
. . .
k:=3;
for i:=a to b do
  begin
  gotoxy(1,k); readln(sdudent[i].imya);
  gotoxy(15,k); readln(sdudent[i].fam);
  gotoxy(41,k); readln(sdudent[i].mesto_r);
  k:=k+3;
  end;
```

Этот фрагмент программы может быть записан иначе, если использовать оператор присоединения:

```
k:=3;
for i:=a to b do
```

```
  with student[i] do
  begin
  gotoxy(1,k); readln(imya);
  gotoxy(15,k); readln(fam);
  gotoxy(41,k); readln(mesto_r);
  k:=k+3;
  end;
```

Если один оператор или группа операторов, процедур, функций и т.д. обрабатывает одну переменную типа запись, то можно не указывать каждый раз имя этой переменной, а указывать только имена полей, если эта группа операторов обрамлена оператором ***сокращенной записи*** (или ***оператором присоединения***).

```
  with <список идентификаторов> do
  <оператор>;
```

Список идентификаторов может содержать произвольное число переменных комбинированного типа запись. Если имена полей этих переменных уникальны, то никаких проблем не возникает. В противном случае действует правило:

запись with <имя 1>, <имя 2>, …, <имя n> do <оператор>;

эквивалентна конструкции

```
  with <имя 1> do
  with <имя 2> do
  . . .
  with <имя n> do
  <оператор>;
```

Лекции 8. Подпрограммы. Процедуры и функции.

Описание процедуры. Оператор процедуры.

Процедуры и функции представляют собой важный инструмент Турбо Паскаля, позволяющий писать хорошо структурированные программы. В структурированных программах обычно легко прослеживается основной алгоритм, их нетрудно понять любому читателю, они проще в отладке и менее чувствительны к ошибкам программирования. Все эти свойства являются следствием важной особенности процедур (функций), каждая из которых представляет собой во многом самостоятельный фрагмент программы, связанный с основной программой лишь с помощью нескольких параметров. Самостоятельность процедур (функций) позволяет локализовать в них все детали программной реализации того или иного алгоритмического действия и поэтому изменение этих деталей, например, в процессе отладки обычно не приводит к изменениям основной программы.

Структура процедуры:
Заголовок процедуры имеет вид:
PROCEDURE <имя> [(<сп. ф. п . >)] ;
Имя – правильный идентификатор, сп. Ф. п - список формальных параметров
Примеры описания процедур:

```
Procedure sum;
For i:=1 to n do
S:=s+1;
Writeln(s);
End;

Procedure sum;
Var I,s:integer;
For i:=1 to n do
S:=s+1;
Writeln(s);
End;

Procedure sum(I,s:integer);
For i:=1 to n do
S:=s+1;
Writeln(s);
End;

Procedure sum( var I,s:integer);
For i:=1 to n do
S:=s+1;
Writeln(s);
End;
```

Пример использования процедуры:
Задача Запросить с клавиатуры число от 1 до 30, вывести на экран сумму всех чисел от 1 до введенного числа.(использовать процедуру).
Код программы:

```
program Proc;
uses CRT;
var a:integer;
procedure Sum(var n:integer);
var i,s:integer;
begin
s:=0;
for i:=1 to n do
s:=s+i;
writeln('rezultat: ',s);
end;

begin
clrscr;
writeln('Vvedite chislo ot 1 do 30');
readln(a);
sum(a);
readln;
end.
```

Описание Функции. Оператор функции.

Функции и функции представляют собой важный инструмент Турбо Паскаля, позволяющий писать хорошо структурированные программы. В структурированных программах обычно легко прослеживается основной алгоритм, их нетрудно понять любому читателю, они проще в отладке и менее чувствительны к ошибкам программирования. Все эти свойства являются следствием важной особенности процедур (функций), каждая из которых представляет собой во многом самостоятельный фрагмент программы, связанный с основной программой лишь с помощью нескольких параметров. Самостоятельность процедур (функций) позволяет локализовать в них все детали программной реализации того или иного алгоритмического действия и поэтому изменение этих деталей, например, в процессе отладки обычно не приводит к изменениям основной программы.

FUNCTION <имя> [(<сп.ф.п.>)] : <тил>;
Здесь <имя> - имя подпрограммы (правильный идентификатор);
<сп.ф.п.> - список формальных параметров; <тип> - тип возвращаемого функцией результата.
Примеры описания функций:

```
function sum:integer;
begin
For i:=1 to n do
S:=s+1;
Sum:=s;
End;
function sum:integer;
Var I,s:integer;
begin
For i:=1 to n do
S:=s+1;
Sum:=s
End;

function sum(I,s:integer):integer;
begin
For i:=1 to n do
S:=s+1;
Sum:=s;
End;

function sum( var I,s:integer):integer;
begin
For i:=1 to n do
S:=s+1;
Sum:=s;
End;
```

Пример использования функции:
Задача: Запросить с клавиатуры число x, вывести на экран результат, рассчитанный по формуле: $Y:=\sin^2(x)+\cos^2(x)$;
Код программы:

```
Program phunk;
uses CRT;
var x:real;

function si2(var n:real):real;
var y:real;
begin
y:=sqr(sin(n));
si2:=y;
end;
```

```
function co2(var n:real):real;
var y:real;
begin
y:=sqr(cos(n));
co2:=y;
end;

begin
clrscr;
writeln('Vvedite znachenie X');
readln(x);
writeln('rezultat ',si2(x)+co2(x));
readln;
end.
```

Лекции 9. Структурное программирование.

На всех этапах подготовки к алгоритмизации задачи широко используется структурное представление алгоритма.

Структурное программирование воплощает принципы системного подхода в процессе создания и эксплуатации программного обеспечения ЭВМ. В основу структурного программирования положены следующие достаточно простые положения:

1. алгоритм и программа должны составляться поэтапно (по шагам).
2. сложная задача должна разбиваться на достаточно простые части, каждая из которых имеет один вход и один выход.
3. логика алгоритма и программы должна опираться на минимальное число достаточно простых базовых управляющих структур.

Структурное программирование иногда называют еще "программированием без GO TO". Рекомендуется избегать употребления оператора перехода всюду, где это возможно, но чтобы это не приводило к слишком громоздким структурированным программам.

К полезным случаям использования оператора перехода можно отнести выход из цикла или процедуры по особому условию, "досрочно" прекращающего работу данного цикла или данной процедуры, т.е. завершающего работу некоторой структурной единицы (обобщенного оператора) и тем самым лишь локально нарушающего структурированность программы.

Фундаментом структурного программирования является *теорема о структурировании*. Эта теорема устанавливает, что, как бы сложна ни была задача, схема соответствующей программы всегда может быть представлена с

использованием ограниченного числа элементарных управляющих структур. Базовыми элементарными структурами являются структуры: следование, ветвление и повторение (цикл), любой алгоритм может быть реализован в виде композиции этих трех конструкций.

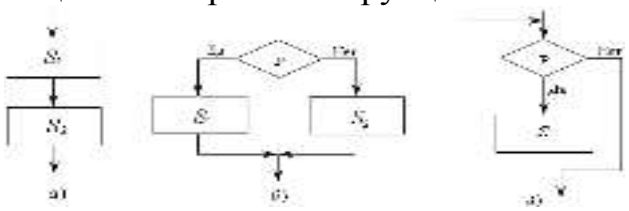


Рис. 5

Первая (*а*) структура - тип последовательность (или просто последовательность), вторая (*б*) – структура выбора (ветвление), третья (*в*) – структура цикла с предусловием.

При словесной записи алгоритма указанные структуры имеют соответственно следующий смысл:

«выполнить S_1; выполнить S_2»,

если P, то выполнить S_1, иначе выполнить S_2»,

«до тех пор, пока P, выполнять S»,

где P - условие; S, S_1, S_2 - действия.

Применительно к языку Паскаль, в котором наиболее полно нашли свое отражение идеи структурного программирования, целесообразно при проектировании алгоритмов дополнительно использовать еще четыре элементарные структуры: сокращенную запись разветвления (рис. 16, *а*); структуру варианта (рис. 16, *б*); структуру повторения или цикла с параметром (рис. 16, *в*); структуру цикла с постусловием (рис. 6, *г*). Каждая из этих структур имеет один вход и один выход.

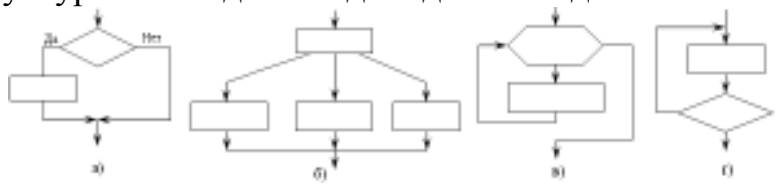


Рис. 6

Ветвящимся (разветвляющимся) называется вычислительный процесс, в котором происходит выбор одного из возможных вариантов вычислений в зависимости от проверки заданных условий.

В зависимости от типа и числа проверяемых условий различают:

- ветвление с простым условием (условие - выражение отношения);
- ветвление с составным условием (условие - логическое выражение);
- сложное ветвление (несколько условий).

Вариант вычислений, определяемый в результате проверки условия, называется *ветвью*.

Циклическим называется процесс многократного повторения некоторого участка вычислений при изменении хотя бы одной из входящих в него величин.

Повторяющийся участок вычисления называется *циклом*. Операции, осуществляемые в цикле, составляют *тело цикла.*

Величина, изменяющая своё значение от цикла к циклу, называется *параметром цикла.*

Зависимость, связывающая текущее и предыдущее значения параметра цикла, определяет *закон изменения параметра цикла.* Зависимость, предписывающая повторение цикла, либо выход из него, называется *условием повторения цикла.*

Полный однократный проход цикла от начала до конца называется *итерацией.*

Все циклические процессы по признаку определения количества повторений (M) разделяются на два класса.

Арифметическим называется циклический процесс, число повторений в котором может быть определено заранее, т.е. не зависит от результатов счёта в теле цикла.

Итерационным является циклический процесс, число повторений в котором зависит от результатов вычислений в теле цикла и не может быть определено заранее.

Независимо от того, к какому классу относится вычислительный процесс, каждый из них содержит обязательные элементы:

- вход в цикл (формирование начального значения параметра цикла);
- вычисления в теле цикла (расчёт текущего значения функций, формирования нового значения параметра цикла, а также вспомогательные операции);
- выход из цикла (проверка условия, определяющего повторение вычислений, либо их прекращение).

По своему содержанию эти элементы зависят от класса и особенностей цикла, в котором используются.

В соответствии с видом задания (изменения) параметра цикла арифметические циклы подразделяются на:

- циклы с аналитическим изменением параметра;
- циклы с табличным заданием параметра.

Выполнение арифметических циклов, т.е. многократное вычисление значений функции при изменяющихся значениях аргумента, называется *табуляцией функции*.

Распространены **две методики (стратегии) разработки программ**, относящиеся к структурному программированию:

– программирование «сверху вниз»;

– программирование «снизу вверх».

Программирование «сверху вниз», или нисходящее программирование – это методика разработки программ, при которой разработка начинается с определения целей решения проблемы, после чего идет последовательная детализация, заканчивающаяся детальной программой.

Такой подход удобен тем, что позволяет человеку постоянно мыслить на предметном уровне, не опускаясь до конкретных операторов и переменных. Кроме того, появляется возможность некоторые подпрограммы не реализовывать сразу, а временно откладывать, пока не будут закончены другие части.

Программирование «снизу вверх», или восходящее программирование – это методика разработки программ, начинающаяся с разработки подпрограмм (процедур, функций), в то время когда проработка общей схемы не закончилась.

Такая методика является менее предпочтительной по сравнению с нисходящим программированием, так как часто приводит к нежелательным результатам, переделкам и увеличению времени разработки.

Подпрограммы бывают двух видов – процедуры и функции. Процедура просто выполняет группу операторов, а функция вдобавок вычисляет некоторое значение и передает его обратно в главную программу. Это значение имеет определенный тип. Данные передаются подпрограмме в виде параметров или аргументов, которые обычно описываются в ее заголовке так же, как переменные. Подпрограммы вызываются, как правило, путем простой записи их названия с нужными параметрами.

Подпрограммы могут быть вложенными – допускается вызов подпрограммы не только из главной программ, но и из любых других программ.

В некоторых языках программирования допускается вызов подпрограммы из себя самой. Такой прием называется рекурсией и опасен тем, что может привести к зацикливанию – бесконечному самовызову.

Достоинства структурного программирования:

– повышается надежность программ (благодаря хорошему структурированию при проектировании, программа легко поддается тестированию и не создает проблем при отладке);

– повышается эффективность программ (структурирование программы позволяет легко находить и корректировать ошибки, а отдельные подпрограммы можно переделывать (модифицировать) независимо от других);

– уменьшается время и стоимость программной разработки;

– улучшается читабельность программ.

Предпрограммная подготовка задачи.

На ЭВМ могут решаться задачи различного характера, например: научно-инженерные; разработки системного программного обеспечения; обучения; управления производственными процессами и т. д. В процессе подготовки и решения на ЭВМ научно-инженерных задач можно выделить следующие этапы:

1. постановка задачи;
2. формирование математической модели задачи;
3. выбор и обоснование метода решения;
4. алгоритмизация вычислительного процесса;
5. программирование;
6. отладка и тестирование программы;
7. решение задачи на ЭВМ и анализ результатов;

8. сопровождение программы.

В задачах другого класса некоторые этапы могут отсутствовать, например, в задачах разработки системного программного обеспечения отсутствует математическое описание. Перечисленные этапы связаны друг с другом. Для уменьшения числа подобных изменений необходимо на каждом этапе по возможности учитывать требования, предъявляемые последующими этапами

Постановка задачи - этап словесной формулировки, определяющий цель решения, исходные данные, основные закономерности, условия и ограничения применения этих закономерностей. Анализируются характер и сущность всех величин, используемых в задаче, и определяются условия, при которых она решается.

Постановка задачи должна отвечать следующим требованиям:

- четкая формулировка цели с указанием вида и характеристик конечных результатов;
- представление значений и размерностей исходных данных;
- определение всех возможных вариантов решения, условий выбора каждого;
- обозначения границы применимости и действия в случае выхода за них.

Формирование математической модели задачи - этап перевода словесной постановки задачи в совокупность математических зависимостей, описывающих исходные данные и вычисления промежуточных и конечных результатов.

Математическая модель формируется с определенной точностью, допущениями и ограничениями. При этом в зависимости от специфики решаемой задачи могут быть использованы различные разделы математики и других дисциплин.

Математическая модель должна удовлетворять по крайней мере двум требованиям: реалистичности и реализуемости. Под реалистичностью понимается правильное отражение моделью наиболее существенных черт исследуемого явления. Реализуемость достигается разумной абстракцией, отвлечением от второстепенных деталей, чтобы свести задачу к проблеме с известным решением. Условием реализуемости является возможность практического выполнения необходимых вычислений за отведенное время при доступных затратах требуемых ресурсов.

Полученная математическая модель должна отвечать следующим требованиям:

- вначале составляется модель исходных данных, затем - расчетные зависимости;
- в модели исходных данных не изменяются размерности данных и не используются никакие математические операции;
- обозначение всех входящих в зависимости величин именами, определяющими их суть;
- указание размерностей всех используемых величин для контроля и дальнейшей модернизации решения;

Выбор и обоснование метода решения - этап разработки или выбора из уже имеющихся метода решения, в том числе выбор стандартных структур вычислительных процессов (линейной, ветвящейся, циклической). Критерии выбора определяются математической моделью решения (предыдущий этап), требованиями к универсальности метода и точности результата, ограничениями технического и программного обеспечении. При обосновании выбора метода необходимо учитывать различные факторы и условия, точность вычислений, время решения задачи на ЭВМ, требуемый объем памяти и другие. Здесь следует указать альтернативные методы и аргументы сделанного выбора.

Алгоритмизация вычислительного процесса - этап разработки совокупности предписаний, однозначно определяющих последовательность преобразования исходных данных в конечные результаты. На данном этапе составляется алгоритм решения задачи согласно действиям, задаваемым выбранным методом решения. Процесс обработки данных разбивается на отдельные относительно самостоятельные блоки, и устанавливается последовательность выполнения блоков. Разрабатывается блок-схема алгоритма.

Программирование. При составлении программы алгоритм решения задачи переводится на конкретный язык программирования. Для программирования обычно используются языки высокого уровня, поэтому составленная программа требует перевода ее на машинный язык ЭВМ. После такого перевода выполняется уже соответствующая машинная программа.

1. Программа должна быть универсальной, то есть не зависящей от конкретного набора данных. Например, если количество обрабатываемых данных может меняться, то следует предусмотреть хранение максимально возможного их количества. Универсальная программа должна уметь обрабатывать ошибки, которые могут возникнуть в процессе обработки информации.
2. Вместо констант[1] лучше использовать переменные[2]. Если в программе используются константы, то при их изменении нужно изменять в исходной программе каждый оператор[3], содержащий прежнюю константу. Эта процедура отнимает много времени и часто вызывает ошибки. В программе следует предусмотреть контроль вводимых данных (в частности, программа не должна выполняться, если данные выходят за пределы допустимого диапазона).

---

[1] Конста́нта в программировании — способ адресования данных, изменение которых рассматриваемой программой не предполагается или запрещается.

[2] Перемéнная в императивном программировании — поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным. Данные, находящиеся в переменной (то есть по данному адресу памяти), называются значением этой переменной.
В других парадигмах программирования, например, в функциональной и логической, понятие переменной оказывается несколько иным. В таких языках переменная определяется как имя, с которым может быть связано значение, или даже как место (location) для хранения значения.

[3] Инструкция или оператор (англ. statement) — наименьшая автономная часть языка программирования; команда. Программа обычно представляет собой последовательность инструкций.
Многие языки (например, Си) различают инструкцию и определение. Различие в том, что инструкция исполняет код, а определение создаёт идентификатор (то есть можно рассматривать определение как инструкцию присваивания).
Ниже приведены основные общие инструкции императивных языков программирования.

3. Некоторые простые приемы позволяют повысить эффективность программы (то есть уменьшить количество выполняемых операций и время работы программы). К таким приемам относится:

- использование операции умножения вместо возведения в степень;
- если некоторое арифметическое выражение встречается в вычислениях несколько раз, то его следует вычислить заранее и хранить в памяти ЭВМ, а по мере необходимости использовать;
- при организации циклов в качестве границ индексов[4] использовать переменные, а не выражения, которые вычислялись бы при каждом прохождении цикла;
- особое внимание обратить на организацию циклов, убрав из них все повторяющиеся с одинаковыми данными вычисления и выполняя их до входа в цикл.

4. Программа должна содержать комментарии, позволяющие легко проследить за логической взаимосвязью и функциями отдельных ее частей.

При написании программы следует структурировать ее текст так, чтобы она хорошо читалась. В частности, в программе должно быть хорошо видно, где начинается и где заканчивается цикл.

Отладка программы – процесс выявления и исправления синтаксических и логических ошибок в программе. Суть отладки заключается в том, что выбирается некоторый набор исходных данных, называемый тестовым набором (тестом), и задача с этим набором решается дважды: один раз – исполнением программы, второй раз – каким-либо иным способом, исходя из условия задачи, так сказать, «вручную». При совпадении результатов алгоритм считается верным. В качестве тестового набора можно выбрать любые данные, которые позволяют:

- обеспечить проверку выполнения всех операций алгоритма;
- свести количество вычислений к минимуму.

В ходе синтаксического контроля программы транслятором выявляются конструкции и сочетания символов, недопустимые с точки зрения правил их построения или написания, принятых в данном языке. Сообщения об ошибках

---

[4] Индекс в языках программирования — элемент перечислимого множества, который указывает на конкретный элемент массива, обычно является неотрицательным скалярным целым числом.
Есть три способа, как элементы массива могут быть проиндексированы целыми неотрицательными числами[1]:
0 («индекс с началом с нуля»)
первый элемент массива имеет индекс 0;
1 («индекс с началом с единицы»)
первый элемент массива имеет индекс 1;
n («индекс началом с n»)
базисный индекс массива может быть свободно выбран. Обычно языки программирования, позволяющие «индекс началом с n», разрешают также в качестве индекса массива выбирать отрицательные значения, а также и другие скалярные типы данных, как перечисления или символы.
Массив может иметь несколько измерений, таким образом обычная практика обращаться к массиву, используя несколько индексов. Например к двумерному массиву с тремя строками и четырьмя столбцами можно было бы обратиться к элементу в 2-ом ряду и 4-ой столбце с помощью выражения: [1,3] (в языке в котором приоритет у строки) и [3,1] (в языке в котором приоритет у столбца) в случае индексом с началом с нуля. Таким образом два индекса используются для двумерных массивов, три — для трехмерного массива, и n — для n-мерного массива.

ЭВМ выдает программисту, при этом вид и форма выдачи подобных сообщений зависят от вида языка и версии используемого транслятора.

После устранения синтаксических ошибок проверяется логика работы программы в процессе ее выполнения с конкретными исходными данными. Для этого используются специальные методы, например, в программе выбираются контрольные точки, для которых вручную рассчитываются промежуточные результаты. Эти результаты сверяются со значениями, получаемыми ЭВМ в данных точках при выполнении отлаживаемой программы. Кроме того, для поиска ошибок могут быть использованы отладчики, выполняющие специальные действия на этапе отладки, например, удаление, замена или вставка отдельных операторов или целых фрагментов программы, вывод или изменение значений заданных переменных.

Тестирование - это испытание, проверка правильности работы программы в целом, либо её составных частей.

Отладка и тестирование (англ. test - испытание) - это два четко различимых и непохожих друг на друга этапа:

- при отладке происходит локализация и устранение синтаксических ошибок и явных ошибок кодирования;
- в процессе же тестирования проверяется работоспособность программы, не содержащей явных ошибок.

Тестирование устанавливает факт наличия ошибок, а отладка выясняет ее причину. В современных программных системах (Turbo Basic, Turbo Pascal, Turbo C и др.) отладка осуществляется часто с использованием специальных программных средств, называемых отладчиками. Эти средства позволяют исследовать внутреннее поведение программы.

Программа-отладчик обычно обеспечивает следующие возможности:

- пошаговое исполнение программы с остановкой после каждой команды (оператора);
- просмотр текущего значения любой переменной или нахождение значения любого выражения, в том числе, с использованием стандартных функций; при необходимости можно установить новое значение переменной;
- установку в программе "контрольных точек", т.е. точек, в которых программа временно прекращает свое выполнение, так что можно оценить промежуточные результаты, и др.

При отладке программ важно помнить следующее:

- в начале процесса отладки надо использовать простые тестовые данные;
- возникающие затруднения следует четко разделять и устранять строго поочередно;
- не нужно считать причиной ошибок машину, так как современные машины и трансляторы обладают чрезвычайно высокой надежностью.

Как бы ни была тщательно отлажена программа, решающим этапом, устанавливающим ее пригодность для работы, является контроль программы по результатам ее выполнения на системе тестов.

Программу условно можно считать правильной, если её запуск для выбранной системы тестовых исходных данных во всех случаях дает правильные результаты.

Для реализации метода тестов должны быть изготовлены или заранее известны эталонные результаты. Вычислять эталонные результаты нужно обязательно до, а не после получения машинных результатов. В противном случае имеется опасность невольной подгонки вычисляемых значений под желаемые, полученные ранее на машине.

Тестовые данные должны обеспечить проверку всех возможных условий возникновения ошибок:

- должна быть испытана каждая ветвь алгоритма;
- очередной тестовый прогон должен контролировать то, что еще не было проверено на предыдущих прогонах;
- первый тест должен быть максимально прост, чтобы проверить, работает ли программа вообще;
- арифметические операции в тестах должны предельно упрощаться для уменьшения объема вычислений;
- количества элементов последовательностей, точность для итерационных вычислений, количество проходов цикла в тестовых примерах должны задаваться из соображений сокращения объема вычислений;
- минимизация вычислений не должна снижать надежности контроля;
- тестирование должно быть целенаправленным и систематизированным, так как случайный выбор исходных данных привел бы к трудностям в определении ручным способом ожидаемых результатов; кроме того, при случайном выборе тестовых данных могут оказаться непроверенными многие ситуации;
- усложнение тестовых данных должно происходить постепенно.

Процесс тестирования можно разделить на три этапа.

1. Проверка в нормальных условиях. Предполагает тестирование на основе данных, которые характерны для реальных условий функционирования программы.

2. Проверка в экстремальных условиях. Тестовые данные включают граничные значения области изменения входных переменных, которые должны восприниматься программой как правильные данные. Типичными примерами таких значений являются очень маленькие или очень большие числа и отсутствие данных. Еще один тип экстремальных условий - это граничные объемы данных, когда массивы состоят из слишком малого или слишком большого числа элементов.

3. Проверка в исключительных ситуациях. Проводится с использованием данных, значения которых лежат за пределами допустимой области изменений. Известно, что все программы разрабатываются в расчете на обработку какого-то ограниченного набора данных. Поэтому важно получить ответ на следующие вопросы:

Что произойдет, если программе, не рассчитанной на обработку отрицательных и нулевых значений переменных, в результате какой-либо ошибки придется иметь дело как раз с такими данными?

Как будет вести себя программа, работающая с массивами, если количество их элементов превысит величину, указанную в объявлении массива?

Что произойдет, если числа будут слишком малыми или слишком большими?

Наихудшая ситуация складывается тогда, когда программа воспринимает неверные данные как правильные и выдает неверный, но правдоподобный результат.

Программа должна сама отвергать любые данные, которые она не в состоянии обрабатывать правильно.

Решение задачи на ЭВМ и анализ результатов. После отладки программы ее можно использовать для решения прикладной задачи. При этом обычно выполняется многократное решение задачи на ЭВМ для различных наборов исходных данных. Получаемые результаты интерпретируются и анализируются специалистом или пользователем, поставившим задачу.

Сопровождение программы:

Разработанная программа длительного использования устанавливается на ЭВМ, как правило, в виде готовой к выполнению машинной программы. К программе прилагается документация, включая инструкцию для пользователя. Так как при установке программы на диск для ее последующего использования помимо файлов с исполняемым кодом устанавливаются различные вспомогательные программы (утилиты, справочники, настройщики и т. д.), а также необходимые для работы программ разного рода файлы с текстовой, графической, звуковой и другой информацией.

А также:

- доработка программы для решения конкретных задач;
- составление документации к решенной задаче, математической модели, алгоритму, программе по их использованию.

Лекции 10. Модули и модульное программирование.

Назначение модулей

Стандартный Паскаль не предусматривает механизмов раздельной компиляции частей программы с последующей их сборкой перед выполнением. Вполне понятно стремление разработчиков коммерческих компиляторов Паскаля включать в язык средства, повышающие его модульность.

Модуль – **это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы инициирующей части.**

Основным принципом модульного программирования является принцип

«разделяй и властвуй». Модульное программирование – это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определенным правилам.

Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

Термин «модуль» в программировании начал использоваться в связи с внедрением модульных принципов при создании программ. В 70-х годах под модулем понимали какую-либо процедуру или функцию, написанную в соответствии с определенными правилами. Например: «*Модуль должен быть простым, замкнутым (независимым), обозримым (от 50 до 100 строк), реализующим только одну функцию задачи, имеющим одну входную и одну выходную точку*».

Первым основные свойства программного модуля более-менее четко сформулировал Парнас (Parnas): «*Для написания одного модуля должно быть достаточно* ***минимальных*** *знаний о тексте другого*». Таким образом, в соответствии с определением, модулем могла быть любая отдельная процедура (функция) как самого нижнего уровня иерархии (уровня реализации), так и самого верхнего уровня, на котором происходят только вызовы других процедур-модулей.

Таким образом, Парнас первым выдвинул концепцию скрытия информации (information hiding) в программировании. Однако существовавшие в языках 70-х годов только такие синтаксические конструкции, как процедура и функция, не могли обеспечить надежного скрытия информации, поскольку подвержены влиянию глобальных переменных, поведение которых в сложных программах бывает трудно предсказуемым.

Решить эту проблему можно было только разработав новую синтаксическую конструкцию, которая не подвержена влиянию глобальных переменных.

Такая конструкция была создана и названа модулем. Изначально предполагалось, что при реализации сложных программных комплексов модуль должен использоваться наравне с процедурами и функциями как конструкция, объединяющая и надежно скрывающая детали реализации определенной подзадачи.

Таким образом, количество модулей в комплексе должно определяться декомпозицией поставленной задачи на независимые подзадачи. В предельном случае модуль может использоваться даже для заключения в него всего лишь одной процедуры, если необходимо, чтобы выполняемое ею локальное действие было гарантировано независимым от влияния других частей программы при любых изменениях.

Впервые специализированная синтаксическая конструкция модуля была предложена Н. Виртом в 1975 г. и включена в его новый язык Modula. Насколько сильно изменяются свойства языка, при введении механизма модулей, свидетельствует следующее замечание Н.Вирта, сделанное им по поводу более

позднего языка Модула-2: «Модули – самая важная черта, отличающая язык Модула-2 от его предшественника Паскаля».

По своей организации и характеру использования в программе модули Турбо Паскаля близки к модулям-пакетам (PACKAGE) языка программирования Ада. В них так же, как и в пакетах Ады, явным образом выделяется некоторая «видимая» интерфейсная часть, в которой сконцентрированы описания глобальных типов, констант, переменных, а также приводятся заголовки процедур и функций. Появление объектов в интерфейсной части делает их доступными для других модулей и основной программы. Тела процедур и функций располагаются в исполняемой части модуля, которая может быть скрыта от пользователя.

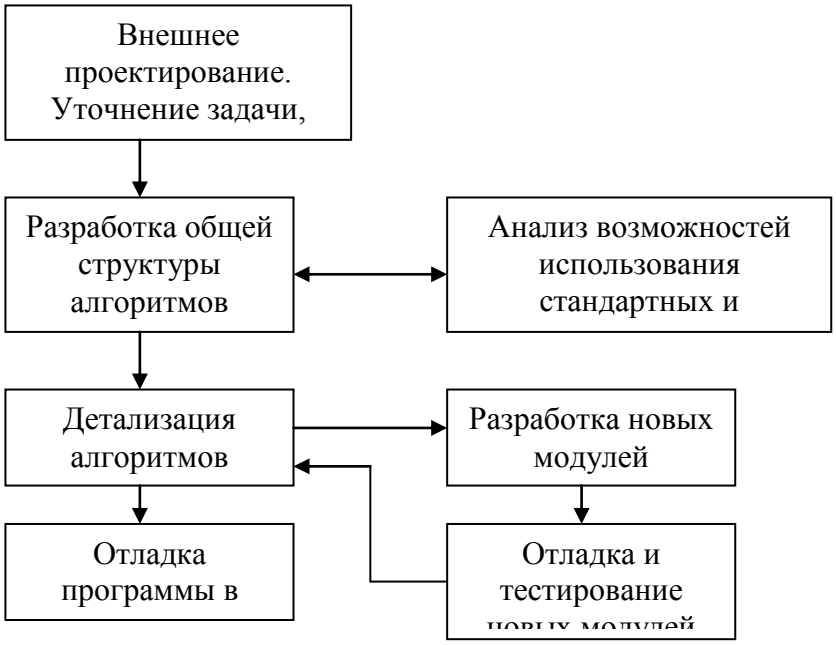


Рис.2. Последовательность разработки программного проекта

Значение модулей для технологии разработки программного проекта может быть продемонстрировано диаграммой на рис. 2.

Модули представляют собой прекрасный инструмент для разработки библиотек прикладных программ и мощное средство модульного программирования. Важная особенность модулей заключается в том, что компилятор размещает их программный код в отдельном сегменте памяти. Длина сегмента не может превышать 64 Кбайт, однако количество одновременно используемых модулей ограничивается лишь доступной памятью, что позволяет создавать большие программы.

Структура модулей

Всякий модуль имеет следующую структуру:

Unit <имя_модуля>;
interface
<интерфейсная часть>;

implementation
<исполняемая часть>;
begin
<инициирующая часть>;
end.

Здесь UNIT – зарезервированное слово (единица); начинает заголовок модуля;

<имя_модуля> - имя модуля (правильный идентификатор);

INTERFACE – зарезервированное слово (интерфейс); начинает интерфейсную часть модуля;

IMPLEMENTATION – зарезервированное слово (выполнение); начинает исполняемую часть модуля;

BEGIN – зарезервированное слово; начинает инициирующую часть модуля; причем конструкция begin <инициирующая часть> необязательна;

END – зарезервированное слово – признак конца модуля.

Таким образом, модуль состоит из заголовка и трех составных частей, любая из которых может быть пустой.

Ниже показана общая структура модуля, дополненная комментариями, поясняющими смысл и назначение каждого раздела модуля.

```
Unit ИдентификаторМодуля;
                                {Интерфейсный раздел}
interface
    {В этом разделе описывается взаимодействие данного модуля}
    {с другими пользовательскими и стандартными модулями, а также}
    {с главной программой. Другими словами – взаимодействие }
    {модуля  с «внешним миром».                              }

                                {Список импорта интерфейсного раздела}
uses
    {В этом списке через запятые перечисляются идентификаторы}
  {модулей, информация интерфейсных частей которых должна   }
    {быть доступна в данном модуле. Здесь целесообразно описывать}
{идентификаторы только тех модулей, информация из которых}
{используется в описаниях раздела interface данного модуля.}

                                {Список экспорта интерфейсного раздела}
const {Список экспорта состоит из подразделов описания констант,}
type      {типов, переменных, заголовков процедур и функций, которые}
var       {определены в данном модуле, но использовать которые разре-}
procedure {шено во всех других модулях и программах, включающих имя}
function  {данного модуля в своей строке uses. Для процедур и функций}
          {здесь описываются только заголовки, но с обязательным}
```

{полным описанием формальных параметров.}

{Раздел реализации}

implementation

{В этом разделе указывается реализационная (личная) часть}
{описаний данного модуля, которая недоступна для других }
{модулей и программ. Другими словами – «внутренняя кухня»}
{модуля.}

{Список импорта раздела реализации}

uses

{В этом списке через запятые перечисляются идентификаторы}
{тех модулей, информация интерфейсных частей которых должна}
{быть доступна в данном модуле. Здесь целесообразно описывать}
{идентификаторы всех необходимых модулей, информация из}
{которых не используется в описаниях раздела interface данного}
{модуля и об использовании которых не должен знать ни один}
{другой модуль. }

{Подразделы внутренних для модуля описаний}

label {В этих подразделах описываются метки, константы, типы,}
const {переменные, процедуры и функции, которые описывают}
type {алгоритмические действия, выполняемые данным модулем, и}
var {которые являются «личной собственностью» исключительно}
procedure {только данного модуля. Эти описания недоступны ни одному}
function {другому модулю. Заголовки процедур и функций в этом}
{подразделе допускается указывать без списка формальных}
{параметров. Если заголовки указаны все же с параметрами, то}
{их список должен быть идентичен такому же списку для}
{соответствующей процедуры (функции) в разделе interface}

{Раздел инициализации}

begin

{В этом разделе указываются операторы начальных установок,}
{необходимых для запуска корректной работы модуля. Операторы}
{разделов инициализации модулей, используемых в программе,}
{выполняются при начальном запуске программы в том же }
{порядке, в каком идентификаторы модулей описаны в }
{предложении uses. Если операторы инициализации не требуются, }
{то слово begin может быть опущено.}

end.

Заголовок модуля и связь модулей друг с другом

Заголовок модуля состоит из зарезервированного слова unit и следующего за ним имени модуля. Для правильной работы среды Турбо Паскаля и возможности подключения средств, облегчающих разработку больших программ, *имя модуля должно совпадать с именем дискового файла*, в который помещается исходный текст модуля. Если, например, имеем заголовок модуля

Unit primer;

то исходный текст этого модуля должен размещаться на диске в файле primer.pas.

Имя модуля служит для его связи с другими модулями и основной программой. Эта связь устанавливается специальным предложением:

uses <список модулей>

Здесь **USES** – зарезервированное слово (использует);

<список модулей> - список модулей, с которыми устанавливается связь; элементы списка – имена модулей через запятую.

Если в программе модули используются, то *предложение* ***uses <список модулей>*** *должно стоять сразу после заголовка программы*, т.е. должно открывать раздел описаний основной программы. В модулях могут использоваться другие модули. В модулях предложение uses <список модулей> может стоять сразу после слова interface или сразу после слова implementation. Допускается и два предложения uses, т.е. оно может стоять и там, и там.

Интерфейсная часть

Интерфейсная часть открывается зарезервированным словом **INTERFACE**. В этой части содержатся объявления всех глобальных объектов модуля (типов, констант, переменных и подпрограмм), которые должны быть доступны основной программе и (или) другим модулям. При объявлении глобальных подпрограмм в интерфейсной части указывается только их заголовок, например:

```
Unit complexn;
Interface
Type
Complex= record
    Re, im: real;
End;
Procedure AddC(x,y: complex, var z: complex);
Procedure MulC (x,y: complex, var z: complex);
```

Если теперь в основной программе написать предложение

Uses complexn;

то в программе станут доступными тип cmplex и две процедуры – AddC и MulC из модуля complexn.

Отметим, что объявление подпрограмм в интерфейсной части автоматически сопровождается их компиляцией с использованием дальней модели памяти. Таким образом, обеспечивается доступ к подпрограммам из основной программы и других модулей.

Следует учесть, что все константы и переменные, объявленные в интерфейсной части модуля, равно как и глобальные константы и переменные основной программы, помещаются компилятором Турбо Паскаля в общий сегмент данных (максимальная длина сегмента 65536 байт).

Порядок появления различных разделов объявлений и их количество может быть произвольным. Если в интерфейсной части объявляются внешние подпрограммы или подпрограммы в машинных кодах, их тела (т.е. зарезервированное слово EXTERNAL, в первом случае, и машинные коды вместе со словом INLINE – во втором) должны следовать сразу за их заголовками в исполняемой части модуля (не в интерфейсной!). В интерфейсной части модулей нельзя использовать опережающее описание.

Исполняемая часть

Исполняемая часть начинается зарезервированным словом **IMPLEMENTATION** и содержит описания подпрограмм, объявленных в интерфейсной части. В ней могут объявляться локальные для модуля объекты – вспомогательные типы, константы, переменные и блоки, а также метки.

Описанию подпрограммы, объявленной в интерфейсной части модуля, в исполняемой части должен предшествовать заголовок, в котором можно опустить список формальных параметров и тип результата для функции, так как они уже описаны в интерфейсной части. Но если заголовок подпрограммы приводится в полном виде, т.е. со списком параметров и объявлением типа результата для функции, то он должен полностью совпадать с заголовком подпрограммы в интерфейсной части, например:

```
Unit complexn;
{-------------------------------}
Interface
Type
Complex= record
    Re, im: real;
End;
Procedure AddC(x,y: complex, var z: complex);
{-------------------------------}
Implementation
Procedure AddC;
z.re:= x.re + y.re;
z.im:= x.im + y.im;
end;
```

end.

Инициирующая часть

Инициирующая часть завершает модуль. Она может отсутствовать вместе с начинающим ее словом BEGIN или быть пустой – тогда вслед за BEGIN сразу следует признак конца модуля.

В инициирующей части размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Эти операторы исполняются до передачи управления основной программе и обычно используются для подготовки ее работы. Например, в инициирующей части могут инициироваться переменные, открываться файлы, устанавливаться связи с другими компьютерами и т.п.:

```
Unit fileText;
{------------------------------}
Interface
Procedure  print(s: string);
{------------------------------}
implementation
var f: text;
const
     name= ‘output.txt’;
procedure print;
begin
   writeln(f, s)
end;
{--------------------------------}
{начало инициирующей части}
begin
   assign(f, name);
   rewrite(f);
{конец инициирующей части}
end.
```

Не рекомендуется делать инициирующую часть пустой, лучше ее опустить.

Компиляция модулей

В среде Турбо Паскаль имеются средства, управляющие способом компиляции модулей и облегчающие разработку больших программ. Определены три режима компиляции: **COMPILE, MAKE, BUILD**. Режимы отличаются способом связи компилируемого модуля или основной программы с другими модулями, объявленными в предложении USES.

При компиляции модуля или основной программы в режиме **COMPILE** все, упоминаемые в предложении USES модули, должны быть предварительно

откомпилированы, и результаты компиляции должны быть помещены в одноименные файлы с расширением TPU (от англ. Turbo Pascal Unit). Файл с расширением TPU создается автоматически при компиляции модуля.

В режиме **MAKE** компилятор проверяет наличие TPU-файлов для каждого объявленного модуля. Если какой-либо файл не найден, система ищет одноименный файл с расширением PAS, т.е. файл с исходным текстом модуля. Если таковой файл найден, система приступает к его компиляции. Кроме того, в этом режиме система следит за возможными изменениями исходного текста любого используемого модуля. Если в PAS-файл внесены изменения, то независимо от того, есть ли в каталоге соответствующий TPU-файл или нет, система откомпилирует его перед компиляцией основной программы. Более того, если изменения внесены в интерфейсную часть, то будут откомпилированы все другие модули, обращающиеся к нему. Режим MAKE существенно облегчает процесс разработки крупных программ с множеством модулей: программист избавляется от необходимости следить за соответствием TPU-файлов их исходному тексту, т.к. система делает это автоматически.

В режиме **BUILD** существующие TPU-файлы игнорируются, система пытается отыскать и откомпилировать соответствующие PAS-файлы для каждого модуля. После компиляции можно быть уверенным, что учтены все сделанные в текстах модулей исправления и изменения.

Подключение модулей к основной программе и их компиляция происходит в порядке их объявления в предложении USES. При переходе к очередному модулю система предварительно ищет все модули, на которые он ссылается. Ссылки модулей друг на друга могут образовывать древовидную структуру любой сложности, однако запрещается явное или косвенное обращение модуля к самому себе. Например, недопустимы следующие объявления:

```
Unit A;                    Unit B;
interface                  interface
uses B;                    Uses A;
…….                        ……
implementation             implementation
……..                      ……
end.                       end.
```

Это ограничение можно обойти, если «спрятать» предложение USES в исполняемые части зависимых модулей:

```
Unit A;                    Unit B;
interface                  interface
…….                        ……
implementation             implementation
uses B;                    Uses A;
……..                      ……
end.                       end.
```

Дело в том, что Турбо Паскаль разрешает ссылки на частично

откомпилированные модули, что приблизительно соответствует опережающему описанию подпрограммы. Если интерфейсные части независимы (это обязательное условие!), Турбо Паскаль сможет идентифицировать все глобальные объекты в каждом модуле, после чего откомпилирует тела модулей обычным способом.

Доступ к объявленным в модуле объектам

Пусть, например, мы создаем модуль, реализующий сложение и вычитание комплексных чисел с помощью процедур:

```
Unit complexn;
Interface
type
   complex= record
          re, im: real;
   end;
procedure AddC (x, y: complex; var z: complex);
procedure SubC (x, y: complex; var z: complex);
const       c: complex= (re: 0.1; im: -1);
implementation
procedure AddC;
begin
  z.re:= x.re + y.re;
  z.im:= x.im + y.im;
end; {AddC}
procedure SubC;
begin
  z.re:= x.re - y.re;
  z.im:= x.im - y.im;
end; {SubC}
end.
```

Текст этого модуля следует поместить в файл complexn.pas. Вы можете его откомпилировать, создав TPU-файл.

В следующей программе осуществляются арифметические операции над комплексными числами:

```
Program primer;
Uses complexn;
Var
     a,b,c: coplex;
begin
     a.re:= 1; a.im:= 1;
     b.re:= 1; b.im:= 2;
AddC(a, b, c);
Writeln (‘сложение :’, c.re: 5:1, c.im: 5:1, ‘i’);
```

```
SubC (a, b, c);
Writeln (‘вычитание :’, c.re: 5:1, c.im: 5:1, ‘i’);
End.
```

После объявления Uses complexn программе стали доступны все объекты, объявленные в интерфейсной части модуля complexn. При необходимости можно переопределить любой из этих объектов, как произошло, например, с типизированной константой ***c***, объявленной в модуле. Переопределение объекта означает, что вновь объявленный объект «закрывает» ранее определенный в модуле одноименный объект. Чтобы получить доступ к «закрытому» объекту, нужно воспользоваться составным именем: перед именем объекта поставить имя модуля и точку. Например:

```
Writeln (complexn.c.re: 5: 1, complexn.c.im: 5: 1);
```

Этот оператор выведет на экран содержимое «закрытой» типизированной константы, объявленной в модуле из предыдущего примера.

Стандартные модули

В Турбо Паскале имеется 8 стандартных модулей, в которых содержится множество различных типов, констант, процедур и функций. Этими модулями являются SYSTEM, DOS, CRT, GRAPH, OVERLAY, TURBO3, GRAPH3. Модули GRAPH, TURBO3, GRAPH3 выделены в отдельные TPU-файлы, а остальные входят в состав библиотечного файла TURBO.TPL. Лишь один модуль SYSTEM подключается к любой программе автоматически, все остальные становятся доступны только после указания их имен в списке подключаемых модулей.

Модуль SYSTEM. В него входят все процедуры и функции стандартного Паскаля, а также встроенные процедуры и функции, которые не вошли в другие стандартные модули (например, INC, DEC, GETDIR и т.п.). Модуль SYSTEM подключается к любой программе независимо от того, объявлен ли он в предложении USES или нет, поэтому его глобальные константы, переменные, процедуры и функции считаются встроенными в Турбо Паскаль.

Модуль PRINTER делает доступным вывод текстов на матричный принтер. В нем определяется файловая переменная LST типа TEXT, которая связывается с логическим устройством PRN. После подключения данного модуля можно выполнить, например, такое действие:

```
Uses printer;
Begin
   Writeln(lst, ‘Турбо Паскаль’);
End.
```

Модуль CRT. В нем сосредоточены процедуры и функции, обеспечивающие управление текстовым режимом работы экрана. С его помощью можно перемещать курсор в любую точку экрана, менять цвет выводимых символов и фона, создавать окна. Кроме того, в данный модуль включены также процедуры «слепого» чтения клавиатуры и управления звуком.

Модуль GRAPH. Содержит набор типов, констант, процедур и функций для управления графическим режимом работы экрана. Этот модуль позволяет создавать различные графические изображения и выводить на экран надписи стандартными или созданными программистом шрифтами.

Модуль DOS. В модуле собраны процедуры и функции, открывающие доступ к средствам дисковой операционной системы MS-DOS.

Модуль OVERLAY. Данный модуль необходим при разработке громоздких программ с перекрытиями. Турбо Паскаль обеспечивает создание программ, длина которых ограничивается лишь основной оперативной памятью. Операционная система MS-DOS оставляет программе около 580 Кбайт основной памяти. Память такого размера достаточна для большинства исполняемых программ, тем не менее, использование программ с перекрытиями снимает это ограничение.

Модули TURBO3 и GRAPH3 введены для обеспечения совместимости с ранней версией системы Турбо Паскаль.

ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Практика: Характеристики компьютера. Данные и системы счисления.

1.1. Типы данных, кодирование и системы счисления.

Кодирование данных: числа, текст, графическая информация, звук.

Код – система условных обозначений или сигналов.

Длина кода – количество знаков, используемых для представления кодируемой информации

Кодирование данных – это процесс формирования определенного представления информации.

Декодирование – расшифровка кодированных знаков, преобразование кода символа в его изображение

Двоичное кодирование – кодирование информации в виде 0 и 1.

Входные данные могут быть различных типов, поэтому важно выбрать унифицированную форму их представления в ЭВМ.

В вычислительной технике принята система **двоичного кодирования** основанная на двоичной системе счисления (цифры 0 и 1). Отсюда и название «bit» (Binary Digit – двоичная цифра).

двоичная логика – «да – нет», «черное – белое», «правда – ложь»

Число используемых битов определяет число реализуемых вариантов.

1 бит - **0 1**

2 бита - **00 01 10 11**

3 бита - **000 001 010 011 100 101 110 111**

Кодирование чисел.

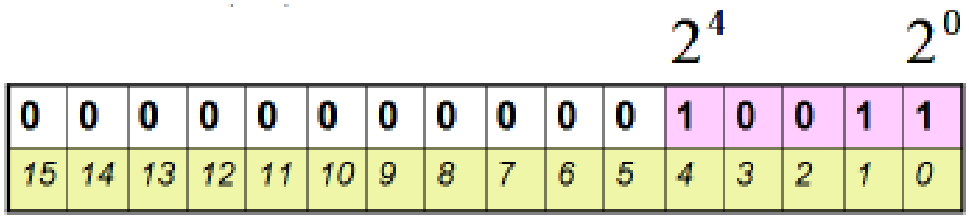
Вопрос о кодировании чисел возникает по той причине, что в машину нельзя либо нерационально вводить числа в том виде, в котором они изображаются человеком на бумаге.

Кодирование целых чисел производиться через их представление в двоичной системе счисления: именно в этом виде они и помещаются в ячейке. Один бит отводиться при этом для представления знака числа (нулем кодируется знак "плюс", единицей - "минус").

Пример: (число 19) нужно представить его степенным рядом по основанию «2» .

19= 1*2^4+ 0*2^3 +0*2^2+1*2^1+1*2^0=16+0+0+2+1,

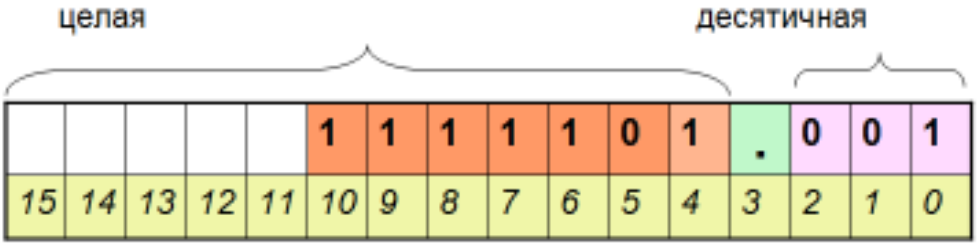
т.е. для числа «19» представления требуется пять (5) двоичных разрядов.



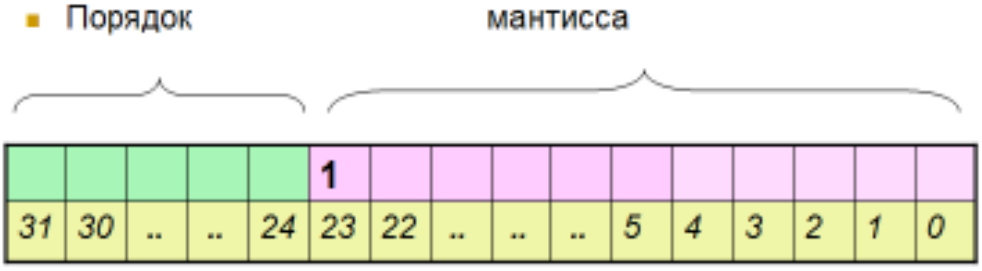
Для кодирования действительных чисел существует специальный формат чисел с плавающей запятой. Число при этом представляется в виде: N = M * qp, где M - мантисса, p - порядок числа N, q - основание системы счисления. Если при этом мантисса M удовлетворяет условию 0,1 <= | M | <= 1 то число N называют нормализованным.

Вещественные числа

Записи числа с фиксированной десятичной точкой - фиксируется число младших двоичных разрядов под десятичную часть вещественного числа, остальные разряды отводятся под целую часть числа. Представим число 125,125.



Число с плавающей точкой состоит из двух битовых полей. Восемь старших разрядов отводятся под порядок двоичного числа, а остальные разряды – под мантиссу. Предварительно производится нормализация числа – отбрасываются все незначащие нули в старших разрядах. Поэтому старший значащий бит мантиссы всегда «1».



Текстовая информация

Способ записи текстовой информации заключается в нумерации алфавита (или символов языка) и хранении полученных целых чисел наравне с обычными целыми числами.

Для кодирования букв и других символов, используемых в печатных документах, необходимо закрепить за каждым символом числовой номер – код. В англоязычных странах используются 26 прописных и 26 строчных букв (A … Z, a …

z), 9 знаков препинания (. , : ! " ; ? ( )), пробел, 10 цифр, 5 знаков арифметических действий (+,-,*, /, ^) и специальные символы (№, %, _, #, $, &, >, <, |, \) – всего чуть больше 100 символов. Таким образом, для кодирования этих символов можно ограничиться максимальным 7-разрядным двоичным числом (от 0 до 1111111, в десятичной системе счисления – от 0 до 127).

Графический объект

В видеопамяти находится двоичная информация об изображении, выводимом на экран. Почти все создаваемые, обрабатываемые или просматриваемые с помощью компьютера изображения можно разделить на две большие части – **растровую и векторную графику.**

Растровые изображения представляют собой однослойную сетку точек, называемых пикселами (pixel, от англ. picture element). Код пиксела содержит информации о его цвете.

В противоположность растровой графике **векторное изображение** многослойно. Каждый элемент векторного изображения – линия. Каждый элемент векторного изображения является объектом, который описывается с помощью математических уравнении. **Сложные объекты** (ломаные линии, различные геометрические фигуры) представляются в виде совокупности элементарных графических объектов.

Звуковая информация

На компьютере работать со звуковыми файлами начали в 90-х годах. В основе цифрового кодирования звука лежит – процесс преобразования колебаний воздуха в колебания электрического тока и последующая дискретизация аналогового электрического сигнала. Кодирование и воспроизведение звуковой информации осуществляется с помощью специальных программ (редактор звукозаписи).

Временная дискретизация – способ преобразования звука в цифровую форму путем разбивания звуковой волны на отдельные маленькие временные участки где амплитуды этих участков квантуются (им присваивается определенное значение).

Это производится с помощью аналого-цифрового преобразователя, размещенного на звуковой плате. Таким образом, непрерывная зависимость амплитуды сигнала от времени заменяется дискретной последовательностью уровней громкости. Современные 16-битные звуковые карты кодируют 65536 различных уровней громкости или 16-битную глубину звука (каждому значению амплитуды звук. сигнала присваивается 16-битный код)

Качество кодирование звука зависит от:

- глубины кодирования звука - количество уровней звука
- частоты дискретизации – количество изменений уровня сигнала в единицу

1.2. Изучение и создание блок-схем.

Алгоритмическая культура - это часть общей математической культуры и общей культуры мышления, предполагающая формирование умений, связанных с пониманием сущности понятия алгоритма и его свойств.

Умение последовательно, четко и непротиворечиво излагать свои мысли тесно связано с умением представлять сложное действие в виде организованной последовательности простых. Такое умение называется *алгоритмическим.* Оно находит свое выражение в том, что человек, видя конечную цель, может составить алгоритмическое предписание или алгоритм, в результате выполнения которого цель будет достигнута.

При этом важно на уроках, посвященных формированию алгоритмической культуры, в процессе формализации концентрировать все внимание учащихся на исполнении алгоритма. Ученики должны убедиться в том, что пошаговое выполнение последовательности команд позволяет им получить ожидаемый результат в том случае, если были четко определены начальные условия.

Составление алгоритмов – сложная задача, поэтому важно уже на начальной ступени образования в школе, ставить целью ее решение, способствуя тем самым развитию логического мышления у школьников.

Следует заметить, что само понятие «алгоритм» чаще всего можно употреблять только условно, т.к. те правила и предписания, которые рассматриваются в начальных классах, не всегда обладают всеми свойствами, его характеризующими. Алгоритмы в начальной школе описывают последовательность действий в конкретном примере, а не в общем виде. В этом случае алгоритм представляет собой полезный инструмент для решения задач, будь то из области математики, общественных дисциплин, естествознания, родного языка или повседневной жизни.

Рассмотрим, какие *требования к знаниям, умениям предъявляются к обучающимся по теме «Алгоритм»*

знать:

1) историю возникновения алгоритма;
2) понятие алгоритма, способы его записи;
3) основные свойства алгоритма (без использования специальной терминологии);
4) базовые структуры алгоритмов.

уметь:

1) исполнять алгоритм (следуя пошаговым предписаниям) для знакомого (или нового) задания и получать конечный результат;
2) изменять алгоритм для выполнения нового (схожего с предыдущим) задания и давать имя новому алгоритму;
3) приводить примеры, когда для выполнения задания можно воспользоваться различными алгоритмами;
4) находить и исправлять ошибки в алгоритме;
5) записывать алгоритм, который использовали при выполнении задания;
6) составлять алгоритм для выполнения задания, аналогичного предыдущему;

7) самостоятельно составлять алгоритм, который может быть исполнен другим человеком, а также уметь продемонстрировать получение ожидаемого результата в процессе исполнения алгоритма.

Тема «Алгоритм» изучается в следующем порядке:

1. Знакомство с историей возникновения понятия алгоритма

Термин «алгоритм» происходит от имени великого математика 9 века Аль-Хорезми, который в своем труде «Арифметический трактат», переведенном в 12 веке с арабского на латынь, изложил правила арифметических действий над числами в позиционной десятичной системе счисления. Эти правила и называли алгоритмами.

2. Формирование представлений об алгоритме.

АЛГОРИТМ – это определенная последовательность действий, выполнение которых позволяет получить решение поставленной задачи. Все действия в алгоритме записываются в повелительной форме (в форме приказа). Примеры алгоритмов: инструкции по использованию техники; медицинские рекомендации; описание гимнастических упражнений и т.д.

3. Знакомство со способами записи алгоритма.

В начальной школе рассматриваются следующие способы записи алгоритмов:

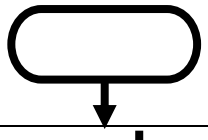
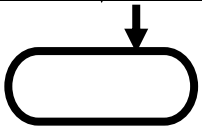
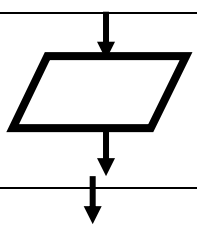
1) Словесный способ – запись алгоритма словами русского языка.

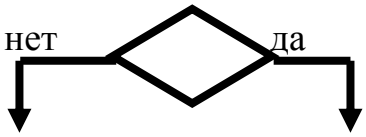
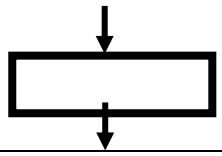
2) Табличный способ – запись алгоритма в форме таблицы.

Например, дано задание: Найдите значение выражения при а=2, 3, 5, 6. В подобном случае решение лучше оформить в виде таблицы.

а	**2**	**3**	**5**	**6**
а+5	**7**	**8**	**10**	**11**

3) Блок – схема – способ записи алгоритмов с помощью специальных блоков.

	название блока	**изображение**
	Блок начала алгоритма.	
	Блок конца алгоритма.	
	Блок ввода данных или сообщения результатов.	

	Блок проверки условия.	
	Блок арифметических операций.	

4. Знакомство с основными свойствами алгоритма

Независимо от решаемой задачи и сферы применения каждый алгоритм должен обладать следующими *свойствами*:

- дискретность, т.е. процесс решения задачи должен протекать в виде последовательности отдельных действий, следующих друг за другом;
- элементарность действий, т.е. каждое действие должно являться настолько простым, что не вызывать сомнений и возможности неоднозначного толкования;
- детерминированность (определенность), т.е. каждое действие должно быть определено однозначно и после его выполнения однозначно определено, какое действие надо выполнять следующим;
- конечность, т.е. алгоритм должен заканчиваться после конечного числа действий (шагов);
- результативность, т.е. в момент прекращения работы алгоритма должно быть известно, что считать его результатом;
- массовость, т.е. алгоритм должен описывать некоторое множество процессов, применимых при различных входных данных.

5. Знакомство с базовыми структурами алгоритма (линейный, условный, циклический).

6. Формирование умения четко исполнять алгоритм.

Например, построение изображений при выполнении заданий на вычерчивание различных фигур (графические диктанты), задания – лабиринты.

7. *Формирование умения проверять правильность алгоритма.* Например, проверьте, правильно ли выполнили вычисления: 54+24-8=70, 40-(13-8)=19. На данном этапе можно также использовать задачи с лишними, недостающими данными, не имеющие решения.

8. *Формирование умения выбирать более рациональный алгоритм, преобразовывать его.* Например: «Даны число х и набор действий: разделить полученное число на 3; умножить х на 2; сообщить результат; прибавить к полученному числу 4; вычесть из полученного числа 7.

Составьте из этих действий два различных алгоритма. Любой ли алгоритм, составленный из этих действий, можно выполнить? Составьте таблицу результатов при различных значениях х».

9. *Формирование умения самостоятельно составлять алгоритм.*

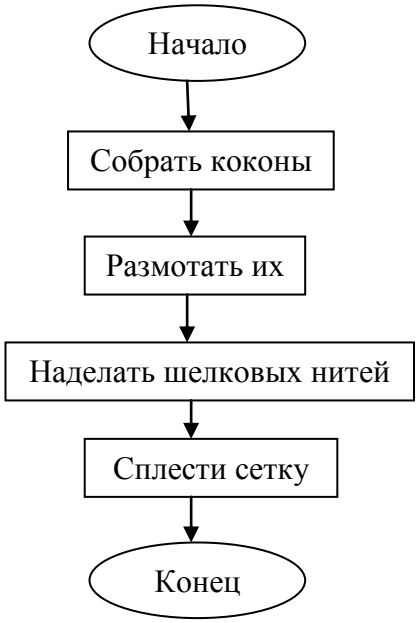
Прочитай, что написано в текстах, как подробно выполнялись эти команды. Составь алгоритм. «… Коротышки могли добывать резину. В городе у них росли цветы, похожие на фикусы. Если на стебле такого цветка сделать надрез, то из него вытекает сок. Этот сок постепенно густеет и превращается в резину, из которой можно делать мячи и калоши».

Более подробно остановимся на блок-схемах алгоритмов. Так как алгоритм на естественном языке дает более подробную информацию, а блок-схема более наглядную, то формирование познавательных УУД, а в данном случае, формирование умения навыков самостоятельно составлять алгоритм более рационально, на наш взгляд, в виде блок-схем.

Изучение начинаем с более легких задач. Например, таких как:

Задача 1. *Прочитай, что написано в тексте и составь алгоритм в виде блок-схемы «… Знайка велел собирать шелковичные коконы, чтобы размотать их и наделать шёлковых нитей. Из этих ниток он велел им сплести огромную сетку».*

Решение:

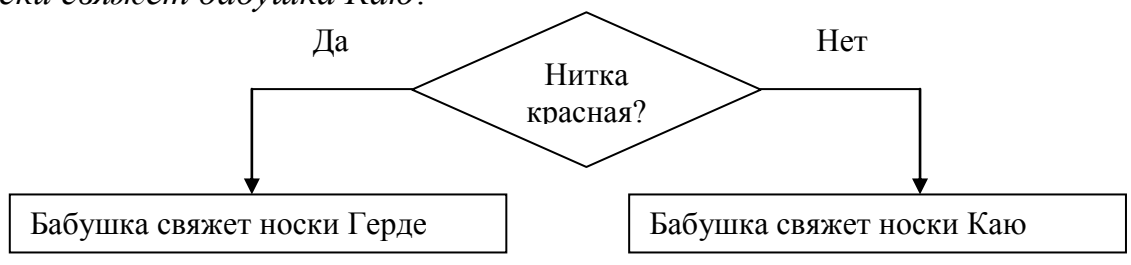


Трудностей при составлении блок-схем линейной структуры практически не возникает, так как обучающиеся хорошо усваивают свойство алгоритма дискретность, в котором шаги записываются и выполняются один за другим.

Проблема возникает при решении задач, которые относятся к классу, в условии которых есть слова «если… , то…».

Например:

Задача 2. *В большой корзине лежали клубки с нитками, чтобы Каю и Герде связать новые носки. Прочитай приведенный алгоритм и ответь, какого цвета носки свяжет бабушка Каю?*



Выбери ответ из предложенных вариантов.

1) Красные

2) Синие

3) Цветные

4) Не красные

Данную задачу нельзя выполнить, осуществляя последовательность шаг за шагом, так как в условии задачи есть слова «если… , то…», поэтому алгоритм решения этой задачи содержит блок проверки условия, в котором один вход и два выхода («Да» и «Нет»).

Задача 3. *Маша попросила Мишу задумать двузначное число и если задуманное число четное, то разделить его на два и назвать результат. Если задуманное число нечетное, то ничего не делать, а просто назвать это число.*

Задача 4. *Даны числа: 4, 26, 10, 55. Используя алгоритм, вычисли результат. Результатов вычисления будет столько же, сколько дано чисел. Нарисуй блок-схему алгоритма вычисления.*

КАК ВЫЧИСЛИТЬ РЕЗУЛЬТАТ.

1. Взять число __

2. Прибавить к числу 10.

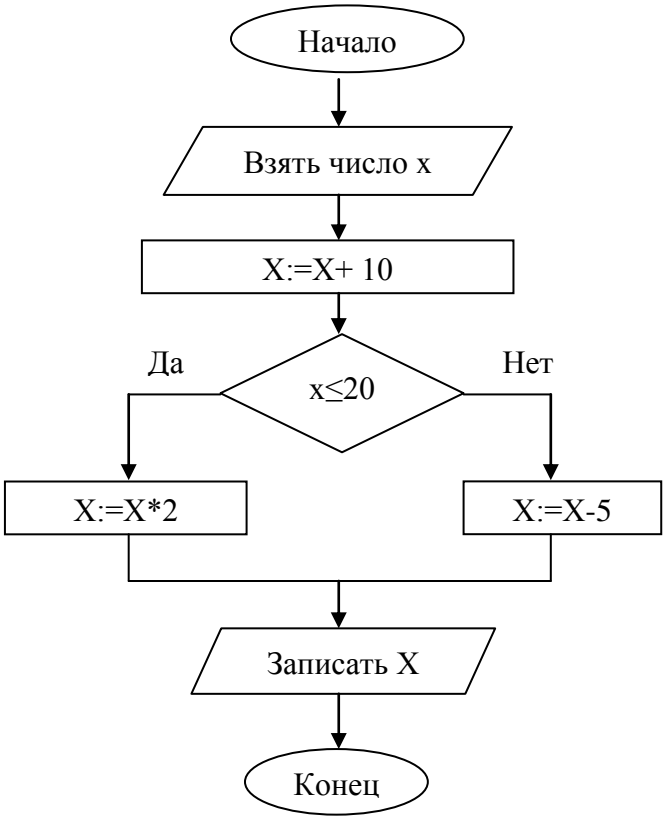
3. Если полученное число меньше или равно 20, то умножить его на 2, иначе перейти к строке № 4.

4. Вычесть 5.

5. Запиши результат в ячейке.

6. Стоп

Решение:



При организации работы, направленной на формирование у младших школьников представлений об алгоритме, учителю рекомендуется использовать следующие *методические приемы*:

1. Обсуждение в классе команд, используемых в алгоритме.
2. Составление алгоритмов, содержащие ошибки. Это дает возможность детям самостоятельно находить ошибки в проверке алгоритма.

Для алгоритмов, составленных учениками, характерны следующие *типичные ошибки*:

- не сформулированы начальные условия (например, «В какую сторону я был ориентирован?»);
- пропущены некоторые шаги;
- не полностью описаны шаги (например, не указано, как далеко вперед надо идти);
- шаги в неправильной последовательности;
- нет проверки условия завершения задания (бесконечные циклы);
- отсутствует имя алгоритма.

3. Оказание консультативной помощи обучающимся при проверке составленных алгоритмов. Например:

- Учитель или ученик выполняет предписания в точности так, как они записаны. При этом выявляются ошибки, а также учащимся становится

понятно, что означает выполнение алгоритма в точном соответствии с его описанием.

- Можно попросить ученика, пропустившего урок, на котором был составлен алгоритм, исполнить его в точном соответствии с описанием. Смог ли ученик выполнить его в соответствии с замыслом класса? Понятны ли ему предписания? Получил ли он ожидаемый результат?

4. Проведение наглядных экспериментов с использованием реальных предметов и действий над ними до составления учащимися алгоритма.
5. Подробное описание предполагаемого результата выполнения алгоритма.

1.3. Создание алгоритмов на базе алгоритмического языка.

алг(алгоритм)	**сим**(символьный)	**дано**	**для**	**да**
арг(аргумент)	**лит**(литерный)	**надо**	**от**	**нет**
рез(результат)	**лог**(логический)	**если**	**до**	**при**
нач(начало)	**таб**(таблица)	**то**	**знач**	**выбор**
кон(конец)	**нц**(начало цикла)	**иначе**	**и**	**ввод**
цел(целый)	**кц**(конец цикла)	**все**	**или**	**вывод**
вещ(вещественный)	**длин**(длина)	**пока**	**не**	**Утв**

Общий вид алгоритма:
алгназвание алгоритма (аргументы и результаты)
даноусловия применимости алгоритма
надоцель выполнения алгоритма
начописание промежуточных величин
| последовательность команд (тело алгоритма)
кон

Часть алгоритма от слова **алг**до слова**нач**называется**<u>заголовком</u>**, а часть, заключенная между словами**нач**и**кон**—**<u>телом</u>**алгоритма.

В предложении **алг**после названия алгоритма в круглых скобках указываются**характеристики**(**арг, рез**) и**тип значения (цел, вещ, сим, лит** или **лог)**всех**входных**(**аргументы**) и**выходных**(**результаты**)**переменных**. При описании массивов (таблиц) используется служебное слово**таб**, дополненное**граничными парами**по каждому индексу элементов массива.

Примеры предложений <u>алг</u>:

- **алг**Объем и площадь цилиндра (**арг вещ**R, H,**рез вещ**V, S)
- **алг**Корни КвУр(**арг веща**, b, c,**рез вещ**x1, x2,**рез лит**t)
- **алг**Исключить элемент(**арг цел**N,**арг рез вещ таб**A[1:N])
- **алг**Диагональ(**арг цел**N,**арг цел таб**A[1:N,1:N],**рез лит**Otvet)

Предложения **дано**и**надо**не обязательны. В них рекомендуется записывать утверждения, описывающие**<u>состояние среды исполнителя алгоритма</u>**, например:

1. **алг**Замена (**арг лит**Str1, Str2,**арг рез лит**Text)
2. **дано**| длины подстрок Str1 и Str2 совпадают

3. **надо**| всюду в строке Text подстрока Str1 заменена на Str2
4.
5. **алг**Число максимумов (**арг цел**N, **арг вещ таб**A[1:N],**рез цел**K)
6. **дано**| N>0
7. **надо**| K - число максимальных элементов в таблице A
8.
9. **алг**Сопротивление (**арг вещ**R1, R2, **арг цел**N,**рез вещ**R)
10. **дано**| N>5, R1>0, R2>0
11. **надо**| R - сопротивление схемы
12.

Здесь в предложениях **дано**и**надо**после знака "|" записаны**комментарии**. Комментарии можно помещать в конце любой строки. Они не обрабатываются транслятором, но существенно облегчают понимание алгоритма.

✓**Оператор присваивания**. Служит для вычисления выражений и присваивания их значений переменным. Общий вид:**A := B**, где знак ":=" означает команду**заменить прежнее значение переменной, стоящей в левой части**,**на вычисленное значение выражения, стоящего в правой части**.

Например, a:=(b+c)*sin(Pi/4); i:=i+1.

✓Для ввода и вывода данных используют команды

- **ввод**имена переменных
- **вывод**имена переменных, выражения, тексты.

✓**Для ветвления**применяют команды**если**и**выбор**, для**организации циклов**— командыдляи**пока**, описанные в разделе 7.9.

Пример записи алгоритма на школьном ая

алгСумма квадратов (**арг цел**n,**рез цел**S)
дано| n > 0
надо| S = 1*1 + 2*2 + 3*3 + ... + n*n
нач цел i
  ввод n; S:=0
  нц дляi**от**1**до**n
    S:=S+i*i
  кц
  вывод "S = ", S
кон

Что такое базовые алгоритмические структуры?

Алгоритмы можно представлять как некоторые структуры, состоящие из отдельных **базовых**(т.е. основных)**элементов**. Естественно, что при таком подходе к алгоритмам изучение основных принципов их конструирования должно начинаться с изучения этих базовых элементов. Для их описания будем использовать язык схем алгоритмов и школьный алгоритмический язык.

Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур:

следование, ветвление, цикл.

Характерной особенностью базовых структур является наличие в них одного входа и одного выхода.

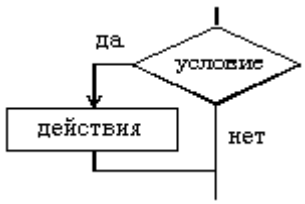
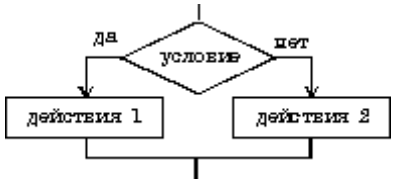
1. **Базовая структура <u>следование</u>**. Образуется из последовательности действий, следующих одно за другим:

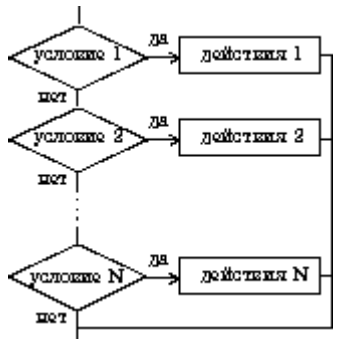
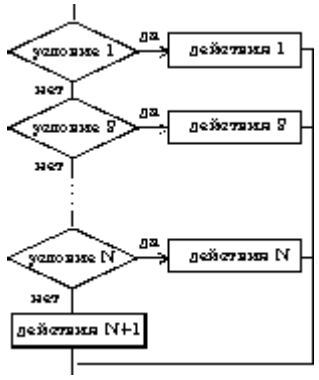
Школьный алгоритмический язык	Язык блок-схем
действие 1 действие 2 . . . . . действие n	

2. **Базовая структура <u>ветвление</u>**. Обеспечивает в зависимости от результата проверки условия (**да**или**нет**) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к**общему выходу**, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран.

Структура **ветвление**существует в четырех основных вариантах:

- если-то;
- если-то-иначе;
- выбор;
- выбор-иначе.

Школьный алгоритмический язык	Язык блок-схем
1. **если-то**	
еслиусловие **то**действия **все**	
2. **если-то-иначе**	
еслиусловие **то**действия 1 **иначе**действия 2 **все**	
3. **выбор**	

выбор **при**условие 1: действия 1 **при**условие 2: действия 2 . . . . . . **при**условие N: действия N **все**	
4. **выбор-иначе**	
выбор **при**условие 1: действия 1 **при**условие 2: действия 2 . . . . . . **при**условие N: действия N **иначе** действия N+1 **все**	

Практические задачи

Задания на соотношение единиц измерения информации

1. 2^{25} бит – сколько Мбайт?
2. Найти значение X из соотношения 4^{2-x}Кб=16Мб
3. Найти X, при котором равны информационные объемы 32^{x+3} килобайт и 256^{x} мегабайт.

Задания на использование формулы Хартли и применение вероятностного подхода к измерению информации

4. Сколько различных звуковых сигналов можно закодировать с помощью 8 бит?
5. Сколько нужно бит, чтобы закодировать алфавит из 64 символов?
6. Когда Вы подошли к светофору, горел желтый свет. Затем зажегся красный. Какой объем информации Вы получили в момент, когда зажегся красный?
7. Какое количество информации несет сообщение о том, что человек живет в первом или втором подъезде, если в доме 16 подъездов?
8. Измеряется температура воздуха, которая может быть целым числом от -30 до 34 градусов. Какое наименьшее количество бит необходимо, чтобы закодировать одно измеренное значение?
9. Метеорологическая станция ведет наблюдение за влажностью воздуха. Результатом одного измерения является целое число от 0 до 100 процентов, которое записывается при помощи минимально возможного количества бит.

Станция сделала 80 измерений. Определите информационный объем в байтах результатов наблюдений.

10. В велокроссе участвуют 779 спортсменов. Специальное устройство регистрирует прохождение каждым из участников промежуточного финиша, записывая его номер с использованием минимально возможного количества бит, одинакового для каждого спортсмена. Каков информационный объем сообщения (в байтах), записанного устройством, после того как промежуточный финиш прошли 280 велосипедистов?
11. Для передачи сигналов на флоте используются специальные сигнальные флаги, вывешиваемые в одну линию (последовательность важна). Какое количество различных сигналов может передать корабль при помощи трех сигнальных флагов, если на корабле имеются флаги четырех различных видов (флагов каждого вида неограниченное количество)?
12. Каждый элемент светового табло может гореть одним из 4 цветов. Какое наименьшее количество элементов должно работать, чтобы можно было передать 500 различных сигналов?
13. Азбука Морзе позволяет кодировать символы для радиосвязи, задавая комбинацию точек и тире. Сколько различных символов (цифр, букв, знаков пунктуации и т.д.) можно закодировать, используя код Морзе длиной не менее пяти и не более шести сигналов (точек и тире)?
14. Некоторое сигнальное устройство за одну секунду передает один из трех специальных сигналов. Какое количество различных сообщений можно передать при помощи этого устройства за четыре секунды?
15. Одна ячейка памяти «троичной ЭВМ» (компьютера, основанного на использовании троичной системы счисления) может принимать одно из трех возможных состояний. Для хранения некоторой величины отвели 6 ячеек памяти. Сколько различных значений может принимать эта величина?
16. В ящике белые и черные шары. Черных среди них 2. Сообщение о том, что достали черный, несет 4 бита информации. Сколько белых шаров в ящике?
17. К празднику надували белые и синие шарики. Белых шариков 24. Сообщение о том, что лопнул синий шарик, несет 2 бита информации. Сколько всего надули шариков?
18. Два исполнителя Шалтай и Болтай проставляют 0 и 1 в каждую из имеющихся в их распоряжении клеточку. Шалтай может закодировать 512 символов и у него на две клеточки больше, чем у Болтая. Сколько клеток в распоряжении у Болтая?
19. Каждая клетка поля 8×8 кодируется минимально возможным и одинаковым количеством бит. Решение задачи о прохождении «конем» поля записывается последовательностью кодов посещенных клеток . Каков объем информации в битах после 11 сделанных ходов? (Запись решения начинается с начальной позиции коня).
20. Учитель, выставляя в журнал четвертные оценки по биологии за третью четверть (3, 4, 5), обратил внимание, что комбинация из трех четвертных

оценок по этому предмету у всех учеников различна. Какое может быть максимальное количество учеников в этом классе?

21. В ящике находится 32 теннисных мяча, среди которых есть мячи желтого цвета. Наудачу вынимается один мяч. Сообщение «извлечен мяч НЕ желтого цвета» несет 4 бита информации. Сколько желтых мячей в ящике?
22. В некоторой стране автомобильный номер длиной 6 символов составляют из заглавных букв (задействовано 30 различных букв) и десятичных цифр в любом порядке. Каждый такой номер в компьютерной программе записывается минимально возможным и одинаковым целым количеством байт (при этом используют посимвольное кодирование и все символы кодируются одинаковым и минимально возможным количеством бит). Определите объем памяти в байтах, отводимый этой программой для записи 50 номеров.
23. Программа генерирует N-символьные пароли следующим образом: в качестве символов используются десятичные цифры, а также строчные и прописные латинские буквы в любом порядке (в латинском алфавите 26 знаков). Все символы кодируются одним и тем же минимально возможным количеством бит и записываются на диск. Программа сгенерировала 128 паролей и записала их в файл подряд, без дополнительных символов. Размер полученного файла составил 1,5 Кбайта. Какова длина пароля (N)?
24. В ящике лежат красные, белые и черные кубики. Сообщение о том, что достали красный кубик, несет 5 бит информации. Вероятность извлечения черного кубика в 2 раза больше, чем красного. Сколько информации несет сообщение об извлечении черного кубика?
25. Склад сети магазинов Медиамания получил от поставщика партию телевизоров, компьютеров и музыкальных центров. Из них 27 телевизоров. Для проверки качества поступившей аппаратуры товаровед случайным образом выбирает одну из поступивших на склад коробок. Информационный объем сообщения «Для проверки выбран не телевизор» равен $4-\log_2 7$ бит. Количество информации в сообщении «Для проверки выбран не компьютер» равно $\log_2 3-1$ бит. Найти количество поступивших на склад компьютеров.
26. Даны длины сторон треугольника A, B, C. Найти площадь треугольника S. Составьте блок-схему алгоритма решения поставленной задачи.
27. Даны координаты вершин треугольника ABC. Найти его площадь. Составьте блок-схему алгоритма решения поставленной задачи.
28. В квадратной комнате шириной A и высотой B есть окно и дверь с размерами C на D и M на N соответственно. Вычислите площадь стен для оклеивания их обоями. Составьте блок-схему алгоритма решения поставленной задачи
29. Дана величина A, выражающая объем информации в байтах. Перевести A в более крупные единицы измерения информации. Составьте блок-схему алгоритма решения поставленной задачи.
30. Вычислить путь, пройденный лодкой, если ее скорость в стоячей воде v км/ч, скорость течения реки v1 км/ч, время движения по озеру t1 ч, а против течения реки – t2 ч. Составьте блок-схему алгоритма решения поставленной задачи.

31. Вычислите значение функции Y при X=2, используя блок-схему алгоритма. Y = 2 РЕШЕНИЕ: 1. X = 2 2. Z = 8 * 2 = 16 3. Z = = 4 4. Z = 4 – 1 = 3 5. Y = 3 * 2 = 6 6. Y = 6 / 3 = 2

Практика: Алгоритмика и программирование.
2.1. Изучение системы(среды) программирования. Простые операторы и создание линейных программ.

Среда разработки Free Pascal предназначена для разработки программ и включает в себя: текстовый редактор, транслятор и отладчик.

Запуск программы в Windows:

Пуск → Программы → Free Pascal→Free Pascal.

Среда программирования Free Pascal

Назначение команд главного меню.

Меню	Назначение
File	работа с файлами (создание, сохранение и т.д.)
Edit	редактирование фрагментов текста (копирование, выделение, вставка и т.д.)
Search	поиск фрагментов текста, процедур, ошибок
Run	запуск программ на выполнение в различных режимах
Compile	осуществляется компиляция файла, находящегося в активном окне
Debug	меню отладки программ
Tools	меню инструментальных средств позволяет задать программы, которые можно запустить не выходя из интегрированной среды, и запускать эти программы
Options	установка параметров среды разработки
Window	позволяет открывать, закрывать, активизировать окна, изменять их размеры
Help	справка

Работа с файлами (меню **File**)[1]:

Команда	Назначение	Горячие клавиши
New	Открыть окно для создания новой программы	-
New from Template	Открыть окно для создания новой программы с использованием шаблона	-
Open	Открыть ранее созданный файл	F3
Save	Сохранить файл	F2
Save as	Сохранить файл с другим именем	-
Save all	Сохранить все открытые файлы	-
Change dir	Изменить текущий каталог	-
Exit	Выход из среды программирования	Alt + X

1: здесь и далее перечислены только основные команды, которые обязательно понадобятся начинающему программировать в данной среде.

Работа с блоками текста(меню **Edit**):

Команда	Назначение	Горячие клавиши
Undo	Отменить исправление	-
Redo	Вернуть исправленное	-
Copy	Копировать фрагмент в буфер;	Ctrl + C (Ctrl + Inc)
Cut	Вырезать фрагмент в буфер;	Ctrl + X (Shift + Del)
Paste	Вставить фрагмент из буфера;	Ctrl + V (Shift + Ins)
Clear	Очистить буфер	Ctrl + Del (Ctrl + Del)
Select All	Выделить весь текст в окне	-
Unselect	Отменить выделение	-
Show clipboard	Открыть буфер обмена программы	-

Компиляция (**Compile**) и запуск программ(**Run**):

Команда	Назначение	Горячие клавиши
Compile/Compile	Проверка программы на наличие синтаксических ошибок и её компиляция	Alt + F9
Run/Run	Компиляция и запуск программы	Ctrl + F9
Run/Step Over	Выполнить одну строку программы	F8
Run/Trace into	Выполнить одну строку программы или подпрограммы	F7
Run/Goto Cursor	Выполнить программу с начала до конца строки	F4

Отладка программы (**Debug**):

Команда	Назначение	Горячие клавиши
User screen	Открыть экран пользователя	Alt + F5
Add Watch	Добавить переменную в окно отладки	Ctrl + F7
Watches	Окно отладки	-

Управление окнами (**Window**):

Команда	Назначение	Горячие клавиши
Size/Move	Изменить размер окна/Переместить окно	Ctrl + F5
Zoom	Развернуть/Восстановить окно	F5
Next	Перейти к следующему окну	F6
Previous	Перейти к предыдущему окну	Shift + F6
Close	Закрыть окно	Alt + F3

Создание программа в среде Free Pascal по шагам.

ШАГ 1. Запуск

Запуск программы в Windows:

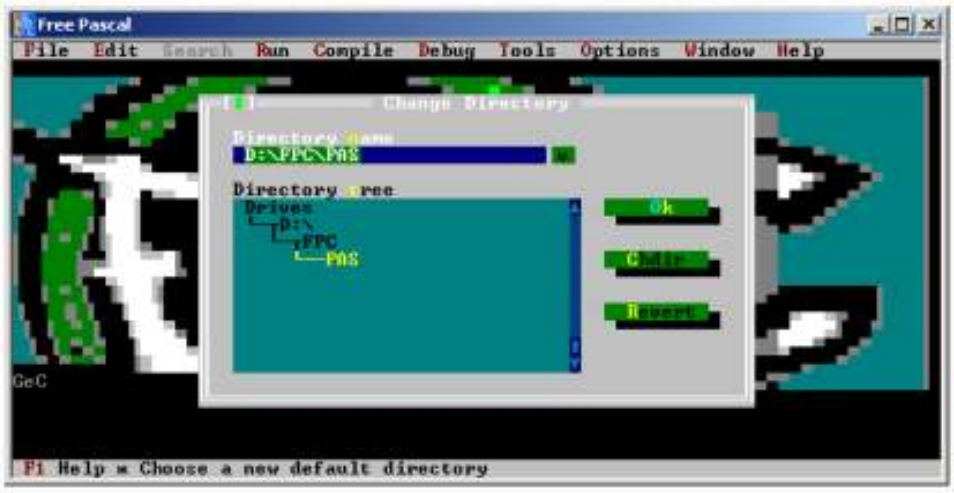
Пуск → Программы → Free Pascal→Free Pascal.

Среда программирования Free Pascal

ШАГ 2. Изменить текущий каталог

Команда ***File → Change dir...***

В поле ***Directory name*** введите имя папки и путь к ней или выберите её в поле ***Directory tree***.



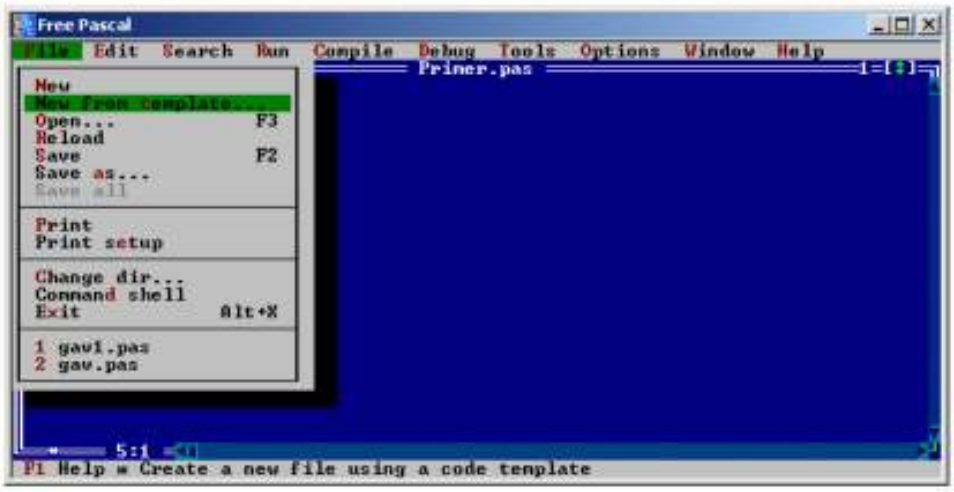
Сохраняйте свои программы в специальной папке. Это в дальнейшем облегчит Вам их поиск.

ШАГ 3. Открыть новое окно

Команда ***File → New***

Созданному файлу по умолчанию будет присвоено стандартное имя ***noname00.pas***. Если это вторая программа, то – ***noname01.pas*** и т.д.

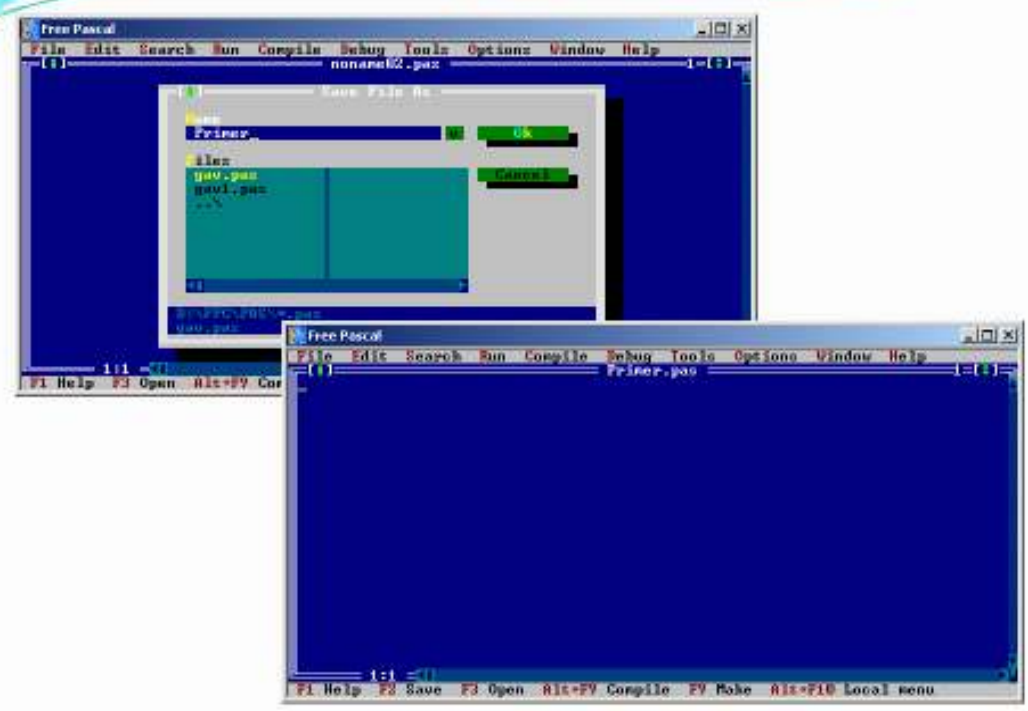
С помощью команды *File → New from template...* можно создать программу с использованием шаблона.



Нужно выбрать интересующий шаблон (например, *Program*) и нажать кнопку «*Ok*»

Ввести имя программы.

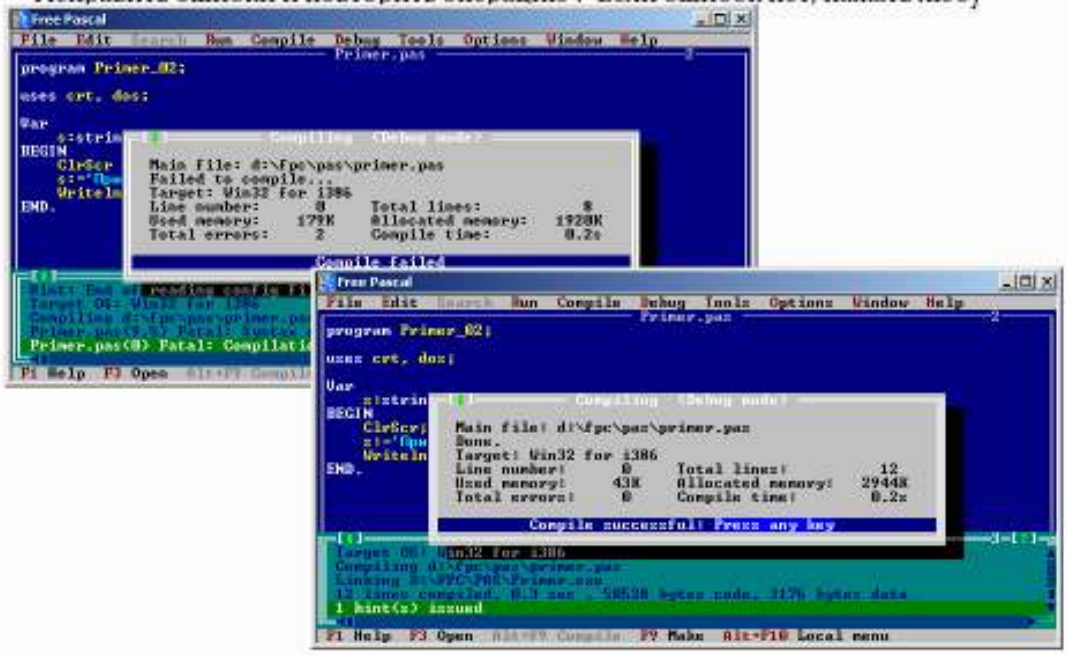
Ввести имя файла и нажать кнопку «Ok»

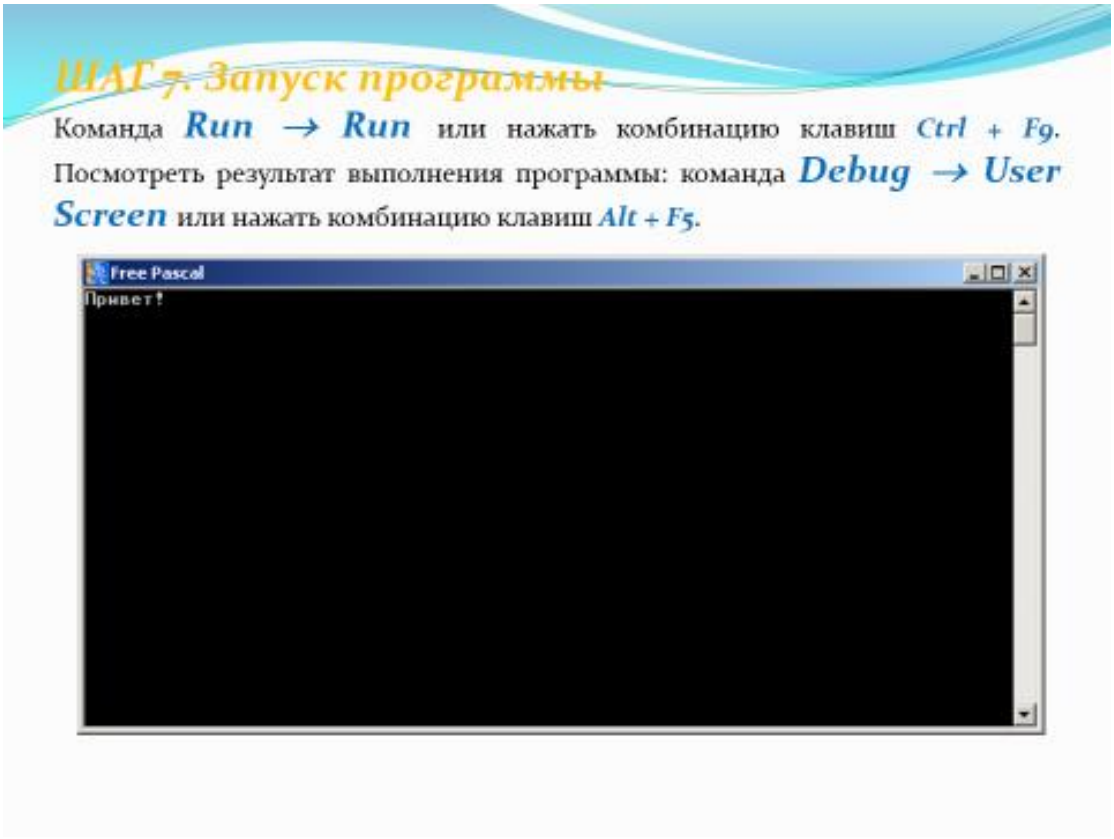


ШАГ 5. Набрать текст программы

ШАГ 6. Компиляция программы

Команда **Compile → Compile** или нажать комбинацию клавиш *Alt + F9*.
Исправить ошибки и повторить операцию . Если ошибок нет, нажать любу





2.2. Составные операторы, описание разветвляющихся процессов.

Разветвления в программах используют операторы перехода, условный и выбора варианта.

1) Оператор безусловного перехода GOTO позволяет изменять последовательность выполнения операторов в программе.

Формат записи оператора:

GOTO <метка>; где

<метка> - имя метки, отличающей ту строку, на которую требуется выполнить переход.

Имя метки должно быть предварительно описано в разделе описания меток Label.

Неправильное использование в программе оператора GOTO усложняет читаемость программы.

Например:

```
Label 1,3,8,10;
. . . . .
Goto 8
. . . . .
8:  x : = A+ B
```

2) Составной оператор (begin … end) представляет собой совокупность последовательно выполняемых операторов, заключенных в операторные скобки begin … end.

Этот оператор записывается следующим образом:

```
begin
        <оператор 1>;
        <оператор 2>;
        . . . . . .
        <оператор N>
end;
```

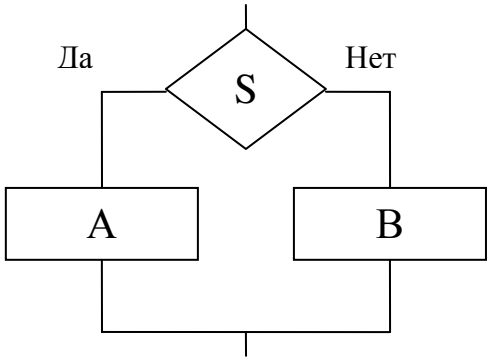
Составной оператор используется в тех случаях, когда в соответствии с правилами записи языка TURBO PASCAL можно записать только один оператор, в то время как требуется выполнить несколько операторов.

Отдельные операторы внутри составного оператора отделяются друг от друга точкой с запятой. Перед символом **end** точку с запятой можно не ставить, т. к. это слово не является отдельным оператором. Если же « ; » будут поставлены, то они будут рассматриваться как пустой оператор, т. е. оператор не выполняющий никакого действия. Сам блок операторов программы является (можно считать) составным оператором, т. к. он тоже заключен в операторные скобки BEGIN . . . END, внутри которых располагается последовательность операторов, разделенных « ; ».

Например:

```
Begin
A:=2;
B:=x+y;
End;
```

3) Условный оператор IF обеспечивает в зависимости от условия выбор одного из возможных действий.



Существует два варианта записи оператора:

a) Полный вариант:

```
If  S   then      A   else   B  ;
```

b) Укороченный вариант:

```
If  S   then      A  ;
```

где: S – условие (логическое выражение), которое проверяется на истинность.

A – оператор (только один!), выполняющейся, если выражение S – истинно.

B – оператор (только один!), выполняющейся, если выражение S – ложно.

Если вместо A или B требуется использовать несколько операторов, то применяется составной оператор (BEGIN . . . END).

Примеры использования оператора IF приведены в Приложении 1.

4) Оператор выбора варианта Case … of … end.

Используется для обработки ситуаций с несколькими вариантами решения (путем выбора одного из нескольких операторов), выбираемых в зависимости от некоторого выражения, называемого селектором.

Существует две форму записи этого оператора:

1) Case S of	2) Case S of
c1: <оператор 1>;	c1: <оператор 1>;
c2: < оператор 2>;	c2: < оператор 2>;
…….	…….
cN: < оператор N>	cN: < оператор N>
else < оператор>	end;
end;	

где S – выражение порядкового типа, значение которого вычисляется;

cl,c2,...,cN—это константы или метки, с которыми сравниваются значения выражения селектора (S). *Они должны быть одного гита с селектором.*

Выбор оператора определяется совпадением значения селектора S и константы (метки cl,c2,...,cN), стоящей перед оператором.

< оператор 1 >, < оператор 2 >, < оператор N > — один оператор, либо составной оператор (begin... end),

< оператор > — оператор, который выполнится, если значение выражения S не совпадет ни с одной из меток (констант) cl,c2,...,cN.

Примеры использования оператора выбора варианта приведены в Приложении 2.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Информация, необходимая для выполнения лабораторной работы, приведена в соответствующих разделах. Порядковый номер студента по списку группы соответствует номеру варианта. По приведенному заданию требуется:

1. Определить порядок реализации вычисление.
2. Составить графическую схему алгоритма.
3. Записать программу на языке (Turbo Pascal).
4. Выполнить расчеты на ЭВМ.
3. Написать отчет по лабораторной работе.

Задание

Вычислить и вывести на экран значение заданной функции (варианты заданий взять из таблицы). Проверить программу на ЭВМ.

№ Вариант	Функция	Условие	Исходные данные
1	2	3	4
1	$y=\begin{cases} at^2\ln(t) \\ \sin(t)+b \\ e^{at}\cos(bt) \end{cases}$	1 ≤ t ≤ 2 t < 1 t > 2	a = - 0,5 b = 2
2	$y=\begin{cases} \pi x^2-\dfrac{7}{x^2} \\ ax^3+7\sqrt{x} \\ \lg(x+7\sqrt{x}) \end{cases}$	x < 1.3 x = 1.3 x > 1.3	a = 1.5
3	$\omega=\begin{cases} ax^2+bx+c \\ \dfrac{a}{x}+\sqrt{x^2+1} \\ \dfrac{a+bx}{\sqrt{x^2+1}} \end{cases}$	x < 1.2 x = 1.2 x > 1.2	a = 2.8 b = -0.3 c = 4

1	2	3	4
4	$Q=\begin{cases} \pi x^2-\dfrac{7}{x^2} \\ ax^3+7\sqrt{x} \\ \ln\left(x+7\sqrt{\lvert x+a\rvert}\right) \end{cases}$	x < 1.4 x = 1.4 x > 1.4	a = 1.65
5	$y=\begin{cases} 1.5\cos^2 x \\ 1.8ax \\ (x-2)^2+6 \\ 3tgx \end{cases}$	x < 1 x = 1 1< x< 2 x > 2	a = 2.3
6	$\omega=\begin{cases} x^3\sqrt{x+a} \\ x\cdot\sin ax \\ e^{-ax}\cdot\cos ax \end{cases}$	x < a x = a x > a	a = 2.5
7	$Q=\begin{cases} bx-\lg bx \\ \cos x^2+bx^3 \\ bx+\lg bx \end{cases}$	x < 1 x = 1 x > 1	b = 1.5
8	$y=\begin{cases} \sin x\lg x \\ \cos^2 x \end{cases}$	x > 3.5 x ≤ 3.5	-

9	$f=\begin{cases}\lg(x+1)\\ \sin^2\sqrt{\lvert ax\rvert}\end{cases}$	x > 1 x ≤ 1	a = 20.3
10	$z=\begin{cases}\dfrac{\ln x^3+x^2}{\sqrt{x+t}}\\ \sqrt{x+t}+\dfrac{1}{x}\\ \cos x+t\sin^2 x\end{cases}$	x < 0.5 x = 0.5 x > 0.5	t = 2.2
11	$s=\begin{cases}\dfrac{a+b}{e^x+\cos x}\\ \dfrac{a+b}{x+1}\\ e^x+\sin x\end{cases}$	x < 2.8 2.8 ≤x< 6 x > 6	a = 2.6 b = -0.39
12	$y=\begin{cases}a\lg x+\sqrt{\lvert x\rvert}\\ 2a\cos x+3x^2\end{cases}$	x > 1 x ≤ 1	a = 0.9
1	2	3	4
13	$\omega=\begin{cases}\dfrac{a}{i}+bi^2+c\\ \sin(i)+1\\ ai+bi^3\end{cases}$	i < 4 4 ≤i≤ 6 i > 6	a = 2.1 b = 1.8 c = -20.5
14	$z=\begin{cases}a\sin\left(\dfrac{i^2+1}{n}\right)\\ \cos\left(i+\dfrac{1}{n}\right)\end{cases}$	$i > 0$ $i \le 0$	a = 0.3 n = 10
15	$z=\begin{cases}\sin(x)\\ \cos(x)\\ tg(x)\end{cases}$	x < 2 2 <x< 5 x > 5	-
16	$\omega\begin{cases}\sqrt{at^2+b\sin(t)+1}\\ at+b\\ \sqrt{at^2+b\cos(t)+1}\end{cases}$	t < 0.1 t = 0.1 t > 0.1	a = 2.5 b = 0.4
17	$z=\begin{cases}c*\cos\dfrac{i^2+1}{n}\\ \sin\dfrac{i^2+1}{n}\end{cases}$	$\cos\dfrac{i^2+1}{n}>0$ $\cos\dfrac{i^2+1}{n}<0$	c = 0.5 n = 10

Пример выполнения работы

1) Задание: Вычислить и вывести на экран значение заданной функции

$$s=\begin{cases}\dfrac{a}{t}+b, & если \quad t<1\\ \sin(t)+c, & если \quad t=1\\ \cos(b*t) & если \quad t>1\end{cases}$$

a = 5; b = 7; c = 8.

Проверить программу на ЭВМ.

Решение

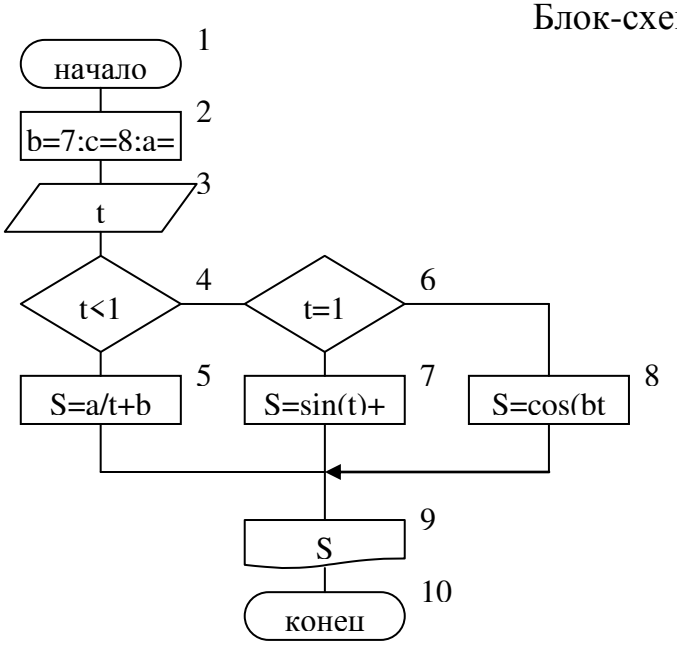
Программа на языке TURBO PASCAL:

```
Program lab_2;
Uses Crt;
Const a = S; b = 7; c = 8;
Var
    t,s:real;
BEGIN
        Clrscr ;
        Writeln ( 'Введите t ');
        Read(t);
         If  t < l  then s : = a/i+b
                  else  if t = 1 then s : = sin(t)+c
                                 else s : = cos(b*t);
         Write( ' s = ', s:8:3);
END.
```

Результаты счета:

t1=0,7	s1 =14443
t2=1	s2=8,841
t3=1,3	s3=-0,948

Блок-схема алгоритма



2.3. Индексные переменные, их описание и обработка.

1. Определение и типы массивов

Массив - это множество однотипных элементов, объединённых общим именем и занимающих в компьютере определённую область памяти. Количество элементов в массиве всегда конечно. В общем случае массив - это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип.

Другими словами можно сказать, что массив представляет собой фиксированное количество упорядоченных однотипных компонент, снабженных индексами, т.е. является совокупностью конечного числа данных одного типа. В качестве элементов массива можно использовать любой тип данных, поэтому вполне правомерно существование массивов записей, массивов указателей, массивов строк, массивов и т.д.

Массивы могут быть:

- одномерными (одна строка – несколько столбцов);
- многомерными (несколько строк – несколько столбцов).

Для создания массива его предварительно необходимо описать либо в разделе var, либо в разделе type. Для задания массива используется зарезервированное слово **array**, после которого указывается тип индекса (-ов) компонент (в квадратных скобках) и после слова **of -** тип самих компонент:

Type
<имя массива>=array[<тип индекса(-ов)>] of <тип компонент>;
Или
Var
<имя массива>:array[<тип индекса(-ов)>] of <тип компонент>;

Введя тип массив, можно задавать переменные или типизированные константы этого типа. Размерность массива может быть любой, компоненты массива могут быть любого, в том числе и структурированного, типа; индекс может быть любого порядкового типа, кроме типа longint.

При задании значений константе-массиву компоненты указываются в круглых скобках и разделяются запятыми, причем, если массив многомерный, внешние круглые скобки соответствуют левому индексу, вложенные в них круглые скобки - следующему индексу и т.д.

Например,

```
Type
   arr = array [1..3] of real;
   matrix = array [1..3, 1..2] of integer;
Const
  mas1: arr = (1, 2, 3);
  mas2: matrix = ((1, 2), (3, 4), (5, 6));
```

1	2
3	4
5	6

массив mas2

Тип массив можно вводить и непосредственно при определении соответствующих переменных или типизированных констант.

Например,

```
Var
   m1, m2 : array [1..3] of integer;
   matr : array [1..3, 1..3] of real;
```

Доступ к компонентам массива осуществляется указанием имени массива, за которым в квадратных скобках помещается значение индекса (-ов) компоненты. В общем случае каждый индекс компоненты может быть задан выражением соответствующего типа.

Например, m1 [2], matr[i,j].

Для обработки массива и последовательного доступа к данным, как правило, используется цикл FOR.

Например,

```
for i:=1 to 10 do read(mas[i]);
```

Обработка элементов двумерного массива обычно выполняется с помощью двойного цикла. Один цикл управляет перебором номеров строк, другой - столбцов.

Например,

```
for i:=1 to 10 do
for j:=1 to 10 do read(mas[i, j]);
```

Над элементами массива можно производить те же операции, которые допустимы для данных его базового типа. Если два массива имеют одинаковые типы индексов и одинаковые типы элементов, то к ним применимы булевы операции (<>=).

2. Основные операции обработки массивов

2.1 Определение размерности массива, заполнение массива

Размерностью массива называется количество ячеек данного массива. При описании массива необходимо указывать конкретное число ячеек, но при реализации задачи не всегда необходимо заполнять все зарезервированные ячейки.

Например,

Задача 1. «Дан одномерный массив из 10 компонент...» - эта формулировка означает, что и при описании и при обработке массива всегда будут использоваться 10 ячеек.

Задача 2. «Дан массив размерность N…» - данная формулировка означает, что размерность массива будет определяться самим пользователем. Т.е. от разработчика такой программы требуется:

- определить максимальную размерность массива (как правило, вполне достаточно 100 ячеек);
- дать возможность пользователю указать количество требуемых ячеек (writeln(“Введите размерность массива”);
  readln(n)- теперь n обозначает размерность).

Для того чтобы заполнить массив необходимо последовательно перебрать все ячейки (компоненты) массива и записать в них некоторые значения. Для перебора ячеек массива используется цикл **for**, в котором с помощью счетчика перебираются индексы ячеек. Заносимые в массив данные могут запрашиваться как с клавиатуры, так и выбираться случайным образом с помощью генератора случайных чисел Random.

Например,

Способ 1. Заполнение одномерного массива с клавиатуры

```
writeln(“Введите размерность массива”);
readln(n);
For i:=1 to n do
Begin
Writeln(“Введите ”,i,” элемент массива”);
Readln(mas[i]);
End;
```

Ход выполнения:

	i =	**Writeln**	**Readln**	**Действие**
Шаг 1	1	Введите 1 элемент массива	Mas[1]	Считываем число в 1 ячейку
Шаг 2	2	Введите 1 элемент массива	Mas[2]	Считываем число в 2 ячейку
Шаг 3	3	Введите 1 элемент массива	Mas[3]	Считываем число в 3 ячейку

…	…	…	…	…
Шаг n	n	Введите n элемент массива	Mas[n]	Считываем число в последнюю ячейку

Способ 2. Заполнение массива случайными числами

```
writeln("Введите размерность массива");
readln(n);                  {определяем размерность массива}
Randomize;                  {включаем генератор случайных чисел}
For i:=1 to n do            {начинаем перебирать массив}
Mas[i]:=random(100)         {выбор любого числа из указанного диапазона и размещение его в массиве}
```

Ход выполнения:

	i =	**Random**	**Mas[i]**	**Действие**
Шаг 1	1	Любое число до 100	Любое число до 100	Считываем выбранное число в массив
Шаг 2	2	Любое число до 100	Любое число до 100	Считываем число в 2 ячейку
Шаг 3	3	Любое число до 100	Любое число до 100	Считываем число в 3 ячейку
…	**…**			**…**
Шаг n	n	Любое число до 100	Любое число до 100	Считываем число в последнюю ячейку

Примечание: *в случае, если размерность массива известна заранее, то чикл for выполняется от 1 до определенного числа.*
Например*, for i:=1 to 5 do – перебор 5 ячеек массива.*

2.2 Вывод массива на экран

Для вывода массива необходимо последовательно перебрать все ячейки (компоненты) массива и вывести лежащие там значения на экран с помощью оператора **write / writeln**.

Оператор **write** выведет значения массива в строку

Оператор **writeln** выведет значения массива в столбец

Для перебора ячеек массива используется цикл **for**, в котором с помощью счетчика перебираются индексы ячеек.

Способ 1. Вывод одномерного массива размерностью 3 с помощью оператора writeln

```
Writeln('Массив');
For i:=1 to 3 do
Writeln(mas[i]);
```

```
Массив
1
2
3
```

Экранное представление

Способ 2. Вывод одномерного массива размерностью 3 с помощью оператора write
Writeln(‘Массив:’);
For i:=1 to 3 do
Write(mas[i],’ ‘);

Экранное представление

Массив
1 2 3

2.3 Поиск требуемого элемента в массиве

Общий алгоритм поиска в массиве определенного элемента можно представить следующим образом:

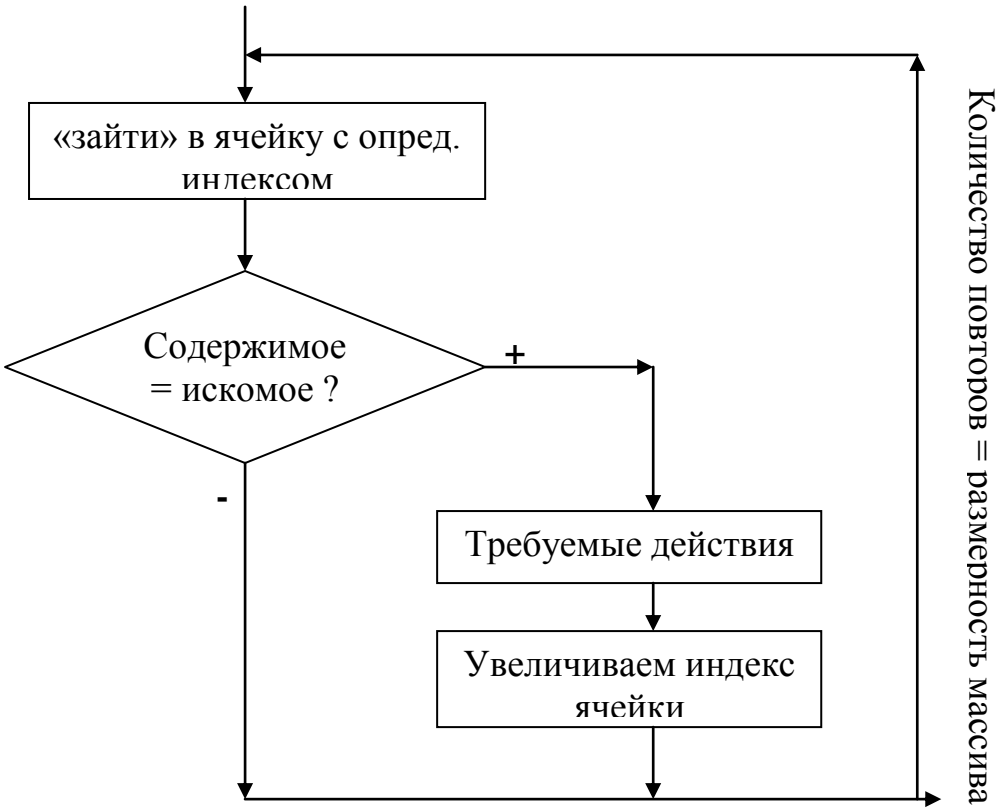


Рисунок 1. Блок-схема поиска требуемого элемента в массиве

Например,
Дан одномерный массив из 7 ячеек. Определить, сколько в нем чисел кратных 7.

```
Var
mas:array[1..7] of integer;
i:integer;          {необходима для перебора массива}
kol:integer;        {количество подходящих элементов}
begin
for i:=1 to 7 do
```

```
begin
write('Введите' ,i, ' элемент');
readln(mas[i]);      {заполняем массив}
if (mas[i] mod 7 =0) and (mas[i]<>0) then kol:=kol+1
{если элемент делится на 7 и в остатке ноль и при этом сам элемент не равен нулю, то увеличиваем счетчик kol}
end;
writeln("Количество чисел кратных 7 -", kol) {вывод результата}
end.
```

Ход выполнения:

Введенный массив

9	-7	0	14	5	21	-5	Элементы
1	2	3	4	5	6	7	Индексы

	i =	**Readln(mas[i])**	**Проверка (mas[i] mod 7 =0) and (mas[i]<>0)**	**Действие**
Шаг 1	1	Mas[1]=9	ложь	Kol=0
Шаг 2	2	Mas[2]=-7	истина	Kol=0+1
Шаг 3	3	Mas[3]=0	ложь	Kol=1
Шаг 4	4	Mas[4]=14	истина	Kol=1+1
Шаг 5	5	Mas[5]=5	ложь	Kol=2
Шаг 6	6	Mas[6]=21	истина	Kol=2+1
Шаг 7	7	Mas[7]= -5	ложь	Kol=3
Вывод результата на экран				

2.4 Поиск максимального и минимального элементов массива

Общий алгоритм работы можно представить в следующем виде:

- определяются переменные (max и min), в которых в результате выполнения алгоритма разместятся соответственно максимальное и минимальное значения;
- начинается перебор массива с одновременной проверкой «является ли просматриваемый элемент больше максимального или меньше минимального»; в случае, если условие истинно – значение элемента массива присваивается соответствующей переменной;

Основной алгоритм выглядит следующим образом:

```
Max:= - 32000;
Min:= 32000;
For i:=1 to n do
begin
If mas[i]>max then max:=mas[i];
If mas[i]<min then min:=mas[i];
End;
```

Ход выполнения:

Введенный массив

9	-7	0	-10	Элементы
1	2	3	4	Индексы

	i =	Данные	Условие	Ответ	Действие
Шаг 1	1	Max=-32000 Min=32000 Mas[1] = 9	9 > -32000 9 < 32000	истина истина	Max=9 Min=9
Шаг 2	2	Max=9 Min=9 Mas[2]= - 7	-7 > 9 -7 < 0	ложь истина	Max=9 Min= - 7
Шаг 3	3	Max=9 Min= - 7 Mas[3]=0	0 > 9 0 < -7	ложь ложь	Max=9 Min= - 7
Шаг 4	4	Max=9 Min= - 7 Mas[4] = - 10	-10 > 9 - 10 < -7	ложь истина	Max=9 Min= - 10

2.5 Сортировка элементов массива

Существует много алгоритмов сортировки массива, но наиболее простым и понятным является сортировка методом «пузырька», при которой самый «легкий» элемент «всплывает», а самый тяжелый «тонет».

Например,

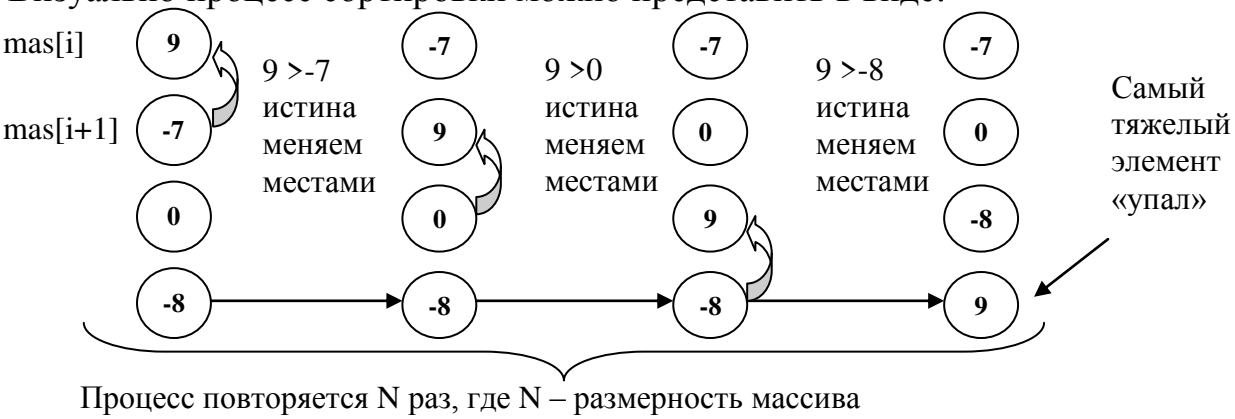
Дан массив

9	-7	0	-8	Элементы
1	2	3	7	Индексы

Необходимо расположить эти элементы в порядке возрастания, т.е. в результате работы программы необходимо получить массив

-8	-7	0	9	Элементы
1	2	3	7	Индексы

При сортировке методом «пузырька» сравниваются два соседних элемента (mas[i] и mas[i+1]). Если mas[i] > mas[i+1], то происходит перестановка элементов. Визуально процесс сортировки можно представить в виде:



Процесс повторяется N раз, где N – размерность массива

Рисунок 2. Сортировка методом «пузырька»

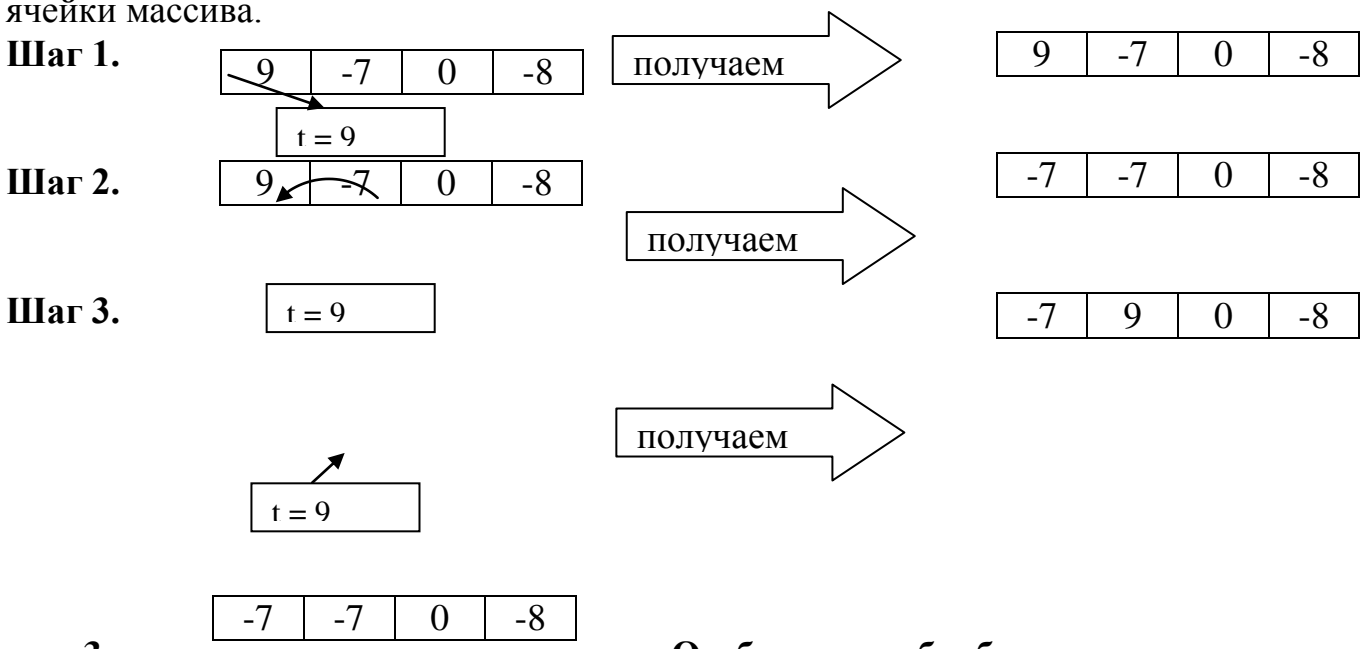
Таким образом, для организации сортировки потребуется два цикла, которые выстраиваются в следующем порядке:

(n-размерность массива)

```
For j:=1 to n do    {количество просмотров массива}
For i:=1 to n-1 do  {перебор элементов массива}
If mas[i] > mas[i+1] then begin
                            t:=mas[i];
                            mas[i]:=mas[i+1];
                            mas[i+1]:=t;
                          end;
```

} перестановка элементов массива

Как видно из примера для перестановки элементов массива требуется дополнительная переменная, которая служит временным хранилищем значения ячейки массива.



3. Особенности обработки двумерных массивов

Двумерный массив – структура данных, хранящая в себе прямоугольную матрицу. В матрице каждый элемент определяется номером строки и номером столбца, на пересечении которых он расположен.

Для описания двумерных массивов используются те же способы, что и для одномерных массивов, но в качестве размерности массива задается двойное значение (Например, ***[1..10,1..10]***).

Таким образом, для создания двумерного целочисленного массива размерностью 5×7 (5 строк, 7 столбцов) необходимо записать:

Способ 1

```
Type mas=array[1..5,1..7] of integer;
```

Способ 2

```
Var mas:array[1..5,1..7] of integer;
```

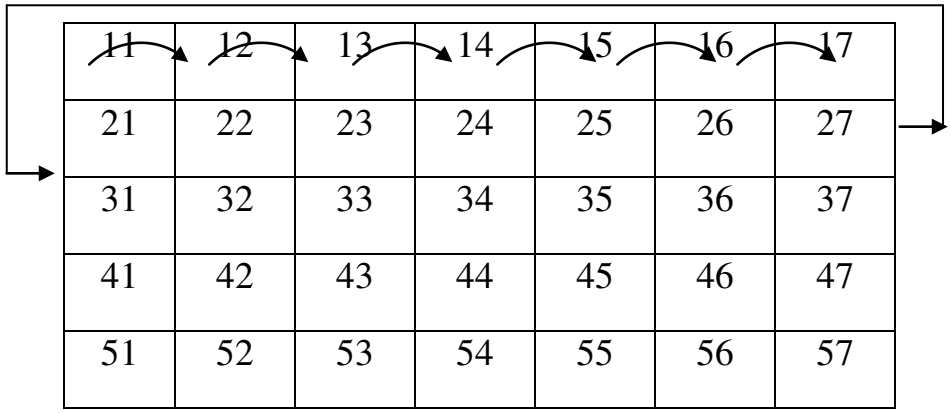
Для последовательного перебора всех элементов двумерного массива необходимо использовать т.н. вложенный цикл:

For i:=1 to 5 do {перебор строк матрицы}

For j:=1 to 7 do {перебор столбцов (ячеек) в строке}

Т.е. значение индекса строки (i) увеличится только в том случае, если индекс столбца (j) дойдет до своего конечного значения (в примере j = 7).

При такой организации перебора элементов массива процесс перебора будет проходить по следующей схеме:



11, 12 … - индексы ячеек

Рисунок 3. Процесс перебора элементов двумерного массива

Ход выполнения:

	i =	**j=**	**Условие перехода на следующую строку j=7**	**Обрабатываемый элемент**
Шаг 1	1	1	нет	Mas[1,1]
Шаг 2	1	2	нет	Mas[1,2]
Шаг 3	1	3	нет	Mas[1,3]
Шаг 4	1	4	нет	Mas[1,4]
Шаг 5	1	5	нет	Mas[1,5]
Шаг 6	1	6	нет	Mas[1,6]
Шаг 7	1	7	да	Mas[1,7]
Шаг 8	2	1	нет	Mas[2,1]
Шаг 9	2	2	нет	Mas[2,2]
Шаг 10	2	3	нет	Mas[2,3]
…	**…**	**…**	**…**	**…**

Для того чтобы вывести двумерный массив на экран в виде таблицы необходимо после вывода содержимого каждой строки предусмотреть переход на строку ниже:

```
For i:=1 to n do
Begin
```

```
    For j:=1 to n do
    Write(mas[i,j],' ');
    Writeln;
End;
```

4. Обработка квадратных матриц

Если количество строк массива равно количеству столбцов, то такой массив называется квадратной матрицей. При работе с квадратной матрицей, в отличие от обычного двумерного массива, можно выделить:

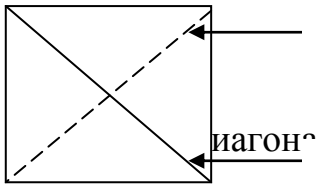
- диагонали (главная, побочная);
- элементы, расположенные над и под диагоналями;
- четверти матрицы.

11	12	13	14	15
21	22	23	24	25
31	32	33	34	35
41	42	43	44	45
51	52	53	54	55

Здесь первая цифра номера элемента обозначает номер строки матрицы (i), вторая цифра – номер столбца (j)

Для определения элементов, входящих в любой из перечисленных разделов, существует формула, основными составляющими которой являются i – номер строки, j – номер столбца и N – размерность массива. Например, для определения элемента с номером 43, расположенного под побочной диагональю можно использовать формулу **i+j>N+1**, где i=4, j=3, N=5, таким образом, получаем 4+3>5+1.

4.1 Определение диагоналей массива



Побочная диагональ, **i + j = N+1**

Главная диагональ, **i = j**

Таким образом, в матрице, представленной в п. 4, элементы с номерами 11, 22, 33, 44 и 55 являются элементами главной диагонали. Элементы с номерами 15, 24, 33, 42, 51 – элементы побочной диагонали.

Расположение элементов, находящихся над или под диагональю, определяется по отношению к одной из диагоналей.

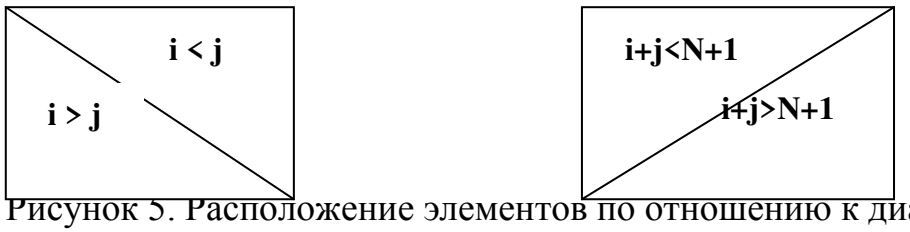


Рисунок 5. Расположение элементов по отношению к диагоналям

Элементы 12, 13, 14, 15, 23, 24, 25, 34, 35 и 45 расположены над главной диагональю, 21, 31, 32, 41, 42, 43, 51, 52, 53, 54 расположены под главной диагональю. Элементы 11, 12, 13, 14, 21, 22, 23, 31, 32 и 41 расположены над побочной диагональю, 25, 34, 35, 43, 44, 45, 54, 53, 54, 55 расположены под побочной диагональю.

4.2 Определение четвертей матрицы

Относительно обеих диагоналей элемент массива может находиться в одной из четвертей.

12, 13, 14, 23 – элементы первой четверти
25, 34, 35, 45 – элементы второй четверти
43, 52, 53, 54 – элементы третьей четверти
21, 31, 32, 41 – элементы четвертой четверти

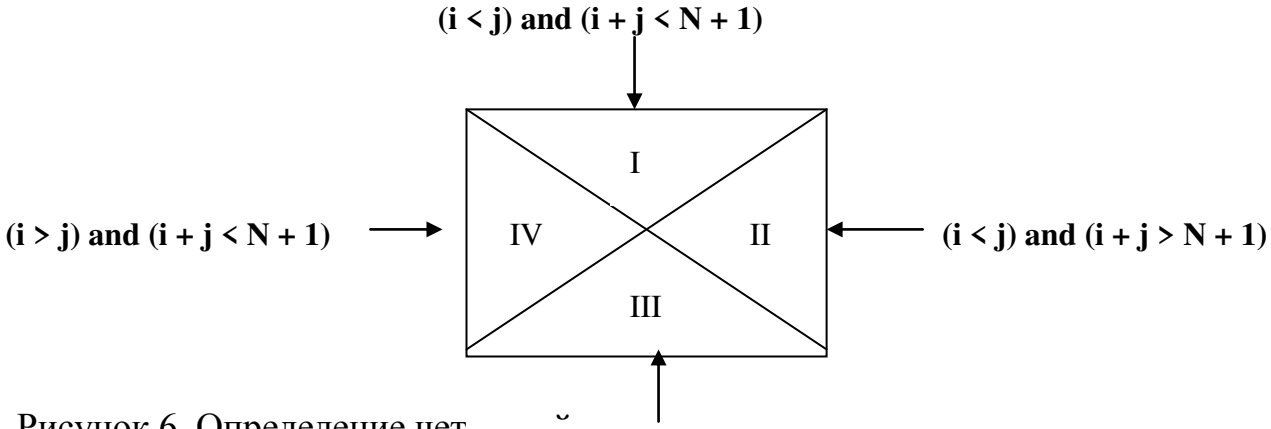


Рисунок 6. Определение чет

Используя правила, очень легко можно программным путем формировать матрицы требуемого вида.

Например, сформировать матрицу N × N вида:

4	0	0	0	5
1	4	0	5	2
1	1	4	2	2
1	5	3	4	2
5	3	3	3	4

на главной диагонали – 5;
на побочной диагонали – 4;
в I четверти – 0;
во II четверти – 2;
в III четверти – 3;
в IV четверти – 1.

```
Var
mas:array[1..100,1..100] of integer; i,j,n:integer;
Begin
Writeln('Введите размерность массива');
Readln(n);
For i:=1 to n do   {заполняем массив в соответствии с правилами}
For j:=1 to n do
Begin
   If i=j then mas[i,j]:=4;
   If i+j<N+1 then mas[i,j]:=5;
   If (i<j) and (i+j<N+1) then mas[i,j]:=0;
   If (i<j) and (i+j>N+1) then mas[i,j]:=2;
```

```
    If (i>j) and (i+j>N+1) then mas[i,j]:=3;
    If (i>j) and (i+j<N+1) then mas[i,j]:=4;
End;
For i:=1 to n do {выводим полученный массив на экран в виде таблицы}
   Begin
      For j:=1 to n do
      Write(mas[i,j],' ');
      Writeln;
    End;
End.
```

5. Открытые массивы

Открытые массивы используются в процедурах и функциях как параметры, у которых не задаются размеры. Фактический размер в этом случае определяется с помощью функций High. Индексация всегда начинается с нуля.

Например,

Вычислить максимальный элемент в массиве

```
   function Max (Var Mas: array of integer): integer;
   Var        Ma : integer;
              i : Byte;
Begin
       Ma : = Mas [0];
       for i : = 1 to High (Mas) do
            if Ma < Mas [i] then
              Ma : = Mas [i];
       Max : = Ma
   End.
```

Данная функция может работать с любым одномерным массивом целых чисел.

2.4. Изучение типов и принципов действия операторов цикла.

Командой повторения или *циклом* называется такая форма организации действий, при которой одна и та же последовательность действий повторяется до тех пор, пока сохраняется значение некоторого логического выражения. При изменении значения логического выражения на противоположное повторения прекращаются (цикл завершается). Для организации цикла необходимо выполнить следующие действия:

1. перед началом цикла задать начальное значение параметра;
2. внутри цикла изменять параметр цикла с помощью оператора присваивания;
3. проверять условие повторения или окончания цикла;
4. управлять циклом, т.е. переходить к его началу, если он не закончен, или выходить из цикла в противном случае.

Различают циклы с известным числом повторений (*цикл с параметром*) и итерационные (с *предусловием* и *постусловием*).

Оператор цикла с параметром

Данный оператор применяют тогда, когда известно число повторений одного и того же действия. Начальное и конечное значения параметра цикла могут быть представлены константами, переменными или арифметическим выражениями. Оператор имеет две формы:

1. **For** *параметр* := a **to** b **do** *тело цикла*;
2. **For** *параметр* := a **downto** b **do** *тело цикла*;

где a и b - величины целого типа, причем a - начальное значение, а b - конечное значение переменой - параметра.

Сначала вычисляются значения выражений a и b. Если a окажется меньшим или равным b, то *параметр* последовательно принимает значения, равные a, a+1, …, b-1, b, и для каждого из этих значений выполняется *тело цикла*. Если же a>b, то *тело цикла* не будет выполнено ни разу, и управление будет передано следующему оператору программы.

Оператор

For *параметр* := a downto b do *тело цикла*;

выполняется аналогичным образом, но *параметр* принимает значения, равные a, a-1, …, b+1, b. Если *тело цикла* состоит из нескольких операторов, то операторы тела цикла заключаются в операторные скобки begin - end. Например:

```
For x: =1 to 10 do
writeln (x);
For i: =10 to 100 do y: =y+5;
```

Оператор while

Данный оператор используется для программирования циклических процессов, в которых число повторений неизвестно, но задается некоторое условие его окончания.

Оператор имеет вид:

While *условие* **do** *тело цикла*;

Работа оператора начинается с проверки условия, записанного после слова while. Если это условие соблюдается, то выполняется *тело цикла*, а затем вновь проверяется условие и т. д. Как только на очереднбом шаге окажется, что условие не соблюдается, то выполнение *тела цикла* прекратится. Если *тело цикла* состоит из нескольких операторов, то они объединяются операторными скобками. В *теле цикла* обязательно должен быть оператор, влияющий на соблюдение условия, в противном случае произойдет зацикливание.

Оператор repeat

Для программной реализации циклических процессов с неизвестным числом повторений существует еще один оператор, который имеет вид:

Repeat
тело цикла
Until *условие*;

Данный оператор аналогично предыдущему оператору цикла, но отличается от него тем, что проверка условия проводится после очередного выполнения *тела цикла*. Это обеспечивает его выполнение хотя бы один раз. Сначала выполняется последовательность операторов, входивших в *тело цикла*, после чего проверяется выполнение условия, записанного за служебным словом until. Если условие не соблюдается, цикл завершается. В противном случае *тело цикла* повторяется еще раз, после чего проверяется соблюдение условия. Естественно, *тело цикла* должно содержать оператор, влияющий на условие окончания (продолжения), иначе цикл будет бесконечным.

В качестве примера рассмотрим задачу планирования закупки товаров в магазине на определенную сумму, не превышающую заданной величины. Обозначим через c и k соответственно цену и количество товара, через p - заданную предельную сумму, через s - общую стоимость покупки. Начальное значение общей стоимости покупки (s) равно нулю. Значение предельной суммы (p) должно вводиться с клавиатуры. Необходимо повторять запрос о цене и количестве выбранного товара, вычислять его стоимость, суммировать ее с общей стоимостью и выводить результат на экран до тех пор, пока s не превысит предельную сумму p. В таком случае на экран можно вывести сообщение о превышении предельного значения. Решение этой задачи можно записать в виде следующей программы:

```
Program pr2;
Var c, k, p, s: integer;
Begin
Writeln ('Предельная сумма - '); readln (p);
s: = 0;
Repeat
Writeln ('Введите цену товара и его количество');
Readln (c, k);
s: s+c*k;
Writeln ('Стоимость покупки равна', s);
Until s>p;
Writeln ('Общая стоимость покупки превысила предельную сумму');
Readln;
End.
```

Задание.

Методом итераций вычислить корень уравнения вида f(x)=0, расположенный в интервале [A, B], с абсолютной погрешностью в соответствии с вариантом задания. Определить также число итераций, необходимое для нахождения корня.

вариант задания	уравнение	отрезок	точность
1	$e^x - e\sim^x - 2 = 0$	[0;1]	1E-4
2	3sin Vx + 0,35 x - 3,8 = 0	[2;3]	1E-4
3	x - 2 + sin(1/ x) = 0	[12;2]	1E-4
4	1 - x + sinx - ln(1+x)=0	[0;1,5]	1E-4
5	x^2- ln(1+x)-3=0	[2;3]	1E-4
6	1/(x - 3+sin3,6x) = 0	[0;0,85]	0.5E-4
7	ln x - x + 1,8 = 0	[2;3]	0.5E-4
8	$0,1x^2$ -x ln x = 0	[1;2]	0.5E-4
9	x + cos($x^{0,52}$ + 2) = 0	[0,5;1]	1E-4
10	V1 - 0.4x - arcsin x = 0	[0;1]	1E-4

2.5. Проектирование вложенных циклов.

Вложенные циклы.

Циклы могут быть *простыми* или *вложенными* (цикл в цикле).

Например:

```
Program VCicl;
var i,j:integer;
begin
for i:=1 to 5 do
begin
writeln;
for j:=10 to 13 do
write('i=',i,' j=',j);
end;
readln;
end.
```

Для цикла for i:=1 to 5 do телом цикла является:

```
begin for j:=10 to 13 do
write(' i= ', i, ' , j = ', j);
writeln;
end;
```

Этот цикл является внешним, по отношению к нему внутренним будет цикл:

for j:=10 to 13 do с телом write (' i = ', i , j =', j);

Разберѐм работу программы, с *вложенным* циклом.

Сначала программа и начинает выполнять внешний цикл: присваивает i=l, затем переходит к его телу, а здесь встречает внутренний цикл и присваивает j значение 10, после чего выполняет тело внутреннего цикла, т.е. выводит на экран i=l, j=10. Так как внутренний цикл еще не окончен, то машина продолжает его выполнять, т.е. присваивает j значение 11 и добавляет к уже выведенной строке i=l, j=11.

Заметим, что оператор write отличается от оператора writeln тем, что он не начинает вывод с новой строки, а продолжает писать в той же строке, т.е. после второго выполнения внутреннего цикла на экране появится

i= 1, j=10 i= 1, j=11.

Программа продолжит выполнение внутреннего цикла, и, когда он закончится

(выполнится для j = 10,11,12,13), на экране будет строка

i = 1 j = 10 i =l j = 11 i = 1 j = 12 i = 1 j = 13.

Внутренний цикл закончится, однако тело внешнего цикла еще не закончи-лось, поэтому

выполняется оператор writeln, который переводит курсор на новую строку. После этого тело внешнего цикла закончится, но сам цикл отработал только для i = 1. Поэтому внешний цикл продолжит работу, присвоив i: =2 и вновь начав выполнение своего тела. Встретив внутренний цикл j, на экран с новой строки выведется: i=2, j=10, затем к этой строке добавится i=2, j=11 и т.д., пока не закончится внутренний цикл.

Таким образом, внешний цикл, изменяя индекс i от 1 до 5, заставит каждый раз

выполняться полностью внутренний цикл, и в результате работы программы на экране

появится:

i=l, j=10 i=1, j=11 i=1, j=12 i=1, j=13
i=2, j=10 i=1, j=11 i=1, j=12 i=1, j=13
i=3, j=10 i=1, j=11 i=1, j=12 i=1, j=13
i=4, j=10 i=1, j=11 i=1, j=12 i=1, j=13
i=5, j=10 i=1, j=11 i=1, j=12 i=1, j=13

Вкладывать циклы друг в друга можно сколько угодно раз, необходимо лишь помнить,

что количество выполнений самого внутреннего тела цикла при этом будет расти в

геометрической прогрессии. Например:

```
for i:=l to 9 do
for j:=l to 9 do
for k:=l to 9 do
writeln (i, j, k);
```

дает столбик цифр:

111 112 113 114 … 119
– – – – –
121 122 123 124 … 129
– – – – –
211 212 213 214 … 219
– – – – –
991 992 993 994 … 999

что составляет 1000 строчек.

Вопросы:

• Какие виды циклов вы знаете?
• Что называют вложенным циклом?
• Сколько может быть уровней вложения циклов?
• В чѐм отличие простых циклов от вложенных?

III. Практическая часть.

На прошлом уроке мы рассмотрели на практике применение циклов в ЯП Pascal ABC.

Сегодня мы рассмотрим на практике применение вложенных циклов в ЯП Pascal ABC.

1. Программа наглядно показывающая работу вложенных циклов:

```
Program VlozCicly;
var x,y:integer;
begin
for x:=1 to 2 do
for y:=1 to 4 do
begin
writeln('x=',x,' y=',y);
end;
readln;
end.
```

2. Программа вычисляющая сумму всех чисел последовательности от 0 до 100, с помощью вложенных циклов:

```
Program SummDvuznach; // Сумма всех чисел последовательности от 0 до 100
var s: real; i,j:integer;
begin
s := 0;
for i:=1 to 99 do
s := s + i;
for j:=1 to 99 do
write(j:3);
writeln;
writeln(' Сумма всех чисел от 0 до 100 = ',s);
end.
```

Задание.

Вычислить значение суммы членов бесконечного ряда с заданной точностью Eps. На печать вывести значение суммы и число членов ряда, вошедших в сумму.

Вариант	Сумма членов ряда	Значение	Точность вычисления
1	$S=-\frac{2x^2}{2}+\frac{((2x)^2)^2}{24}+....+(-1)^n\frac{(2x)^{2n}}{(2n)!}$	0.20	10^{-5}
2	$S=x-\frac{x^3}{3}+.....(-1)^n\frac{x^{2n-1}}{2n-1}$	0.10	$0.5*10^{-4}$
3	$S=\frac{x^3}{5}-\frac{x^5}{17}+.....+(-1)^{n+1}\frac{x^{2n+1}}{4n^2+1}$	0.15	10^{-5}
4	$S=1+Cos(\pi/4)\frac{x}{1!}+.....+\frac{Cos(n*\pi/4)}{n!}x^n$	0.12	10^{-4}
5	$S=1+\frac{x^2}{2!}+.....+\frac{x^{2n}}{(2n)!}$	0.70	10^{-4}
6	$S=4\cdot(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-....+(-1)^n\frac{1}{2n-1}$	-	10^{-4}
7	$S=\frac{1}{x}-\frac{1}{3x^3}+\frac{1}{5x^5}+....+(-1)^n\frac{1}{(2n+1)x^{2n+1}}$	1.5	$0.5*10^{-5}$
8	$S=x+\frac{x^3}{3!}+\frac{x^5}{5!}+....+\frac{x^{2n+1}}{(2n+1)!}$	1.7	10^{-5}
9	$S=1+\frac{x^2}{2!}-\frac{3x^4}{4!}+.....+(-1)^n\frac{2n-1}{(2n)!}x^{2n}$	0.75	$0.5*10^{-5}$
10	$S=1+\frac{Cosx}{1!}+\frac{Cos2x}{2!}+....+\frac{Cosnx}{n!}$	0.3	10^{-4}

2.6. Организация и обработка записей.

Описание записей в Паскале

При использовании записей они д.б. описаны с использованием раздела описания либо непосредственно в разделе описания переменных

Запись имеет следующую структуру:

```
IZ = RECORD
     IK1 : T1;
     IK2 : T2;
      . . .
     IKN :TN;
  END;
```

Где: IZ – имя типа записи**;**
IK – имя поля (компоненты);
T - тип компоненты, м.б. любой стандартный либо предварительно определенный (кроме файлового);
RECORD – запись;
RECORD-END – операторные скобки.

Пример описания записи

```
    Const  n=25;
 Type   Date = Record
                  Year      : integer;   {Год}
                  Month   : 1..12;       {Число}
```

```
                Day       : 1..31;       {Месяц}
              End;
        Student = Record
                FIO       : String[15]; {ФИО}
                Dr        : Date;       {Дата рождения}
                Str       : String[20]; {Страна}
          b1,b2,b3,b4,b5  : 2..5;       {Поле оценок}
              End;
      Vec = array[1..n] of Student;

Var   EMS_43a, EMS_43a, EMS_43a   :Vec;
```

Порядок следования полей в записи не имеет значения.

Поле может иметь несколько имен, при этом они разделяются запятыми. Для наглядности поля следует располагать на отдельных строках и снабжать комментариями.

Запись может содержать вариантную часть. Например, для иностранных студентов можно указать страну.

```
   Type x=(Yes, No);
     Student = Record
                FIO       : String[25]; {ФИО}
                Mark      : 2..5;       {Оценки}
                Variant   :x;
                 Case x of
                     Yes  : (Str  : String[20]); {Страна}
                      No  :  (    );
            End;
```

Обработка записей в Паскале

При обработке записей все действия выполняются над полями записи в соответствии с их типами. Набор операций, процедур и функций определяется типом поля. Для записи в целом м.б. использован только оператор присваивания. Запись можно передавать в качестве параметров в процедуру и функцию.

Для обращения к полям записи используются составные имена.

Составное имя – совокупность имени переменной типа запись и имен полей разделенных точкой.

Например:

```
EMS_43a[1].FIO :='Антонов';
EMS_43a[1].Dr.Day :=13;
EMS_43a[1].Dr. Month :=1;
EMS_43a[1].Dr. Year  :=1986;
```

Оператор присоединения в Паскале

Оператор присоединения позволяет упростить обращение к элементам записи и имеет следующий формат записи:

With <Имя> do

OP;

Где **With** - с;

<Имя> - имя записи (простое или составное);

do - делать;

OP - оператор (простой или составной);

Вввод / вывод записей в Паскале

При вводе/выводе записей используются отдельные поля записи

For i:=1 to n do Read (EMS_43a[i].FIO, EMS_43a[i].Str)

При использовании оператора присоединения

```
For i:=1 to n  do
With EMS_43a[i] do
  Begin
     Read (FIO);
     Read (Str);
   End;
```

Примеры программ

Пример 18.1. В файле **sp_gr.txt** содержатся результаты сдачи сессии студентами группы КИТ-23а. Составить программу анализа результатов сдачи сессии.

Вычислить средний бал каждого студента, результат вывести на экран и в файл**rez.txt.**На экран вывести также средний балл каждого студента по первому предмету в списке и общий средний балл группы.

Исходный файл **sp_gr.txt**

```
Антонов И.И.   5 5 4 3 4
Андеев И.Т.    4 5 3 4 2
Иванов Н.Р.    5 5 4 2 4
Петров П.И.    3 4 3 2 5
Агапов Е.И.    5 4 3 5 2
Токаев П.И.    4 5 2 4 4
Павлов О.О.    3 3 3 4 5
Мщисеев Н.Т.   5 4 3 4 5
Сидоров Н.Р.   5 4 3 2 5
Егоров Н.Р.    5 3 4 3 5
```

Текст программы **(srbal.pas)**

```
Program srbal;
{Программа анализа успеваемости студентов группы}
Uses crt;
Const n=10;            {Число студентов}
Type st=Record
fam:string[15];
b1,b2,b3,b4,b5:2..5;
End;
vec = array[1..n] of st;
Var
KIT_23b         :vec;
sb1,sbo,sb      :real;
i               :byte;
f1,f2           :text;
Begin
Clrscr;
Assign(f1,'sp_gr.txt');  Reset(f1);
Assign(f2,'rez.txt');    Rewrite(f2);
sbo:=0;  sb1:=0;
For i:=1 to n do
Begin
sb:=0;
With KIT_23b[i] do
Begin
Read(f1,fam);  Write(fam); Write(f2,fam);
Readln(f1,b1,b2,b3,b4,b5);
sb:=sb+(b1+b2+b3+b4+b5)/5;
Writeln(sb:4:2);  Writeln(f2,sb:4:2);
sb1:=sb1+b1/n;
End;
sbo:=sbo+sb/n;
End;
Writeln;
Writeln('sbo=',sbo:4:2,' ':10,'sb1=',sb1:4:2);
Close(f1);  Close(f2);
End.
```

Результаты работы программы:

(экран) **(файл rez.txt)**

```
Антонов И.И.   4.20
Андеев И.Т.    3.60
Иванов Н.Р.    4.00
Петров П.И.    3.40
```

```
Агапов Е.И.     3.80
Токаев П.И.     3.80
Павлов О.О.     3.60
Мщисеев Н.Т.  4.20
Сидоров Н.Р.   3.80
Егоров Н.Р.     4.00
      Антонов И.И.  4.20
Андеев И.Т.     3.60
Иванов Н.Р.     4.00
Петров П.И.    3.40
Агапов Е.И.     3.80
Токаев П.И.     3.80
Павлов О.О.    3.60
Мщисеев Н.Т. 4.20
Сидоров Н.Р.   3.80
Егоров Н.Р.     4.00
    sbo=3.84         sb1=4.40
```

Пример 18.2. Составить список студентов группы КИТ-23а имеющих право на получение стипендии (средний бал не ниже 4.0). Средний бал каждого студента по результатам сессии приведен в файле **rez.txt** (см. пример №1).

Текст программы

```
Program STIP;
{Программа анализа упеваемости}
uses CRT;
const n=10;
Type st=Record
FIO: string[15];
SB:  real;
end;
vec = array [1..n] of st;
Var sp,sp1    :vec;
i,k,kst   :integer;
f        :text;
Procedure VVod (var st:vec);
{Ввод сиска студентов}
Begin
For i:= 1 to n do readln (f,st[i].FIO,st[i].SB);
End;
Procedure Out(st:vec;var st1:vec; var ks:integer);
{ Список студентов на стипендию  с баллом выше 4,5 }
Begin
```

```
k:=1;
for i:=1 to n do if st[i].sb >4.5 then
begin
st1[k]:=st[i];  ks:=k; k:=k+1;
end;
end;
Begin
assign (f,'inp2.txt'); reset (f);
clrscr;
Writeln('Исходный массив записей');
Writeln;
VVod (sp);  Out(sp,sp1,kst);
For i:=1 to n do
Writeln (sp[i].FIO,' ', sp[i].SB:4:2);
Writeln ('Результат:':30);
Writeln('Список студентов на стипендию с баллом выше 4,5 ');
For k:=1 to kst do Writeln (sp1[k].FIO,' ',sp1[k].SB:4:2);
Writeln ('Количество студентов с баллом выше 4,5 :',kst);
Writeln;
Readln;
End.
```

Результаты работы программы:

Исходный массив записей
Антонов И.И. 4.20
Андеев И.Т. 3.60
Иванов Н.Р. 4.00
Петров П.И. 3.40
Агапов Е.И. 3.80
Токаев П.И. 3.80
Павлов О.О. 3.60
Мщисеев Н.Т. 4.20
Сидоров Н.Р. 3.80
Егоров Н.Р. 4.00
Результат:
Список студентов на стипендию с баллом не ниже 4
Антонов И.И. 4.20
Иванов Н.Р. 4.00
Мщисеев Н.Т. 4.20
Егоров Н.Р. 4.00
Количество студентов с баллом не ниже 4 равно :4

Пример 18.3. Составить список студентов группы КИТ-23а имеющих право на голосование (с датой рождения до 30.10.2004 г.).

Исходный файл **sp_gr.txt**

```
   Антонов И.И.   05 12 1986
Андеев И.Т.     14 05 1985
Иванов Н.Р.     15 12 1985
Петров П.И.     31 10 1986
Агапов Е.И.     11 04 1985
Токаев П.И.     14 05 1986
Павлов О.О.     30 10 1986
Мщисеев Н.Т.   22 09 1985
Сидоров Н.Р.    29 12 1985
Егоров Н.Р.     17 10 1985
```

Текст программы

```
Program GOLOS;
{Сотавление списка для голосования. Выборы 31.10.2004 г.}
uses CRT;
const n=10;
Type
Date=Record
Day   :1..31;
Month :1..12;
Year  :1980..1987;
end;
st=Record
FIO: string[15];
DR : Date;
end;
vec = array [1..n] of st;
Var sp,sp1    :vec;
i,kvo,k   :byte;
f         :text;
Procedure VVod (var st:vec);
{Ввод списка студентов из файла и вывод на экран}
Begin
For i:= 1 to n do
With st[i] do
begin
readln (f,FIO,DR.Day,DR.Month,DR.Year);
Writeln(FIO,DR.Day:4,DR.Month:4,DR.Year:6);
end;
```

```
End;
Procedure Out (st:vec;var st1:vec; Var kn:byte);
{ Формирование списка студентов для голосования}
Begin
k:=1;
for i:=1 to n do if
(st[i].DR.Year<=1986)and
(st[i].DR.Month<=10)and
(st[i].DR.Day<=30) then
begin
st1[k]:=st[i]; kn:=k; k:=k+1;
End;
End;
Begin
assign (f,'sp_gr1.txt'); reset (f);
clrscr;
Writeln('Исходный массив записей:':30);
Writeln;
VVod (sp); Out(sp,sp1,kvo);
Writeln ('Результат:':30);
Writeln ('Спиок студентов для голосования:');
For k:= 1 to kvo do
With sp1[k] do Writeln(FIO,DR.Day:4,DR.Month:4,DR.Year:6);
Writeln ('Число студентов в списке=> ',kvo);
Readln;
End.
```

Результаты работы программы:

```
Исходный массив записей:
Антонов И.И.      5  12  1986
Андеев И.Т.      14   5  1985
Иванов Н.Р.      15  12  1985
Петров П.И.      31  10  1986
Агапов Е.И.      11   4  1985
Токаев П.И.      14   5  1986
Павлов О.О.      30  10  1986
Мщисеев Н.Т.     22   9  1985
Сидоров Н.Р.     29  12  1985
Егоров Н.Р.      17  10  1985
Результат:
Спиок студентов для голосования:
Андеев И.Т.      14   5  1985
Агапов Е.И.      11   4  1985
```

Токаев П.И. 14 5 1986
Павлов О.О. 30 10 1986
Мщисеев Н.Т. 22 9 1985
Егоров Н.Р. 17 10 1985
Число студентов в списке=> 6

2.7. Разработка процедур и функций.

Основные понятия.

Широко распространена в программах форма повторяемости, когда одна и та же последовательность действий должна выполняться на различных этапах информации. В программах такого рода в различных местах встречаются фрагменты, одинаковые по выполняемым действиям и различающиеся только в значениях исходных данных. При составлении программы приходиться задавать одну и ту же группу операторов, соответствующую каждому из повторяющихся фрагментов. Для более эффективного программирования подобных повторений в языке введено понятие подпрограммы. Повторяющаяся группа операторов оформляется в виде самостоятельной единицы – подпрограммы, записывается однократно, а в соответствующих местах программы обеспечивает лишь обращение к ней. Использование аппарата подпрограммы позволяет сократить объем и улучшить структуру программы с точки зрения наглядности и читаемости. Подпрограмма может быть рассмотрена как самостоятельная программа (со своими входными и выходными данными).

В языке Паскаль подпрограммы реализуются в виде процедур и функций, которые вводятся в программу с помощью своего описания.

Описание процедуры.

Процедуры описываются в специальном разделе описательной части программы вслед за разделом переменных.

Любая процедура состоит, аналогично программе, из заголовка процедуры и блока. Заголовок процедуры представляет собой:

PROCEDURE <ИМЯ> (<СПИСОК ПАРАМЕТРОВ>),

где PROCEDURE – служебное слово, ИМЯ – имя процедуры, СПИСОК ПАРАМЕТРОВ - перечень имен для обозначения исходных данных и результатов работы процедуры с указанием их типов. Параметры, перечисленные в списке, называются формальными. Допускается описание процедуры, несодержащей формальных параметров:

PROCEDURE <ИМЯ>;

Содержательная часть процедуры представляет собой блок и состоит, следовательно, из раздела описаний (меток, констант, типов, переменных, процедур, и функций) и раздела операторов, представляющего собой составной

Если в заголовке процедуры параметры указаны без слова VAR, то это параметры-значения. Параметры-значения могут изменяться внутри процедуры, но для внешней программы это изменение окажется незамеченным.

Для получения результатов в основной программе используются параметры-переменные. Эти параметры перечисляются после служебного слова VAR с обязательным указанием типа.

Тело процедуры состоит:

1) из описательной части, где определена переменная I, необходимая и имеющая смысл только внутри данной процедуры и называемая локальной переменной (значение локальной переменной недоступно в основной программе);
2) из составного оператора BEGIN-END, реализующего алгоритм вычисления степени действительного числа с натуральным показателем.

Функция.

Функция – это подпрограмма, результат выполнения которой есть единственное скалярное значение, присваиваемое имени этой функции. Следовательно, функции являются частным случаем процедур и принципиально отличаются от них тем, что, во-первых, результат выполнения функции – одно значение, а процедуры – одно или несколько; во-вторых, результат выполнения функции передается в основную программу, как значение имени этой функции, а результаты выполнения процедуры – как значения ее параметров.

Описание функции аналогично описанию процедуры и состоит из заголовка и блока. Заголовок функции имеет вид:

FUNCTION <ИМЯ> (<СПИСОК ПАРАМЕТРОВ>):<ТИП>

где FUNCTION – служебное слово, ИМЯ – имя функции, СПИСОК ПАРАМЕТРОВ – перечень формальных параметров (исходных данных) с указанием их типов, ТИП – тип результата: значение, которое должно приобретать имя функции.

Допускается описание функции без параметров:

FUNCTION <ИМЯ>: <ТИП>;

В содержательной части программы-функции имени должно быть присвоено некоторое значение (значение ответа), т.е. имя хотя бы один раз должно присутствовать в левой части некоторого оператора присваивания.

Обращение к подпрограммам.

Описание процедуры (или функции), расположенное в разделе описаний, само по себе никакого действия не вызывает. Чтобы исполнить процедуру (или функцию), необходимо в нужном месте программы поместить обращение к ней. Обращение к процедуре производится с помощью оператора процедуры, имеющего вид:

<ИМЯ> (<СПИСОК АРГУМЕНТОВ>);

где ИМЯ – имя процедуры, к которой происходит обращение, СПИСОК АРГУМЕНТОВ – перечень конкретных значений (выражений) и имен, подставляемых на место формальных параметров процедуры при ее выполнении.

При вызове процедуры формальные параметры, указанные в заголовке, заменяются аргументами в порядке их следования: первому слева параметру в списке ставится в соответствие первый аргумент, второму – второй и т.д. Аргументы, перечисленные в операторе процедуры, называются также фактическими параметрами. Число, тип и порядок следования формальных и фактических параметров должно совпадать. Структура программы, содержащей процедуру, имеет вид:

```
Program <имя>;
Procedure< имя >;
{Секция описания подпрограммы}
Begin
   < оператор 1 >
   < оператор 2 >
     ……
   < оператор N >
End;
Begin
   < оператор 1 >
   < оператор 2 >
     ……
   < оператор N >
End.
```

Секция описания программы

Основная программа

Формальные параметры – это переменные, фиктивно (формально) присутствующие в процедуре и определяющие тип и место подстановки фактических параметров.

Фактические параметры – это реальные объекты (программы, заменяющие в теле процедуры при ее вызове формальные параметры). Над этими объектами и производятся действия, предусмотренные операторами тела процедуры.

Имена формальных и фактических параметров целесообразно выбирать различными, что сделает программу более наглядной.

Обращение к функции осуществляется аналогично обращению к стандартным функциям (sin, cos, tan и т.д.) и является разновидностью операнда в выражениях в отличие от вызова процедуры, являющегося разновидностью оператора. В этом месте выражения, где это необходимо, записывается имя функции, вслед за которым в скобках перечисляются фактические параметры. Если вызывается функция без параметров, то указывается только ее имя.

Задание

1. Из таблицы 1. По номеру варианта взять задание и составить программу, используя подпрограмму- процедуру.

2. Из таблицы 2 по номеру варианта взять задание и составить программу, используя подпрограмму FUNCTION.

Таблица 1

№	Содержание задания
1.	Определить математическое ожидание и дисперсию для четырёх случайных чисел, заданных векторами: A=[0,5; 1,5; 2] b=[6,7; 8; 7,5; 6] C=[0,1; 10; 4] d=[3,2; 5,1] Расчёт математического ожидания и дисперсии производится по следующим формулам: $M=\frac{1}{n}\sum_{i=1}^{n}x_i \qquad D=\frac{1}{n}\sum_{i=1}^{n}x_{i^2}-M^2$
2.	Определить расстояние от начала координат до точки P, делящей отрезок с координатами P={2;6} M={10;8} в отношении L=3/2 . Расстояние определяется по формуле: $R=\sqrt{x^2+y^2}$, где $x=(x_1+L\cdot x_2)/(1+L) \quad y=(y_1+L\cdot y_2)/(1+L)$
3.	Определить расстояние между точками A и B с координатами A=[2;5], B=[2;1], и точками C и D с координатами C=[20;4], D=[12;8]. Расстояние определяется по формуле: $f=\sqrt{p_1{}^2+p_2{}^2-2\cdot p_1\cdot p_2\cdot\cos(\varphi_2-\varphi_1)}$, где $p=\sqrt{x^2+y^2}$ и $\varphi=arctg\left(\frac{y}{x}\right)$
4.	Вычислить значение функции $v=e^{x_1+y_1}-e^{x_2+y_2}$, где x_1,x_2 корни уравнения $A\cdot x^2+B\cdot x-1{,}5=0$; y_1,y_2 - корни уравнения $2\cdot y^2-y+C=0$. Корни уравнения находятся в подпрограмме-процедуре. Если корни мнимые, то считать их равными нулю. Исходные данные: A=0,5; B=3; C=1;
5.	Заданы стороны двух треугольников ABC (стороны a, в, c) и DEF (стороны d, e, f). Найти сумму и разность площадей треугольников ABC DEF. Площадь треугольника NKM со сторонами n, k, m вычисляется по формуле: $W=\sqrt{r\cdot(r-n)\cdot(r-k)(r-m)}$ r - полупериметр треугольника NKM. Исходные данные: a=3; b=2,5; c=1,7; d=2; e=7,8; f=7
6.	Три точки заданы своими декартовыми координатами a={1;2}, b={1,2;1}, c={-3; -4}. Вычислить полярные координаты этих точек. Полярный радиус r и полярный угол φ вычисляются по

	формулам: $r = \sqrt{x^2 + y^2}$, $\varphi = arctg\left(\frac{y}{x}\right)$
7.	Определить номера точек, лежащих в круге радиусом r. Координаты точек заданы массивами X_i и Y_i. Исходные данные: n=6; r=3; i=1,2….n X_i={-1; 2,3; 3,3; -1,8; -2,4} Y_i={2;-2,8;6;1;-2; 1}
8.	Вычислить значение $L = \sum_{l=1}^{8} \frac{\sin^2(x_i)}{2}$, где x_i заданы массивом; X_i=(0,2; 0,46; 0,33; 0,97; 0,15; 0,61; 0,54; 0,77)
9.	Вычислить значение R=$\frac{X^h}{S}$, где X=$\sum_{i=1}^{n} Ti$, S=$\sum_{i=1+n}^{c} Ti$ – элементы массива. Исходные данные:n=3; m=8; i=1,.2..m; =(5,6; 0,3; -0,9; 3; 2,8; 1,45; -4,6; 1)
10.	Определить значение функции Z=sh(x+y) и M=sh(xy), где M изменятся от 1 до 0,5 с шагом 0,1; Y изменяется от 2 до 2,6 с шагом 0,2. Гиперболический синус вычисляется по формуле: $SH(n) = \frac{e^n}{2}$
11.	Заданы стороны двух треугольников JKL (стороны j,c,l), и ABD (стороны a,b,d).Переменной S присвоить значение –1, если площадь треугольника JCL меньше или равна площади треугольника ABD, и значение 1,если площадь треугольника JCL больше площади треугольника ABD. Площадь треугольника MNK со сторонами m,n,k вычисляются по формуле Герона. W=$\sqrt{r \cdot (r-m) \cdot (r-n) \cdot (r-k)}$, где r-полупериметр треугольника MNK. Исходные данные j=1; c=2,5; i=2,7; a=1; b=2,7; d=3,2.
12.	Построить таблицу Z=ch(x^2=y^2),где имеется от 3до 4 с шагом 0,1, у меняется от 2 до 3 с шагом 0,2. Гиперболический косинус вычисляется по формуле: CH(n)=(e^n+e^n)/2.
13.	Заданы два квадратных уравнения Ax^2+Dx+C=0, Dx^2+Fx+R=0. Найти минимальное значение среди корней этих уравнений. В случае если корни мнимые считать их равными нулю. Решение квадратного уравнения оформить в виде подпрограммы-процедуры.

	Исходные данные:a=2; b=-5,2; c=1,3; d=3,7; f=1,8; r=6.
14.	Четыре точки заданы своими координатами X=(x1,x2), Y=(y1,y2) Z=(z1,z2), P=(p1,p2) Вычислить и напечатать, сколько из них принадлежит полосе, аналитически заданной неравенством: f<Ma1+Na2<r. Проверку на принадлежность точки полосе оформить в виде подрограммы-процедуры. Исходные данные M=5; N=3; f=2,5; r=7,1; X=(-4,2;3); Y=(1,8; 0,8); Z=(-8,6; -4,1); P=(-1; -0,1).
15.	Задана окружность $(x-a)^2+(y-b)^2=r$ и две точки P=(p1,p2) и F=(f1,f2).Выяснить и напечатать, сколько точек (нуль,одна или две) лежит внутри окружности. Проверку, лежит ли точка внутри окружности, оформить в виде подпрограммы-процедуры. Исходные данные a=3,2; b=4,1; r=2; p=(6,1; 4,3); f=(27,48;-6) .

Таблица 2

№	Содержание задания
1.	$f=\frac{A\cdot x^2+B\cdot x-C}{\sin x+\cos x}$ при a=4,5; b=0,7; c=6,2; A x принимает значения 0,2; 0,56; 0,83
2.	$d=\cos(a-b)+\frac{\sin(a+b)}{\sqrt{a\cdot b\cdot c}}$, при a=0,8; b=0,16; c=0,4; a=0,6; b=0,4; c=1,2; a=0,47; b=0,1; c=0,5.
3	$n=\frac{\sqrt{a^2+b^2}}{\ln(a+b)}$, при a=0,15; b=1,5; a=1,7; b=0,1.
4.	$k=\frac{e^{d\cdot x+y}-e^{d\cdot x-y}}{x^2+y^2}$, при x=1,4; y=0,8; x=0,9; y=0,6; x=2,9; y=0,4; a d=5,3, при всех значениях x,y.
5.	$z=arctg(x+a)-\frac{\cos(y-b)}{a\cdot x+b\cdot y}$, при x=0,4; y=1,2; x=0,25; y=1,3; a=0,54; b=1 при всех значениях x, y
6.	$i=a\cdot\sin(x+y)-\frac{b\cdot\cos(x-y)}{\sqrt{4\cdot x\cdot y}}$, при a=10,7; b=6,3; y=0,35; a x принимает значения 0,6; 0,51; 0,42.

7.	$c = \frac{a \cdot e^{x^2} + b \cdot e^{y^2}}{a \cdot x + b \cdot y}$, при x=3; y=4; x=1,6; y=5,8; x=4,5; y=2,7; a=7,1; b=2,4 при всех значениях x, y
8.	$t = \frac{tg(x \cdot a) - b^3}{\cos^2(y \cdot b) + a}$, при x=0,1; y=0,7; x=0,4; y=0,6 x=0,5; y=0,2; a=2; b=0,1 при всех значениях x, y
9.	$m = \frac{e^{x+a \cdot b} + e^{a \cdot x}}{\sqrt[3]{a \cdot b}}$, при a=9,7; b=2,7, а x принимает значения 4,8; 9,6; 0,44.

10.	$w = \frac{\sin^3(x-y) + \cos^3(x+y)}{e^{(a+b) \cdot x}}$, при x=0,35; y=0,1;
	x=0,82; y=0,12;
	x=0,67; y=0,3,
	a=0,24;
	b=4,9 при всех значениях x, y.

11.	$y = \frac{e^{(a^2-b^2) \cdot x} + \ln(a \cdot x)}{\sqrt{x + a \cdot b}}$, при a=4; b=2,7, а x принимает значения 0,1; 0,25; 0,14; 0,21.

12.	$f = \ln(a + 8 \cdot x) - \ln^2(b + 6 \cdot y) / \sqrt{a \cdot b + x \cdot y}$
	при x = 0,1; y = 2,7;
	x = 0,4; y = 6,2;
	x = 1,6; y = 1,8; a = 0,2;

13.	$d = (e^{x-a} + e^{y+b}) / \sqrt{a \cdot x^2 + b \cdot y^2}$
	при x = 4,8; y = 2,1;
	x = 5,2; y = 4,7;
	x = 9,6; y = 3,2; a = 2,5; b = 4,1;
	при всех значениях x, y.

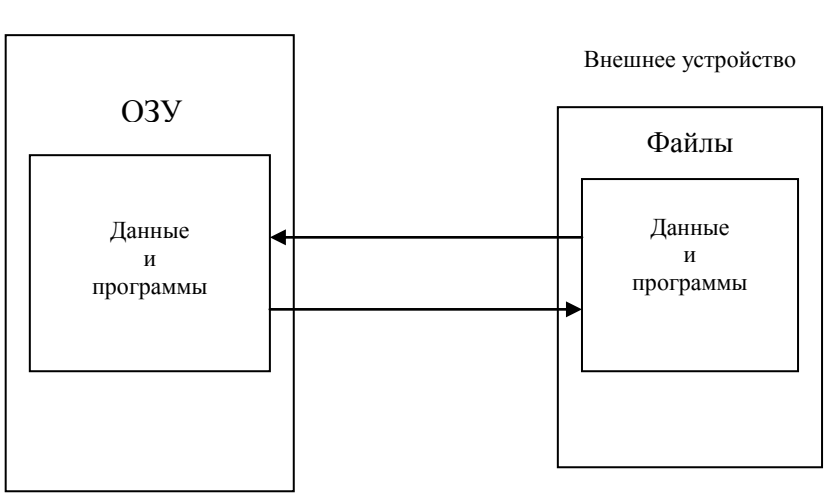
14.	$n = \left(0{,}7e^{x-5} + \cos(y)\right) / \sqrt{a \cdot x \cdot y}$ при x = 2,4; y = 1,5; x = 0,8; y = 1,2; a = 2,8 при всех x = 0,5; y = 2,9; значениях x, y.

15.	$g = (\sin^2(a \cdot x) + \cos^2(b \cdot y)) / \sqrt{a \cdot x + b \cdot y}$ при $x = 0{,}2;\quad y = 0{,}4;$ $x = 0{,}5;\quad y = 0{,}6;$ $x = 0{,}14;\quad y = 0{,}18;\quad a = 1{,}17;$ $b = 1{,}6$ при всех значениях x, y

2.8. Описание и обработка файлов.

Файл *– последовательность однотипных компонент (элементов) на внешнем носителе, имеющая имя.*

Обмен данными между памятью и внешним устройством осуществляется при помощи файлов.



С файлами связано два понятия:

- физический файл
- логический файл.

Физический файл находится на внешнем устройстве и имеет имя.

Если файл находится на магнитном диске, то имя задается полной спецификацией файла:

{логическое имя МД**:**}{\путь к файлу по каталогам\}имя файла{**.**расширение}

Например: **'W:\info\paskal\p1.pas'**

Если файл находится на другом внешнем устройстве, то имя файла задается логическим именем устройства:

‘CON’ – консоль (при вводе – клавиатура, при выводе – экран монитора)

‘LPn’, где n – 1,2, и т.д. – параллельный порт ввода-вывода

‘COMn’, где n – 1,2, и т.д. – последовательный порт ввода-вывода

‘LPT’ – принтер
‘NUL’ – фиктивное устройство.

Логический файл – это файловая переменная, связанная с типом файла при его описании. Файловую переменную можно задать любым идентификатором.

При работе с файлом сначала устанавливается соответствие между физическим и логическим именем файла (при помощи процедуры Assign), далее в программе используется только файловая переменная.

Например:

```
Var f:text;   {текстовый файл описывается через файловую переменную f }
        Begin assign(f, ‘w:\info\kl10\a.txt’);   {устанавливается соответствие между
          файловой переменной f                  и физическим файлом с именем
                                                 ‘w:\info\kl10\a.txt’}
        Reset(f);          {открытие файла для чтения}
        { обработка файла}
        Close(f);          {закрытие файла}
 End.
```

Существуют стандартные файловые переменные: INPUT – для ввода с клавиатуры, OUTPUT – для вывода на экран монитора. Они могут быть переопределены пользователем, т.е. могут быть связаны с другими устройствами ввода-вывода.

В Паскале существует три типа файлов:

- текстовый
- типизированный
- бестиповый.

Текстовые файлы.

Текстовый файл *– это стандартный тип файла, у которого два типа компонент (элементов) – символ (char) и строка (string).*

Описание текстового файла:

```
Var f:text;
    Ch:char;
     S:string;
```

где f – файловая переменная, ch, s – компоненты файла. Идентификаторы могут быть любые (по правилам написания идентификаторов).

Структура текстового файла - линейная:

					#13	#10						#13	#10							#13	#10	#26
строка								строка											строка		

где #13 и #10 – признак (маркер) конца строки, а #26 – признак (маркер) конца файла.

Так как строки в текстовом файле могут быть разной длины, то в конце каждой строки ставится признак (маркер) конца строки, а в конце файла ставится маркер конца файла.

Доступ к компонентам текстового файла только последовательный, т.е. компоненты файла читаются или записываются только последовательно друг за другом.

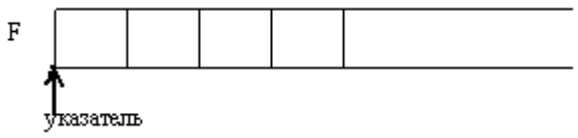
Текстовые файлы работают со всеми внешними устройствами.

Процедуры и функции, определенные над текстовым файлом

1. **Assign(var f:text; filename:string);** – устанавливает соответствие между физическим и логическим именами файла.
   Наличие этой процедуры обязательно.
   Примеры:
   Assign(f, 'w:\info\kl10\a.txt');
   Assign(f,'PRN');
   Writeln('input filename'); {ввод физического имени файла через переменную filename}
   Readln(filename);
   Assign(f,filename);
2. **Reset(var f:text); -** открытие файла для чтения.

Указатель файловой переменной устанавливается перед первой компонентой. Файл готов для чтения. Если файла с таким именем нет, то выдается сообщение об ошибке.

Reset(f);



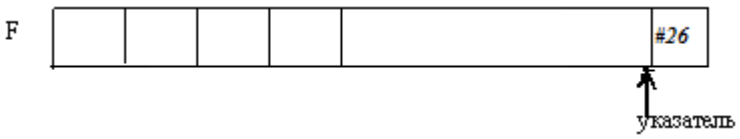
3. **Rewrite(var f:text); -** открытие файла для записи.

Если такого файла нет, то создается метка файла и указатель файловой переменной устанавливается перед первой компонентой. Если файл с таким именем есть, то указатель также устанавливается перед первой компонентой. Содержимое файла при записи будет стерто. За этим должен следить программист.

4. **Append(var f:text); -** открытие текстового файла для дозаписи в конец.

Указатель файловой переменной устанавливается перед маркером конца файла.

Append(f);



Если файла с таким именем нет, то выдается сообщение об ошибке.

5. **Eof(var f:text): boolean; -** определяется конец файла. Функция возвращает значение true, если встретился маркер конца файла (#26).
6. **Eoln(var f:text):boolean; -** определяет конец строки. Функция возвращает значение true, если встретился маркер конца строки (#13,#10).
7. **Read(var f:text; ch:char); -** чтение компоненты типа char из файла.
8. **Readln(var f:text); -** перевод на новую строку. Используется при чтении файла посимвольно. Например:

```
While not eof(f) do
  Begin while not eoln(f) do
                  Read(f,ch);
          Readln(f)
  End;
```

9. **Readln(var f:text; s:string); -** чтение компоненты типа string из файла и перевод на новую строку.

```
While not eof(f) do Readln(f,s);
```

10. **Write(var f:text; ch:char); -** запись компоненты типа char в файл. Write(f,ch);
11. **Writeln(var f:text); -** устанавливается признак конца строки (#13, #10) и переход на новую строку. Writeln(f);
12. **Writeln(var f:text; s:string);** - запись компоненты типа string в файл, установка признака конца строки и переход на новую строку. Writeln(f,s);
13. **Close(var f:text); -** закрытие файла для чтения или записи. Устанавливается маркер конца файла (#26). По окончании работы с файлом он **должен быть обязательно закрыт**, иначе к нему не будет доступа. Close(f);
14. **Seekeof(var f:text):boolean; -** возвращает значение true, если перед маркером конца файла (#26) стоят только пробелы или коды табуляции.
15. **Seekeofln(var f:text):Boolean; -** возвращает значение true, если перед маркером конца строки (#13, #10) стоят только пробелы или коды табуляции.
16. **Rename(var f:text; newname:string); -** переименование файла. Применима лишь к закрытым файлам.
17. **Erase(var f:text); -** удаляет файл. Применима лишь к закрытым файлам.

Особенности текстового файла

1. Это стандартный тип файла.
2. Два типа копонент: символ (char) и строка (string).
3. Только последовательный доступ к компонентам файла.
4. Файл может быть открыт или для чтения (reset(f);) или для записи (rewrite(f);).
5. Возможна дозапись компонент только в конец файла (append(f);).
6. Текстовые файлы работают со всеми внешними устройствами.

Алгоритм работы с файлом

1. Установить соответствие между физическим и логическим именами файлов при помощи процедуры Assign(f,filename).
2. Открыть файл для чтения (reset(f);) или для записи (rewrite(f);).
3. Чтение или запись компонент.
4. Закрыть файл (Close(f)).

Замечание: Одновременно могут быть открыты несколько файлов.

Примеры программ

1. Создание текстового файла на диске.

```
Program example_1;
        Var f:text;
           i, n: integer;
           S: string;
        Begin
             Assign(f, 'w:\info\pascal\file1.txt'); {устанавливаем связь файловой
переменной
с физическим файлом на диске}
           Rewrite(f);    {открываем файл для записи}
           Readln(n);       {определим количество вводимых строк}
           for i:=1 to n do
           begin
              readln(s);     {вводим с клавиатуры строки}
              writeln(f,s); {записываем последовательно строки в файл}
           end;
           close(f);         {закрываем файл}
        end.
```

2. Вывести содержимое текстового файла на экран монитора.

```
Program example_2;
Var f:text;
     S,filename:string;
Begin writeln('input filename');
        Readln(filename);
        Assign(f,filename);          {Связываем  файловую  переменную с файлом
на  диске}
        Reset(f);                    {Открываем файл для чтения}
        While not eof(f) do          {Пока  не достигнут  конец  файла}
           Begin readln(f,s);        {Считываем очередную строку}
                 Writeln(s);         {Выводим строку на экран}
            End;
```

```
        Close(f);                    {Закрываем файл}
End.
```

3. В текстовом файле подсчитать и вывести на экран количество символов в каждой строке.

```
Program example_3;
Var f: Text;
x, k: Integer;
c:char;
Begin
 Assign (f, 'w:\info\pascal\pr.txt');
      {Связываем файловую переменную с файлом на диске}
 Reset (f);       {Открываем файл для чтения}
 x:=0;            {Обнуляем счетчик строк}
 While Not Eof(f) Do  {Пока не достигнут конец файла}
 Begin
  k:=0;           {Обнуляем счетчик элементов строки}
  inc(x);         {Увеличиваем счетчик строк}
  While Not Eoln(f) Do {Пока не достигнут конец строки}
    Begin
     Read(f, c) ;     {Считываем очередной символ}
     Write(c,' ');    {Выводим его на экран}
     Inc(k);          {Увеличиваем счетчик символов}
    End;
  Writeln( ' В ',x, ' строке ', k, ' элементов') ;
  Readln(f);          {Переходим к следующей строке файла}
 End;
 Close(f);    {Закрываем файл}
 Readln;
End.
```

4. Переписать из входного файла в выходной только те строки, которые начинаются с буквы «А» или «а».

```
Program example_4;
        Var f1,f2:text;
           i, n: integer;
           s, filename: string;
        Begin writeln('input filename1');
             Readln(filename);              {вводится имя входного файла}
      Assign(f1,filename);          {устанавливаем связь первой файловой
       переменной с физическим файлом}
             Reset(f1);                     {открываем первый файл для
чтения}
```

```
            writeln('input filename2');
            Readln(filename);                    {вводится имя выходного файла}
             Assign(f2,  filename);              {устанавливаем связь  второй
файловой
        переменной с физическим файлом}
            Rewrite(f2);                         {открываем второй файл для записи}
         {Дальше необходимо последовательно считывать строки из первого файла,
проверять
        выполнение условия и записывать нужные строки во второй файл. Для
чтения из
        текстового файла рекомендуется использовать цикл по условию «пока не
конец
        файла»}
           While not eof(f1) do
           Begin
             Readln(f1,s);                       {считываем очередную строку из
первого файла}
             If (s[1]='A') or (s[1]='a') then
             Writeln(f2,s);                      {записываем во второй файл
строки, удовлетворяющие условию}
           End;
           Close(f1);                            {заканчиваем работу с первым
файлом}
             Close(f2);                          {заканчиваем работу со вторым
файлом}
         End.
```

Типизированные файлы.

Типизированный файл – *это файл произвольного типа, у которого тип компонент может быть любым, кроме файлового.*

Описание типизированного файла:

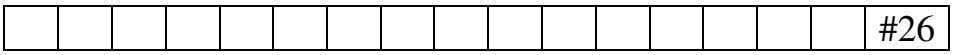
```
Type  tf=file of <тип компонент>;
Var f:tf;
    g:file of <тип компонент>;
```

Например:

```
Type tf=file of real;
     tm=array[1..50] of integer;
Var f:tf;
    g:file of char;
    fm: file of tm;
    x:real;
```

c:char;
m:tm;

Структура типизированного файла - линейная:



Так как все компоненты одного типа, то между ними нет никаких маркеров. Ставится только маркер конца файла (#26). **Компоненты типизированного файла нумеруются с нуля.**

Для типизированного файла реализовано два способа доступа к компонентам:

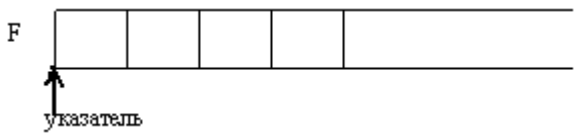
- последовательный (компоненты файла читаются и записываются одна за другим последовательно)
- прямой (можно установить указатель перед любой компонентой и прочитать или записать ее. Указатель устанавливается при помощи процедуры seek) .

Типизированные файлы работают только с магнитными носителями.

Процедуры и функции, определенные над типизированными файлами

1. Assign(var f:tf; filename:string); – устанавливает соответствие между физическим и логическим именами файла.
Наличие этой процедуры обязательно.
Примеры:
Assign(f, ‘w:\info\kl10\a.dat’); в качестве расширения у типизированного файла указывается ‘**dat**’.
Writeln(‘input filename’); {ввод физического имени файла через переменную filename}
Readln(filename);
Assign(f,filename);

2. Reset(var f:tf); - открытие файла для чтения.
Указатель файловой переменной устанавливается перед первой компонентой. Файл готов для чтения. Если файла с таким именем нет, то выдается сообщение об ошибке. Reset(f);



3. Rewrite(var f:tf); - открытие файла для записи.

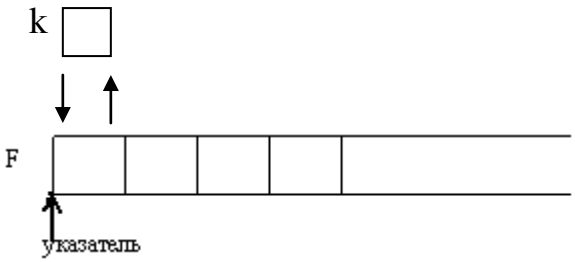
Если такого файла нет, то создается метка файла и указатель файловой переменной устанавливается перед первой компонентой. Если файл с таким именем есть, то указатель также устанавливается перед первой компонентой. Содержимое файла при записи будет стерто. За этим должен следить программист.

4. Eof(var f:tf): boolean; - определяется конец файла. Функция возвращает значение true, если встретился маркер конца файла (#26).

5. Close(var f:tf); - закрытие файла для чтения или записи. Устанавливается маркер конца файла (#26). По окончании работы файл **должен быть обязательно закрыт**, иначе к нему не будет доступа. Close(f);

6. Read(var f:tf; k:<тип компонента**>); -** чтение компоненты из файла в переменную k.

7. Write(var f:tf; k:<тип компонента**>); -** запись из переменной k в компоненту файла.



После чтения/записи компоненты, указатель перемещается к следующей компоненте.

8. FileSize(var F) : LongInt; - возвращает количество компонент типизированного файла**.**

9. FilePos(var F) : LongInt; - возвращает номер позиции указателя файловой переменной.

10. Seek(var F; N: LongInt); - устанавливает указатель перед компонентой N.

Например: seek(f,0); - устанавливает указатель перед нулевой компонентой.
seek(f,5); - устанавливает указатель перед пятой компонентой.

11. Truncate(var F); - удаляется часть файла, начиная с текущей позиции указателя и до его конца.

Например: seek(f,15);
truncate(f); - удаляются все компоненты, начиная с 15-й и до конца файла.

Так как типизированные файлы работают только с магнитными носителями, то ввод и вывод компонент файла осуществляется опосредованно через текстовый файл.

Особенности типизированного файла.

1. Это произвольный тип файла.
2. Один тип компонент (любой, кроме файлового).
3. Доступ к компонентам файла прямой и последовательный.

4. Файл открывается процедурами reset и rewrite и для чтения и для записи.
5. Типизированный файл работает только с магнитными носителями.

Примеры программ

1. Создание типизированного файла. (**Типизированный файл нельзя создать при помощи текстового редактора.**)

```
Var f:file of real;
     k:real;
     fname:string;
     n,i:integer;
begin writeln('input fname');
       readln(fname);
       assign(f,fname);
       rewrite(f);
       writeln('input n');
       readln(n);
       for i:=1 to n do
          begin readln(k);
                write(f,k);
           end;
        close(f);
end.
```

2. Подсчитать сумму всех компонент целочисленного типизированного файла. (Последовательный доступ.)

```
Var f:file of integer;
     fname:string;
     k, sum:integer;
begin writeln('input fname');
       readln(fname);
       assign(f,fname);
       reset(f);
       sum:=0;
       while not eof(f) do
          begin read(f,k);
                sum:=sum+k;
          end;
        close(f);
        writeln('sum=',sum);
end.
```

3. В типизированном файле вещественного типа поменять местами первую и последнюю компоненты. (Прямой доступ.)

```
Var f:file of real;
      x,y:real;
      fname:string;
begin writeln('input fname');
        readln(fname);
        assign(f,fname);
        reset(f);
        read(f,x);
        seek(f,filesize(f)-1);
        read(f,y);
        seek(f,0);
        write(f,y);
        seek(f,filesize(f)-1);
        write(f,x);
        close(f);
end.
```

Задание.

Написать и отладить следующие программы:

1. Подсчитать в текстовом файле количество строк.
2. Подсчитать в текстовом файле количество слов.
3. Найти в текстовом файле слово максимальной длины.
4. Создать новый текстовый файл из исходного, записывая в него только те строки, в которых есть слово 'Procedure'.

Написать и отладить следующие программы:

1. Дан файл f, состоящий из действительных чисел. Найти сумму наибольшего и наименьшего из его компонент.
2. Дан целочисленный типизированный файл f. Все отрицательные элементы заменить нулями.
3. Дан типизированный файл символьного типа. В конец файла дописать символы 'e','n','d'.
4. Дан целочисленный файл. Последний и центральный элементы поменять местами.

3.1. Использование стандартных модулей, создание структурных и модульных программ.

Задание:

При выполнении задания по модульному программированию студент должен создать отдельный модуль, реализующий простейшие операции над данными указанного типа. С помощью созданного модуля необходимо решить основную задачу.

Перед написанием основной программы, необходимо проверить правильность работы всех процедур модуля. Для этой цели необходимо разработать тестовые задания, написать программу тестирования модуля.

При написании программы решения задачи необходимо уделить особое внимание интерфейсу: оформление, контроль правильности ввода, вывод сообщений и т.д.

Задачи:

Вариант 1.

Модуль. Реализовать операции над целыми числами в 16-ричной системе счисления: сложения, умножения, операцию преобразования из 10-тичной в 16-ричную систему, операции вычитания, деления над целыми числами в 16-ричной системе счисления, операцию преобразования из 16-ричной в 10-тичную систему.

Задача. Дан массив целых чисел, представленных в 16-ричной системе счисления. Найти произведение наибольшего и наименьшего чисел. Вычислить сумму элементов массива, расположенных между наибольшим и наименьшим числами. Ответ выдать в десятичной и 16-ричной системе счисления.

Вариант 2.

Модуль. Реализовать операции над целыми числами в 2-ичной системе счисления: сложения, умножения, операцию преобразования из 10-тичной в 2-ичную систему, операции вычитания, деления над целыми числами в 2-ичной системе счисления, операцию преобразования из 2-ичной в 10-тичную систему.

Задача. Дан массив целых чисел, представленных в 2-ичной системе счисления. Вычислить сумму наибольшего и наименьшего чисел. Найти произведение чисел, расположенных в массиве левее наибольшего.

Вариант 3.

Модуль. Реализовать операции над целыми числами в 8-ричной системе счисления: сложения, умножения, операцию преобразования из 10-тичной в 8-ричную систему, операции вычитания, деления над целыми числами в 8-ричной системе счисления, операцию преобразования из 8-ричной в 10-тичную систему.

Задача. Дан массив целых чисел, представленных в 8-ричной системе счисления. Вычислить индексы наибольшего и наименьшего чисел. Найти произведение чисел, расположенных в массиве правее наименьшего.

Вариант 4.

Модуль. Реализовать операции преобразования чисел из десятичной системы счисления в 2-ичную, 8-ричную, 16-ричную, а также функцию нахождения большего из двух чисел, операции преобразования чисел из 2-ичной, 8-ричной, 16-ричной систем счисления в десятичную, а также функцию нахождения меньшего из двух чисел.

Задача. Даны 4 целых числа, представленных в различных системах счисления: A_{10}, B_2, C_8, D_{16}. Вычислить значение выражения A*D-C*B. Результат вывести в каждой системе счисления по основанию 10, 2, 8, 16. Вывести числа в порядке возрастания в тех системах счисления, в которых они были введены.

Вариант 5.

Модуль. Реализовать сложение, вычитание, умножение длинных целых чисел, операции отношения для длинных целых чисел.

Задача. Даны два натуральных длинных числа. Вычислить значение выражения $(\max(n, m) - \min(n,m))^2$.

Вариант 6.

Модуль. Реализовать операции сложения, умножения, дифференцирования многочленов, операции вычитания, деления, интегрирования многочленов, вычисления значения многочлена в заданной точке.

Задача. Даны два многочлена P(x) и Q(x). Вычислить

$$\int_a^b (P(x)^2 - Q(x))dx$$

Вариант 7.

Модуль. Реализовать набор подпрограмм для работы с векторами: сложение, вычитание, вычисление длины вектора, скалярное произведение векторов, умножение вектора на число, вычисление угла между векторами.

Задача. Дан массив векторов. Найти вектор, который образует с самым длинным вектором массива наибольший угол.

Вариант 8.

Модуль. Реализовать в виде модуля набор подпрограмм для выполнения операций над комплексными числами: сложения, вычитания, умножения, деления, модуля комплексного числа, возведения комплексного числа в натуральную степень.

Задача. Дан массив комплексных чисел. Получить новый массив, элементами которого будут модули сумм рядом стоящих комплексных чисел.

Вариант 9.

Модуль. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над векторами: сложения, вычитания, скалярного умножения векторов, умножения вектора на число, нахождение длины вектора.

Задача. Дан массив векторов. Отсортировать его в порядке убывания длин векторов, найти скалярное произведение самого короткого вектора на самый длинный.

Вариант 10.

Модуль. Разработать набор подпрограмм для работы с длинными неотрицательными числами (числами, выходящими за диапазон допустимых значений любого целого типа): сложение, вычитание, умножение, нахождение частного и остатка от деления одного числа на другое, операции отношения. Длинное число представить следующим типом:

```
Type cifra=0..9;
     Chislo= array [1..1000] of cifra;
```

Задача. Составить программу вычисления числа 2^{64}-3^{46}, в результате сохранить все цифры.

Вариант 11.

Модуль. Разработать способ представления множеств, содержащих более 255 элементов. Реализовать операции над множествами: объединение, пересечение, разность, функцию проверки принадлежности элемента множеству, функцию проверки, является ли данное множество подмножеством (надмножеством) другого.

Задача. Даны три множества A, B, C, содержащие более 255 элементов. Найти множество $D = A \bigcap (B \setminus C)$.

Вариант 12.

Модуль. Реализовать набор подпрограмм для работы с дробными двоичными числами: сложения, вычитания, умножения, деления, перевода из двоичной в десятичную систему счисления и обратно, операции отношения.

Задача. Дан массив дробных чисел в двоичной системе счисления. Вычислить сумму тех элементов массива, которые больше заданного двоичного числа. Результат представить в двоичной и десятичной системах счисления.

Вариант 13.

Модуль. Реализовать набор подпрограмм для работы с дробными числами в 8-ричной системе счисления: сложение, вычитание, умножение, деление, перевод из 8-ричной системы в 10-тичную и обратно, операции отношения.

Задача. Дан массив дробных чисел в 8-ричной системе счисления. Вычислить произведение элементов массива, не превышающих заданного числа, представленного в десятичной системе счисления.

Вариант 14.

Модуль. Реализовать набор подпрограмм для выполнения действий над многочленами: сложение, вычитание, умножение, деление.

Задача. Дано дробно-рациональное выражение $\frac{2P(x)}{P(x)-3Q(x)}$, где P(x) и Q(x) – многочлены. Выполнить действия и найти наибольший общий делитель числителя и знаменателя, сократить и вывести упрощенное выражение на экран.

Лабораторные работы

Подготовка и требования к выполнению лабораторных работ

Для выполнения лабораторных работ необходим персональный компьютер с установленной средой программирования Pascal 7.1 (Free Pascal).

На занятиях лабораторного цикла каждый студент получает индивидуальное задание, направленное на формирование компетенций, определенных данной рабочей программой. Лабораторная работа предусматривает реализацию полученных студентами знаний через организацию учебной работы в среде программирования Turbo Pascal по реализации, отладке и тестированию программ на ЭВМ.

По каждой лабораторной работе учащиеся должны получить у преподавателя индивидуальное задание и выполнить его. Перед выполнением практической работы учащиеся должны освоить навыки работы с интерфейсом интегрированной среды, ответить на контрольные вопросы, выполнить работу согласно предложенному порядку.

Во время выполнения заданий в учебной аудитории студент может консультироваться с преподавателем, определять наиболее эффективные методы решения поставленных задач. Если какая-то часть задания остается невыполненной, студент может продолжить её выполнение во время внеаудиторной самостоятельной работы. .

Отчет оформляется в тетради и представляется преподавателю на проверку по завершению изучения темы.

Для выполнения лабораторной работы необходимо:

1. Изучить краткие теоретические сведения, необходимые для успешного выполнения конкретной работы.
2. Внимательно изучить все примеры программ, рассмотренные в лекции и представленные в описании лабораторной работы.
3. Ответить на контрольные вопросы, предложенные в данной лабораторной работе.
4. Выполнить индивидуальные задания: составить программу, произвести ее отладку и тестирование в среде Turbo Pascal.
5. Оформить отчет о выполненной лабораторной работе в соответствии с образцом

Отчет должен содержать:

- Название темы
- Цель работы
- Условие задачи и описание используемых переменных и констант.
- Тексты программ по данной теме.
- Результат выполнения программы и, при необходимости, ручную трассировку.

Отчет о лабораторной работе принимает преподаватель во время лабораторного занятия. В процессе защиты оценивается самостоятельность

работы, понимание механизма работы алгоритма, знание используемых в программе операторов, умение анализировать результаты выполнения программы.

Лабораторная работа №1. Алгоритмы разветвляющихся и циклических процессов.

Разветвляющийся алгоритм - алгоритм, содержащий хотя бы одно условие, в результате проверки которого ЭВМ обеспечивает переход на один из двух возможных шагов.

Пример.

Даны целые числа x, y. Определить, принадлежит ли точка с координатами x, y кругу радиуса r.

Решение:

Входные данные: **r** - радиус окружности, x, y - координаты точки *Выходные данные:* **Вывод надписи на экран: входит- не входит.**

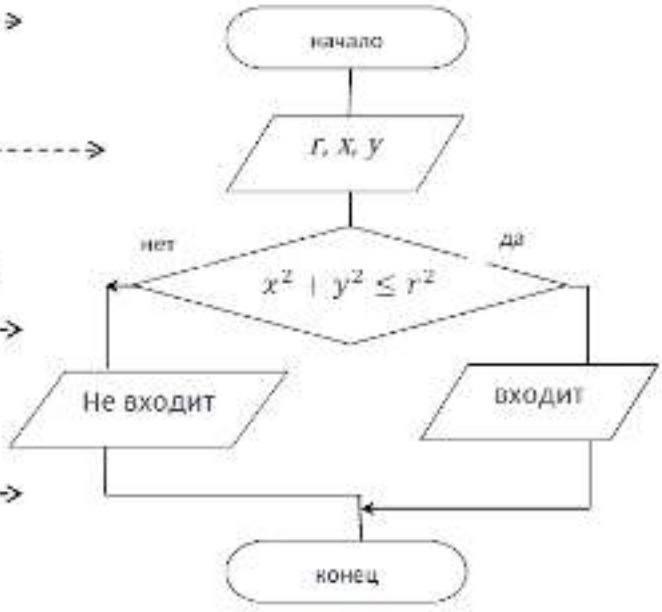
Для большей наглядности приведем пример словесного представления алгоритма с графическим.

1. Начало ---------->
2. Ввести радиус окружности и координаты точки: *r, x, y.* ---------->
3. Проверить, выполняется ли условие: $x^2 + y^2 \le r^2$ ---------->

3.1 Если условие выполняется, то ---> на экран выводится надпись: входит

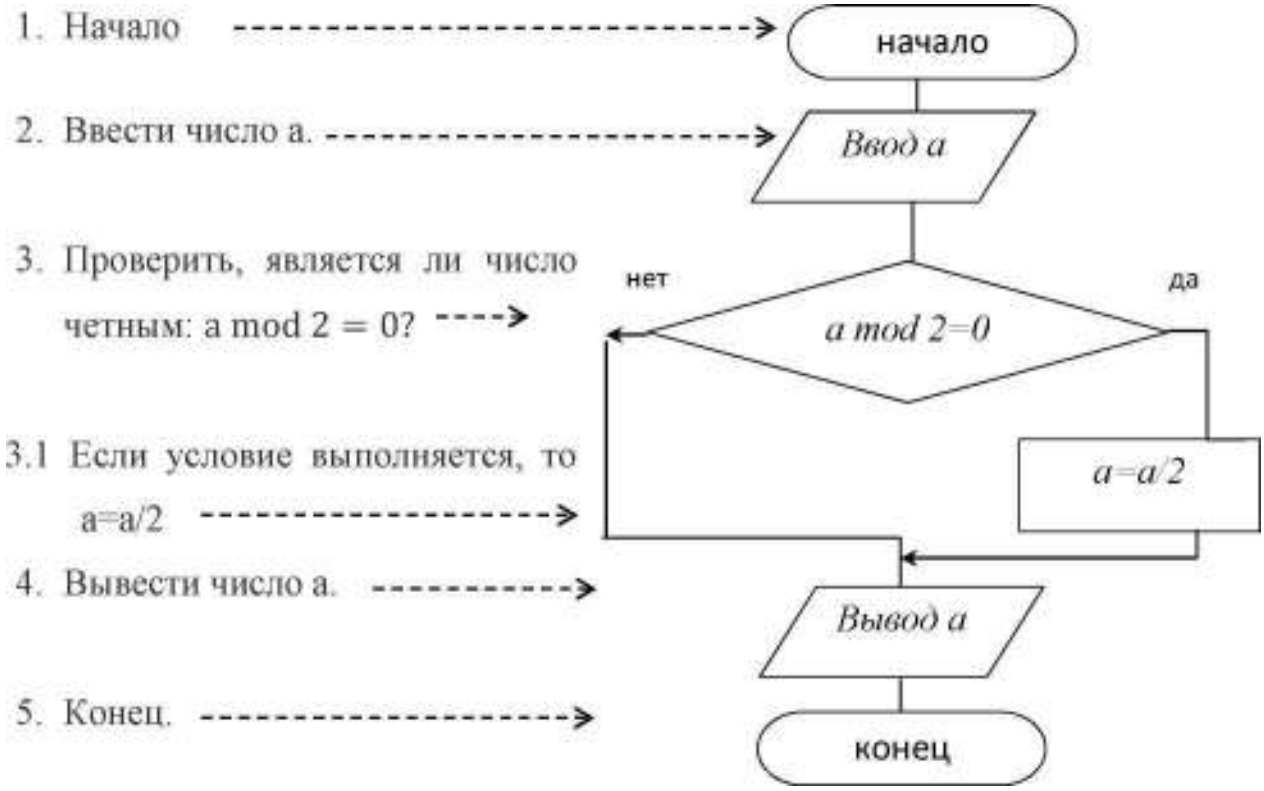
3.2 Если условие не выполняется, то на экран выводится ---> надпись: не входит

4. Конец ---------->



Пример.

Дано число a. Если оно четное, то разделить его на 2. Вывести число a



на экран. *Входные данные:* **число a.** *Выходные данные:* **число a.**

В некоторых задачах ветвление представлено как множественный выбор. Это отражено в следующем примере.

Пример.

Вычислить значение функции в точке:

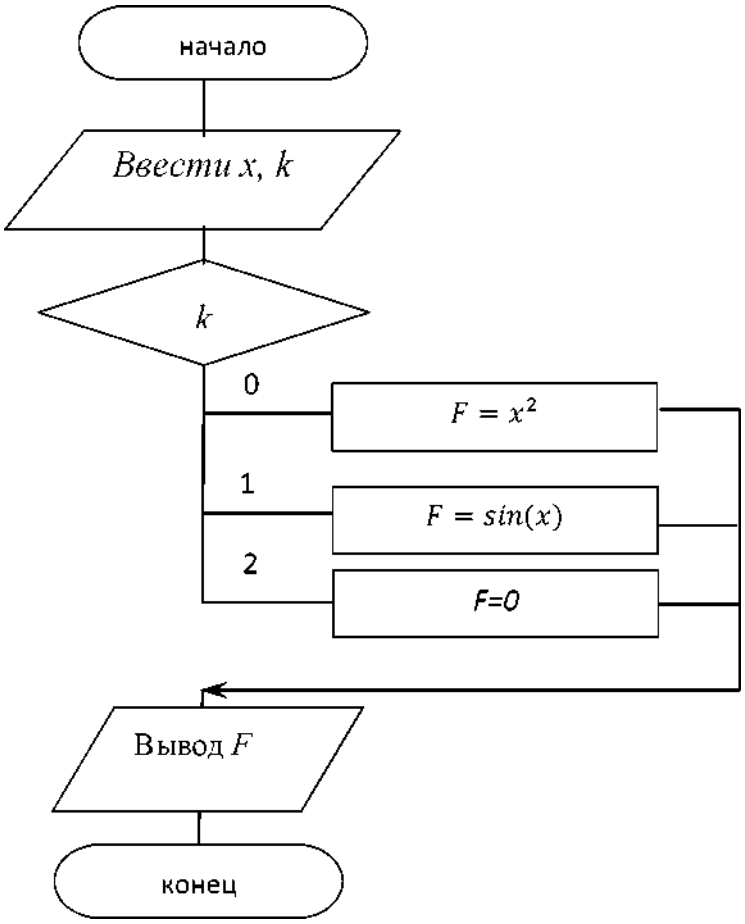
$$F(x)=\begin{cases} x^2, \text{если } k=0 \\ \sin(x), \text{если } k=1 \\ 0, \text{если } k=2 \end{cases}$$

Решение:

Входные данные: **число** *к,* **параметр функции** *x.*

Выходные данные: **значение функции** *F* **в точке x при определенном** *к.*

Блок-схема:



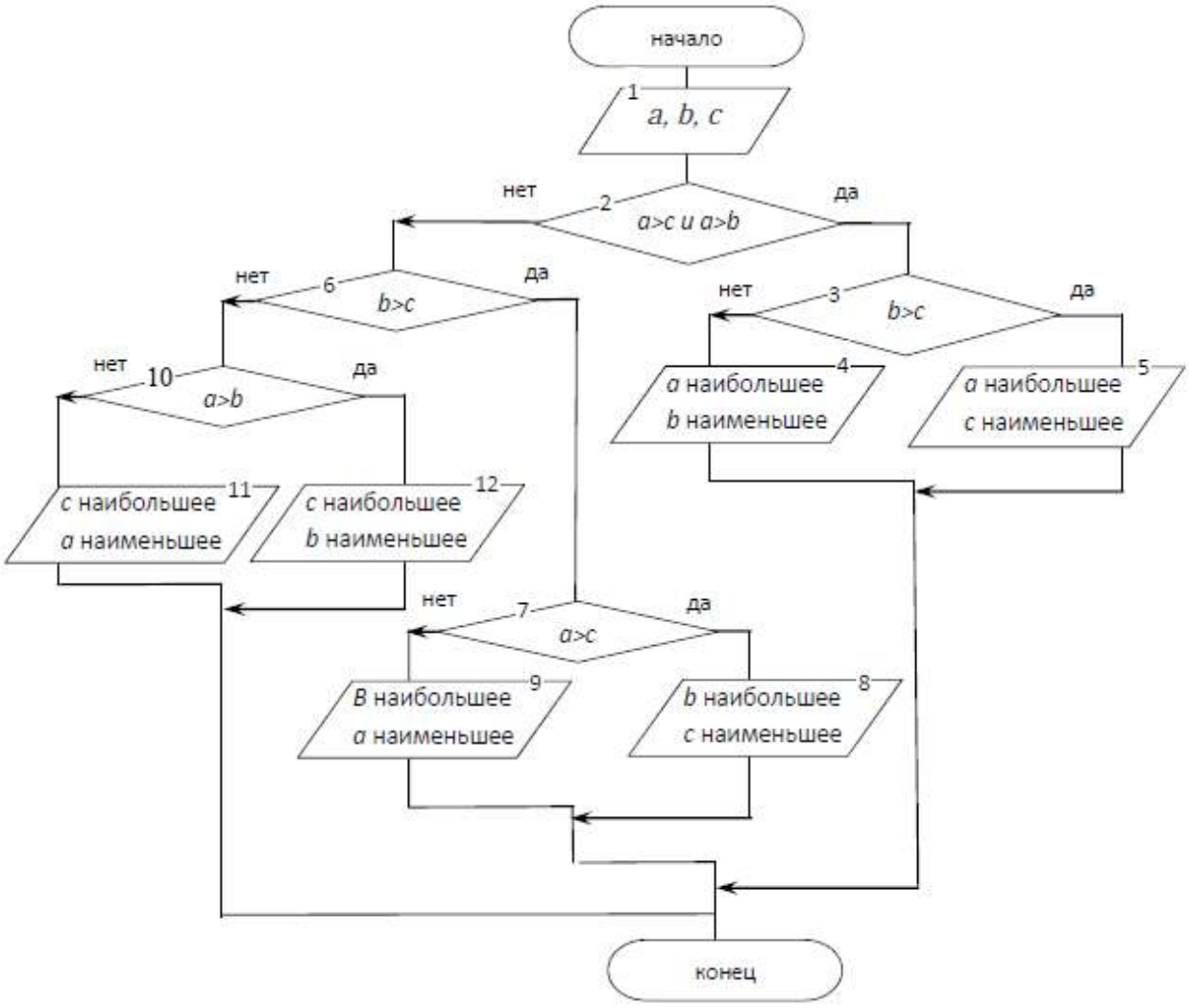
Пример.

Даны три различных числа: a, b, c. Определить самое наибольшее и самое наименьшее из них.

Решение:

Входные данные: числа: a, b, c.

Выходные данные: значения наибольшего числа и наименьшего числа. Блок-схема:



Выполним трассировку этого алгоритма для различных a b и c. 1. a=3, b=2, c=6

№ шага	№ блока	A	b	c	Примечание
1	1	3	2	6	Вводим значения переменных
2	2				a>b и a>c = ложь (Не выполняется 2е условие)
3	6				b>c = ложь
4	10				a>b =истина
12					Выводится «c-наибольшее, b-наименьшее»

2. a=5, b=5, c=4

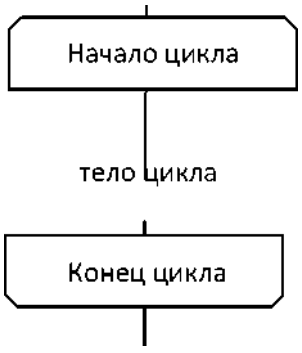
№ шага	№ блока	A	b	c	Примечание
1	1	5	5	4	Вводим значения переменных
2	2				a>b и a>c = ложь (Не выполняется 1 - е условие)
3	6				b>c = истина
4	7				8>c =истина
12					Выводится «b-наибольшее, c-наименьшее»

Циклический алгоритм - алгоритм, предусматривающий многократное повторение одного и того же действия (одних и тех же операций) Над новыми исходными данными. К циклическим алгоритмам сводится большинство методов вычислений, перебора вариантов. Цикл программы — последовательность команд (серия, тело цикла), которая может выполняться многократно (для новых исходных данных) до удовлетворения некоторому условию.

Циклические алгоритмы в свою очередь можно разбить на два вида: явные (когда заранее известно количество повторений) и неявные (когда заранее количество повторений неизвестно, концом цикла является выполнение условия).

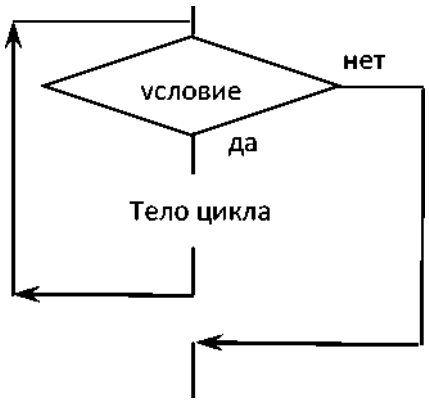
Неявные циклические алгоритмы также можно разделить еще на два вида: с пред условием (сначала проверяется условие, если оно выполняется, делается повторение цикла) и с пост условием (сначала делается повторение цикла, а потом проверяется условие и если оно выполняется, то происходит следующее повторение цикла).

В блок- схеме явный циклический алгоритм представляется следующим образом:

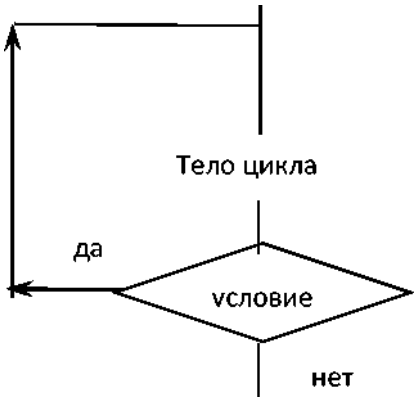


Неявный цикл в блок - схеме обозначается следующим образом:

Цикл с предусловием



Цикл постусловием



Рассмотрим пример с использованием явного цикла.

Пример.

Дана последовательность, каждый член которой вычисляется по правилу: $x_0 = 1$, $x_г = 2x_0$,...,Xj = 2x;_x. Вычислить первые 20 членов последовательности.

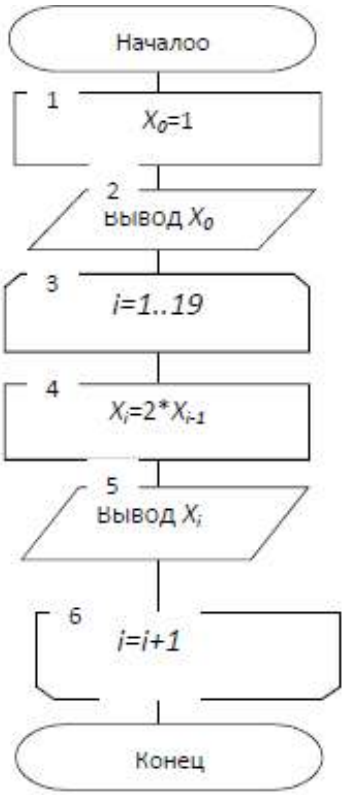
Решение:

Так как нам заранее известно количество повторений (по условию 20 нужно вычислять *x),* то будем использовать явный цикл.

Входные данные: число $x_0 = 1$. *Выходные данные:* значения x*

Снова рассмотрим словесную и графическую запись алгоритма для большей наглядности:

1. Начало ---------->
2. Присвоить $x_0 = 1$ ---------->
3. Вывести x_0 ---------->
4. Повторять 19 раз: ---------->
   - 4.1 $x_i = 2x_{i-1}$ ---------->
   - 4.2 вывести на печать x_i ---------->
   - 4.4. Переход к следующей итерации ---->
5. Конец ---------->



Пример 8.

Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый следующий день он увеличивал дневную норму на 50% от нормы предыдущего дня. Через сколько дней спортсмен пробежит суммарный путь 60 км?

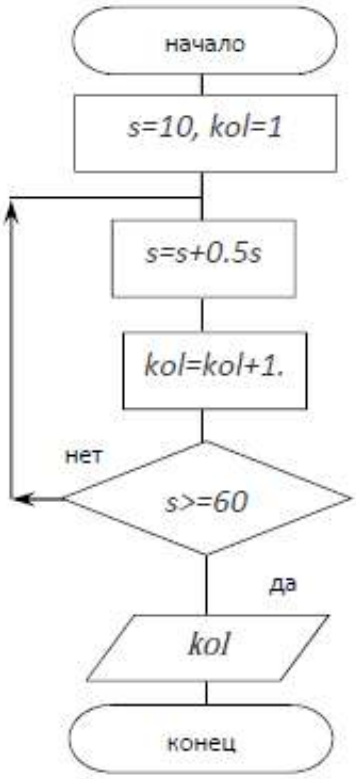
Решение:

В данном случае заранее неизвестно сколько дней должен бегать спортсмен, но известно что всего он должен пробежать 60 км. Значит, будет использоваться неявный цикл, условием выхода из которого будет суммарный путь >=60км.

Входные данные: нет

Выходные данные: kol-количество дней, необходимых для суммарного пути >=60.

1. Начало.
2. Ввести *s, kol.*
3. Посчитать сколько километров пробежал за следующий день и получить суммарный путь: *s=s+0.5s* .
4. Защитать еще один день бега *kol=kol+1.*
5. Если суммарный путь меньше 60, то повторить шаги 3-4, если больше либо равно, то перейти к пункту 6.
6. Вывести на экран *kol.*
7. Конец.



Пример.

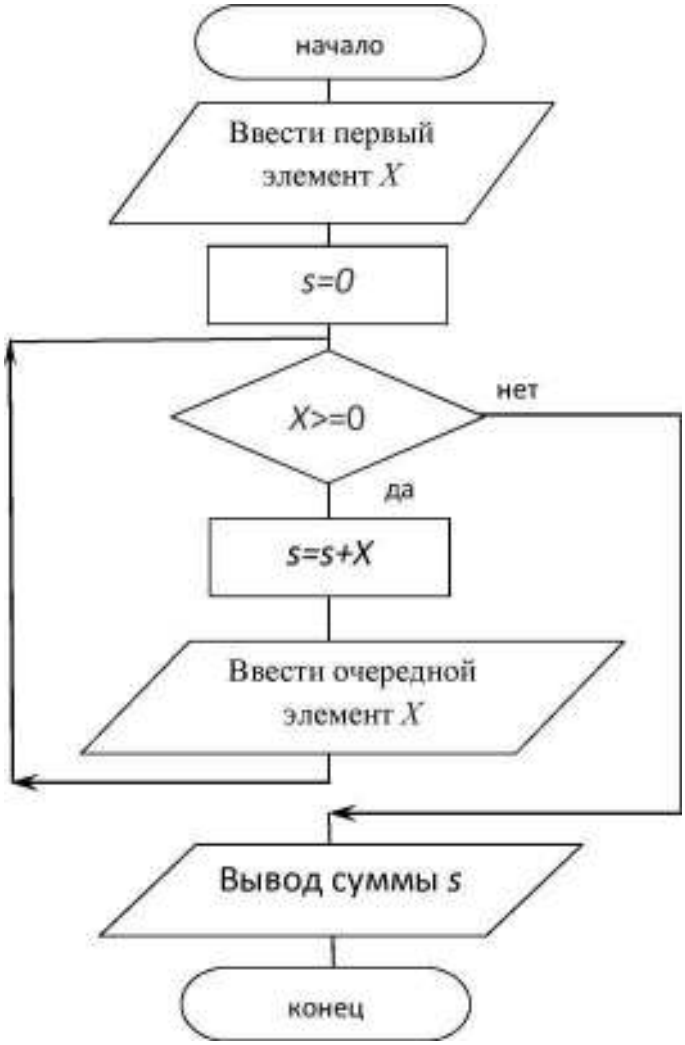
Найти сумму элементов последовательности до первого отрицательного элемента. Элементы последовательности вводятся пользователем.

Решение:

Входные данные: X- элементы последовательности.

Выходные данные: s - сумма элементов последовательности до первого отрицательного.

Блок-схема:



Вводим первый элемент последовательности
Изначально сумма элементов равно нулю.
Пока очередной элемент последовательности больше либо равен нуля
Добавляем его к сумме
Вводим значение следующего элемента на
Когда введем отрицательный элемент - выйдем из цикла и выведем значение суммы на экран

Пример.

Дана последовательность из 100 чисел. Определить, встречаются ли в этой последовательности числа, кратные 13.

Решение:

Входные данные: Элементы последовательности.

Выходные данные: Срока ответа «Да, встречаются» или «Нет, не встречаются»

Блок-схема:

Комментарий к решению: переменная *bFind* логического типа фиксирует результат поиска заданных чисел в последовательности: если она равна *false,* значит среди введенных чисел не было ни одного, кратного 13. Если найдется хотя бы один элемент, кратный 13, то переменная *bFind* устанавливается в *true.* Первоначально устанавливаем эту переменную в *false,* предполагая, что таких чисел нет. В процессе ввода элементов последовательности проверяем каждый элемент на кратность 13 и если хотя бы один элемент кратен 13 - переменную *bFind* устанавливаем в *true.* По выходе из цикла результирующая строка выбирается на основе значения *bFind.*

Варианты заданий к лабораторной работе.

Вариант№1

1. Даны n целых чисел. Определить, являются ли эти числа равными или все они не меньше заданного A.
2. Дана последовательность чисел. Определить порядковый номер числа, которое содержит наибольшую цифру в десятичной записи.

Вариант№2

1. Дана последовательность целых чисел, конец которой обозначен нулем. Определить, кратны ли числа последовательности своему порядковому номеру.
1. Дано число. Разделить цифры десятичной записи этого числа, стоящие на нечётных местах на 3. Если не делятся без остатка, то оставить без изменения.

Вариант№3

1. Дано n целых чисел. Найти количество отрицательных значений и количество тех значений, которые больше первого из n чисел. Числа по одному вводятся с клавиатуры.
2. Дана последовательность чисел. Посчитать сумму цифр десятичной записи всех отрицательных чисел.

Вариант№4

1. Дана последовательность целых чисел, конец которой обозначен нулем. Определить, все ли числа являются положительными или положительные числа чередуются с отрицательными.
2. Дана последовательность чисел. Посчитать сумму цифр десятичной записи всех чётных чисел.

Вариант№5

1. Дана последовательность целых чисел. Известно, что среди них несколько раз встречаются два подряд идущих нуля. Определить, сколько раз встречается эта ситуация.
2. Дана последовательность чисел. Посчитать произведение цифр десятичной записи последнего числа, кратного 5.

Вариант№6

1. Дана последовательность целых чисел до 0. Найти сумму целых чисел, которые одновременно больше 20, меньше 100 и кратны 3.
2. Дана последовательность чисел. Посчитать произведение цифр десятичной записи первого числа, кратного 3.

Вариант№7

1. Дана последовательность n целых чисел, где n- задано. Определить ,все ли числа попадают в заданный интервал [x, y].
2. Дано число. Посчитать сумму цифр в десятичной записи этого числа, стоящих на чётных местах числа.

Вариант№8

1. Дана последовательность вещественных чисел. Определить, образуют

ли они возрастающую последовательность.

2. Дано число. Посчитать произведение тех цифр в десятичной записи этого числа, которые кратны 3.

1. Дана последовательность целых чисел. Определить количество чисел, кратных разности текущего и предыдущего чисел.

2. Дано число. Посчитать разность между первой и последней цифрой десятичной записи этого числа.

Вариант№10

1. Дана последовательность из 100 целых чисел. Определить количество чисел в наиболее длинной подпоследовательности из подряд идущих нулей.

2. Дано число. Разделить каждую цифру десятичной записи этого числа на его порядковый номер. Полученное число напечатать.

Вариант№11

1. Дана последовательность целых чисел, конец которой обозначен нулем. Определить, кратны ли числа последовательности своему порядковому номеру.
2. Дано число. Разделить цифры в десятичной записи этого числа, стоящие на нечётных местах на 3. Если не делятся без остатка, то оставить без изменения.

Вариант№12

1. Дана последовательность вещественных чисел.
Определить порядковый номер первого отрицательного числа.
2. Дано число. Поменять местами соседние цифры в десятичной записи этого числа.

Вариант№13

1. Дана последовательность целых чисел, среди которых есть два нуля. Найти сумму чисел, расположенных между ними.
2. Дано число. Определить является ли сумма цифр десятичной записи этого числа степенью числа 2.

Вариант№14

1. Дано натуральное число. Определить, является ли оно степенью целого числа в диапазоне от 2 до 9.
2. Дано целое число. Определить, отсортированы ли по возрастанию цифры в десятичной записи этого числа.

Вариант№15

1. Дана последовательность целых чисел. Определить, является ли она сверхвозрастающей последовательностью
2. Дано целое число. Разложить это число на простые множители.

1. Дана последовательность вещественных чисел, определить, являются ли они возрастающими по величине дробной части.
2. Дано число. Определить, есть ли в десятичной записи этого числа рядом стоящие одинаковые цифры.

Вариант№17

1. Дана последовательность целых чисел, определить, соблюдается ли в этой последовательности принцип - четные числа на четных позициях, нечетные - на нечетных.
2. Дано целое число. Определить, является ли десятичная запись этого числа палиндромом (читается одинаково справа налево и слева направо, например 43534).

Вариант№18

1. Дана последовательность целых чисел, оканчивающаяся нулем. Определить, является ли эта последовательность арифметической прогрессией.
2. Дано целое число. Определить, одинаковы ли все цифры в десятичной записи этого числа.

Вариант№19

1. Дана последовательность вещественных чисел. Найти и вывести на экран все локальные минимумы - числа, меньшие предыдущего и последующего.
2. Среди чисел от 10 до 1000, наяти и вывести на экран те, которые меньше произведения цифр десятичной записи этого числа

Вариант№20

1. Даны два натуральных числа n и m. Среди чисел в диапазоне от n до m найти то, у которого наибольшее количество делителей.
2. Среди чисел от 1 до 1000 найти все числа Армстронга. Число Армстронга - натуральное число, которое в данной системе счисления равно сумме своих цифр, возведённых в степень, равную количеству
   его цифр (например, $153=1^3+5^3+3^3$)

Лабораторная работа №2. Программирование циклов и матричных задач.

Программирование циклических алгоритмов: цикл с параметром.

Цель: закрепить практические навыки работы с системой Turbo Pascal, научиться правильно использовать оператор цикла с параметром; научиться составлять программы решения задач с использованием циклических структур.

Теоретические сведения:

Для реализации в языке Pascal используется составной оператор, состоящий из операторов for, to, downto, do и при необходимости из операторных скобок. Переменная параметр обязательно объявляется в декларационной части программы и может принадлежать одному из порядковых типов. Если при изменении переменной параметра необходимо использовать переход к следующему значению, то используется оператор to; если переход необходимо осуществить к предыдущему значению, то используется оператор downto. Тогда в общем виде цикл записывается так:

```
for I:=I0 to In do
  begin
    <оператор 1>;
```

<оператор 2>;
...
<оператор n>;
end; где I_0 и I_n – начальное и конечное значения.

Порядок выполнения работы:

1) Составьте программу, *выводящую на экран квадраты и кубы чисел от 10 до 20.* Откомпилируйте программу. Проверьте правильность решения задачи.

2) Составьте программу решения задачи: *Найти значение выражения* $5x^9+13x^7-2x^{11}$, *используя оператор цикла с параметром.*

3) Составьте программу, *выводящую на экран таблицу умножения чисел от 1 до 9*. Откомпилируйте программу. Проверьте правильность решения задачи.

4) Составьте программу решения задачи: Дано натуральное число n. Вычислить

$$S=1!+2!+3!+4!+\ldots+n!\ (n>1).$$

5) Составьте программу решения задачи: *Сколько можно купить ручек, тетрадей и карандашей, платя за ручку 5 рублей, за тетрадь – 10 рублей, за карандаш – 0,5 рублей, если на 100 рублей нужно купить 100 предметов*. Откомпилируйте программу. Проверьте правильность решения задачи.

6) Запишите полученные результаты в тетрадь.

7) Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

8) Выйдите из программы Turbo Pascal.

Программирование циклических алгоритмов: цикл с предусловием.

Цель: закрепить практические навыки работы с системой Turbo Pascal, научиться правильно использовать операторы цикла с условием; научиться составлять программы решения задач с использованием циклических структур.

Теоретические сведения:

Цикл с предусловием.

Для реализации циклов с предусловием используется составной оператор, включающий оператор while, do, операторные скобки.

В общем виде цикл реализуется записью:

while <условие> do <действие>;

Если тело цикла содержит более одного действия, то необходимо использовать операторные скобки:

while <условие> do
begin
<оператор 1>;
<оператор 2>;
...

<оператор n>;
end;

Порядок выполнения работы:

1) Составьте программу решения задачи: *Дано натуральное число n. Вычислить S=1!+2!+3!+4!+…+n! (n>1).*Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

2) Составьте программу решения задачи: *Программа должна запрашивать пароль (например, четырехзначное число) до тех пор, пока он не будет правильно введён.* Откомпилируйте программу. Проверьте правильность решения задачи.

3) Составьте программу решения задачи: *Найти сумму и произведение всех трехзначных чисел, кратных 7.* Откомпилируйте программу. Проверьте правильность решения задачи.

4) Составьте программу решения задачи: *Одноклеточная амёба каждые 3 часа делится на 2 клетки. Определить сколько амёб будет через 3, 6, 9, 12, 15, 18, 21,24 часа.* Откомпилируйте программу. Проверьте правильность решения задачи.

5) Запишите полученные результаты в тетрадь. Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

6) Выйдите из программы Turbo Pascal.

Программирование циклических алгоритмов: цикл с постусловием.

Цель: закрепить практические навыки работы с системой Turbo Pascal, научиться правильно использовать операторы цикла с условием; научиться составлять программы решения задач с использованием циклических структур.

Теоретические сведения:

Цикл с постусловием.

Для реализации цикла используется составной оператор, состоящий из операторов repeat и until. В общем виде цикл записывается так:

repeat
<действие>;
until <условие>;

Порядок выполнения работы:

1) Составьте программу решения задачи: *В банк положили 1000 рублей под проценты. Каждый месяц сумма увеличивается на 5% от суммы предыдущего месяца. Через сколько месяцев сумма на счету достигнет 2000 рублей?* Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

2) Составьте программу решения задачи: *Введите целое число и определите количество цифр в нем.* Откомпилируйте программу. Проверьте правильность решения задачи.

3) Составьте программу решения задачи: *Найти среднее арифметическое натуральных чисел, меньших 200 и кратных 7.*Откомпилируйте программу. Проверьте правильность решения задачи.

4) Составьте программу решения задачи: *Найти все трехзначные числа, средняя цифра которых равна сумме первой и последней.* Откомпилируйте программу. Проверьте правильность решения задачи.

5) Запишите полученные результаты в тетрадь. Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

6) Выйдите из программы Turbo Pascal.

Обработка одномерных и двумерных массивов.

Цель: научиться описывать, заполнять, выводить и обрабатывать одномерные и двумерные массивы.

Теоретические сведения:

Массив – группа элементов одного типа, объединенных под общим именем.

Описание массивов

Массивы описываются в разделе описания переменных Var.

Общий вид описания одномерного массива:

<имя массива>: array [<начальный индекс>..<конечный индекс>] of <тип элемента>;

где имя - имя переменной-массива; array - ключевое слово, обозначающее, что переменная является массивом; нижний_индекс и верхний_индекс - целые числа, определяющие диапазон изменения индексов (номеров) элементов массива и, неявно, количество элементов (размер) массива; тип - тип элементов массива.

Общий вид описания двумерного массива:

<имя массива>:array[<m_1>..<m_2>,<n_1>..<n_2>] of <тип>;

где *Имя* - имя массива; *array* - слово языка Pascal, показывающее, что описываемый элемент данных - массив; m_1, m_2, n_1, n_2- константы или выражения типа INTEGER, определяющие диапазон изменения индексов и, следовательно, число элементов массива; *Тип* - тип элементов массива.

Заполнение массива

Под вводом массива понимается ввод значений элементов массива. Ввод удобно реализовать при помощи инструкции FOR. Чтобы пользователь программы знал, ввода какого элемента массива ожидает программа, следует организовать вывод подсказок перед вводом очередного элемента массива. В подсказке обычно указывают индекс элемента массива. Заполнение массива можно производить:

1) с клавиатуры: For i:=1 to n do readln(a[i]);

2) через датчик случайных чисел: Randomize; For i:=1 to n do begin a[i]:=random(i);

Если требуется, чтобы значения элементов массива выбирались из определенного интервала [a,b], то a+Random(b-a+1);

3) через оператор присваивания (по формуле): For i:=1 to n do

a[i]:=i*3;

Вывод массива

Если в программе необходимо вывести значения всех элементов массива, то для этого удобно использовать инструкцию FOR, переменная-счётчик которой может быть реализована как индекс элемента массива. Например, For i:=1 to n do writeln(a[i]);

Удаление элементов из одномерного массива.

Для того, чтобы удалить из массива k-ый элемент нужно: найти номер элемента k; сдвинуть все элементы, начиная с k-го, на один элемент влево; последнему элементу присвоить значение, равное 0; уменьшить количество элементов массива на единицу.

Вставка элемента в одномерный массив.

Вставлять элемент можно до или после данного элемента, номер этого элемента можно вводить с клавиатуры или искать при определенных условиях. Пусть k - это номер элемента, после которого мы должны вставить элемент x. Тогда вставка осуществляется следующим образом: первые k элементов массива остаются без изменения, все элементы, начиная с (k+1)-го, необходимо сдвинуть на один назад, на место (k+1)-го элемента записываем значение x; увеличить количество элементов в массиве на единицу.

При решении задач с использованием двумерных массивов организуются вложенные циклы:

For i:=1 to m do begin
изменяется номер строки
For j:=1 to n do begin *изменяется номер столбца*
Запись элемента массива: a[i , j]

Главная диагональ i = j	**Побочная диагональ i + j = n+1**
a_{11} a_{12} a_{13} a_{14} a_{21} a_{22} a_{23} a_{24} a_{31} a_{32} a_{33} a_{34} a_{41} a_{42} a_{43} a_{44}	a_{11} a_{12} a_{13} a_{14} a_{21} a_{22} a_{23} a_{24} a_{31} a_{32} a_{33} a_{34} a_{41} a_{42} a_{43} a_{44}

Порядок выполнения работы:

1) Запустите программу Turbo Pascal.

2) *Ввести одномерный массив, состоящий из 10 элементов. Заменить отрицательные элементы на противоположные по знаку. Вывести полученный массив на экран.*

3) *Ввести одномерный массив, состоящий из n элементов. Найти и вывести на экран номера четных элементов.*

4) *Ввести одномерный массив, состоящий из m элементов. Найти количество положительных и отрицательных элементов в данном массиве.*

5) *Ввести одномерный массив, состоящий из n элементов. Удалите из массива третий элемент.*

6) *Ввести одномерный массив, состоящий из n элементов. Вставьте в массив число 100 после пятого элемента.*

7) *Ввести двумерный массив, состоящую из n*m элементов. Найдите сумму всех элементов.*

8) *Заполните двумерный массив размером N*N следующим образом:*

0 1 1 1 1 0
2 0 1 1 0 4
2 2 0 0 4 4
2 2 0 0 4 4
2 0 3 3 0 4
0 3 3 3 3 0.

9) Запишите полученные результаты в тетрадь. Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

10) Выйдите из программы Turbo Pascal.

ВАРИАНТ 1.
1. Найти сумму натуральных чисел от 10 до 150.
2. Арифметическая прогрессия задана формулой $a_n=3n+5$. Вывести первые 10 членов этой прогрессии и найти их сумму.
4. Найти произведение: $P = (1 + x)\cdot(3+2x)\cdot(5+3x)\cdot...$ (n множителей)

ВАРИАНТ 2.
1. Найти сумму натуральных чисел от 100 до 500.
2. Арифметическая прогрессия задана формулой $a_n=2n+3$. Вывести первые 15 членов этой прогрессии и найти их сумму.
4. Найти произведение: $P = (1 + 1/2)\cdot(3+1/3)\cdot(5+1/4)\cdot...$ (n множителей)

ВАРИАНТ 3.
1. Найти сумму натуральных чисел от 1 до 151.
2. Арифметическая прогрессия задана формулой $a_n=5n - 1$. Вывести первые 10 членов этой прогрессии и найти их сумму.
4. Найти произведение: $P = (1 + 2x)\cdot(1+3x)\cdot(1+4x)\cdot...$ (n множителей)

ВАРИАНТ 4.
1. Найти произведение натуральных чисел от 11 до 18.
2. Арифметическая прогрессия задана формулой $a_n=4n+3$. Вывести первые 15 членов этой прогрессии и найти их сумму.
4. Найти сумму: $S = x+2x+3x...$ (n слагаемых)

ВАРИАНТ 5.
1. Найти сумму натуральных чисел от 100 до 250.

2. Арифметическая прогрессия задана формулой $a_n=4n - 5$. Вывести первые 10 членов этой прогрессии и найти их сумму.
4. Найти произведение: $P = (1 + 1/2)\cdot(1+2/3)\cdot(1+3/4)\cdot\ldots$ (n множителей)

ВАРИАНТ 6.
1. Найти произведение натуральных чисел от 10 до 20.
2. Последовательность задана формулой $a_n=2n^2+3$. Вывести первые 15 членов этой прогрессии и найти их сумму.
4. Найти сумму:

$$S = \frac{1}{x+2} + \frac{2}{x+4} + \frac{3}{x+6} - \ldots \text{ (n слаг.)}$$

ВАРИАНТ 7.
1. Найти произведение натуральных чисел от 3 до 15.
2. Арифметическая прогрессия задана формулой $a_n=4n - 3$. Вывести первые 10 членов этой прогрессии и найти их сумму.
4. Найти сумму:

$$S = \frac{1}{x} + \frac{1}{2x} + \frac{1}{3x} - \ldots \text{ (n слаг.)}$$

ВАРИАНТ 8.
1. Найти произведение натуральных чисел от 7 до 21.
2. Арифметическая прогрессия задана формулой $a_n=7n+3$. Вывести первые 8 членов этой прогрессии и найти их сумму.
4. Найти сумму:

$$S = \frac{1}{2} + \frac{2}{5} + \frac{3}{8} + \ldots \text{ (n слаг.)}$$

ВАРИАНТ 9.
1. Найти сумму натуральных чисел от 20 до 80.
2. Арифметическая прогрессия задана формулой $a_n=3n+7$. Вывести первые 10 членов этой прогрессии и найти их сумму.
4. Найти произведение: $P = (1 + x)\cdot(1+x^2)\cdot(1+x^3)\cdot\ldots$ (n множителей)

ВАРИАНТ 10.
1. Найти сумму натуральных чисел от 200 до 250.
2. Арифметическая прогрессия задана формулой $a_n=6n-5$. Вывести первые 15 членов этой прогрессии и найти их сумму.
4. Найти произведение: $P = (1 + 1/2)\cdot(2+1/3)\cdot(3+1/4)\cdot\ldots$ (n множителей)

ВАРИАНТ 11.
1. Найти сумму натуральных нечетных чисел от 15 до 75.
2. Арифметическая прогрессия задана формулой $a_n=5n +4$. Вывести первые 10 членов этой прогрессии и найти их сумму.

4. Найти произведение: $P = (1 + 2x)\cdot(1+4x)\cdot(1+6x)\cdot\ldots$ (n множителей)

ВАРИАНТ 12.
1. Найти произведение натуральных чисел от 9 до 15.
2. Арифметическая прогрессия задана формулой $a_n=4n -1$. Вывести первые 15 членов этой прогрессии и найти их сумму.
4. Найти сумму: $S = x^2+2x^2+3x^2\ldots$ (n слагаемых)

Циклы DO…UNTIL и WHILE.

Контрольные вопросы:
1. Запишите общий вид оператора цикла с предусловием. Объясните механизм его работы.
2. Всегда ли выполняется цикл с предусловием? Поясните ответ.
3. Запишите общий вид оператора цикла с постусловием. Объясните механизм его работы.
4. Всегда ли выполняется цикл с постусловием? Поясните ответ.

Вариант 1
1. Начав тренировки, спортсмен пробежал X км. Каждый следующий день он увеличивал дневную норму на 10% от нормы предыдущего дня. Сколько дней должен тренироваться спортсмен, чтобы суммарный путь превысил S км?
2. Задана арифметическая прогрессия: -302; -287;...
Сколько членов прогрессии нужно сложить, чтобы сумма стала положительна?
3. Найти количество цифр, кратных трем, во введенном натуральном числе.

Вариант 2
1. Начав тренировки, спортсмен в первый день пробежал X км. Каждый следующий день он увеличивал дневную норму на 8% от нормы предыдущего дня. Через сколько дней спортсмен будет пробегать в день больше Y км?
2. Задана арифметическая прогрессия: -3,5; -3,1;...
Сколько чисел нужно сложить, чтобы сумма стала положительна?
3. Найти количество нечетных цифр во введенном натуральном числе.

Вариант 3
1. В 1985г. урожай ячменя составил X ц с га. В среднем каждые 2 года за счет применения передовых агротехнических приемов, урожай увеличивался на 5%. Определить, через сколько лет урожай достигнет Y ц с га.
2. Последовательность задана формулой $a_n=2^n +1$. Определить первый член последовательности, который больше 200 и его порядковый номер.
3. Найти количество четных цифр во введенном натуральном числе.

Вариант 4
1. Плотность воздуха ρ с высотой h убывает по закону $\rho=\rho_0 e^{-hz}$, где $\rho_0=1{,}29$ кг/м^3, $z=1{,}25\cdot10^{-4}$ 1/м. Определить на какой высоте плотность воздуха будет меньше 1 кг/м^3.

2. Последовательность задана формулой $a_n=2n+1/(n+1)$. Определить первый член последовательности, который больше заданного числа x, и его порядковый номер.
3. Определить количество нечетных цифр введенного натурального числа.

Вариант 5

1. В первый день туристы прошли X км. Каждый следующий день они проходили на 7% меньше от нормы предыдущего дня. Сколько дней потребуется туристам, чтобы пройти S км?
2. Задана арифметическая прогрессия: 30; 28,7;...
Сколько членов прогрессии нужно сложить, чтобы сумма стала отрицательна?
3. Определить, содержатся ли четные цифры во введенном натуральном числе.

Вариант 6

1. Начав тренировки, спортсмен в первый день проплыл X км. Каждый следующий день он увеличивал дневную норму на 8% от нормы предыдущего дня. Через сколько дней спортсмен будет проплывать в день больше Y км?
2. Задана арифметическая прогрессия: -1,5; -0,1;...
Сколько чисел нужно сложить, чтобы сумма стала положительна?
3. Определить, содержатся ли нечетные цифры во введенном натуральном числе.

Вариант 7

1. В 1992г. урожай пшеницы составил X ц с га. В среднем каждые 2 года за счет применения передовых агротехнических приемов, урожай увеличивался на 5%. Определить, через сколько лет урожай достигнет Y ц с га.
2. Последовательность задана формулой $a_n=2^{n+1}+10n$. Определить первый член последовательности, который больше 1000 и его порядковый номер.
3. Найти количество единиц во введенном натуральном числе.

Вариант 8

1. Первоначальный вклад в банк составил x руб. Через сколько лет вклад более чем в два раза превысит первоначальный, если годовой процент – y%.
2. Последовательность задана формулой $a_n=1/(n^2+1)$. Определить первый член последовательности, который меньше 0.005, и его порядковый номер.
3. Определить, содержит ли введенное натуральное число цифры, кратные 3.

Вариант 9

1. В 1990 г. завод производил продукции на X млн.руб. Каждый год в среднем производство увеличивалось на 6% от объема предыдущего года. В каком году объем производства превысит Y млн.руб.?
2. Задана последовательность: $a_1=1$, $a_n=a_{n-1}+n^2$
Какое наименьшее количество членов последовательности нужно сложить, чтобы сумма стала больше 1000?
3. Найти количество единиц и количество нулей во введенном натуральном числе.

Вариант 10

1. В 1985г. урожай гречихи составил X ц с га. В среднем каждые 2 года за счет применения передовых агротехнических приемов, урожай увеличивался на 4%. Определить, через сколько лет урожай достигнет Y ц с га.
2. Задана последовательность: $a_1=1$, $a_n=a_{n-1}+n$. Определить первый член последовательности, который больше 100 и его порядковый номер.
3. Найти количество троек во введенном натуральном числе.

Вариант 11

1. Автомобилист начал движение от пункта А со скоростью X км/ч и каждый час увеличивал скорость автомобиля на 7% от предыдущей. Успеет ли добраться за 5 часов до пункта В, который находится на расстоянии S км от А.
2. Задана арифметическая прогрессия: 10; 8,7;...
   Сколько членов прогрессии нужно сложить, чтобы сумма стала отрицательна?
3. Определить, содержатся ли цифры, кратные 3, во введенном натуральном числе.

Вариант 12

1. Туристы, путешествуя по реке, первый день проплыли X км. Каждый следующий день они увеличивали дневную норму на 5% от нормы предыдущего дня. Через сколько дней туристы проплывут в общей сложности больше Y км?
2. Задана арифметическая прогрессия: -1,2; -0,5;...
   Сколько чисел нужно сложить, чтобы сумма стала положительна?
3. Определить, содержатся нули во введенном натуральном числе.

Лабораторная работа №3. Проектирование и реализация подпрограмм.
Организация процедур

Цель: научиться составлять программы решения задач с использованием процедур.

Теоретические сведения:

В языке Паскаль имеется два вида подпрограмм - **процедуры** и **функции**.

Структура описания процедур до некоторой степени похожа на структуру Паскаль-программы: у них также имеются заголовок, раздел описаний и исполняемая часть. Раздел описаний содержит те же подразделы, что и раздел описаний программы: описания констант, типов, меток, процедур, функций, переменных. Исполняемая часть содержит собственно операторы процедур. Одна и та же подпрограмма может вызываться неоднократно, выполняя одни и те же действия с разными наборами входных данных. Параметры, использующиеся при записи текста подпрограммы в разделе описаний, называют **формальными**, а те, что используются при ее вызове - **фактическими**.

Формат описания процедуры имеет вид:

```
procedure имя процедуры (формальные параметры);
  раздел описаний процедуры
begin
  исполняемая часть процедуры
```

end;

Вызов процедуры производится оператором, имеющим следующий формат:

имя процедуры (список фактических параметров);

При вызове фактические параметры как бы подставляются вместо формальных, стоящих на тех же местах в заголовке. В стандарте языка Паскаль передача параметров может производиться двумя способами - по значению и по ссылке. Параметры, передаваемые по значению, называют **параметрами-значениями**, передаваемые по ссылке - **параметрами-переменными**. Последние отличаются тем, что в заголовке процедуры (функции) перед ними ставится служебное слово var. При первом способе (*передача по значению)* значения фактических параметров копируются в соответствующие формальные параметры. При изменении этих значений в ходе выполнения процедуры (функции) исходные данные (фактические параметры) измениться не могут. При втором способе (*передача по ссылке*) все изменения, происходящие в теле процедуры (функции) с формальными параметрами, приводят к немедленным аналогичным изменениям соответствующих им фактических параметров.

Имена, описанные в заголовке или разделе описаний процедуры называют **локальными** для этого блока. Имена, описанные в блоке, соответствующем всей программе, называют **глобальными**.

Порядок выполнения работы:

1) Запустите программу Turbo Pascal.

2) *Напишите программу, состоящую из трех процедур и основной программы. Первая процедура организует ввод двух целых чисел X и Y, вторая проверяет их сумму, третья выводит результат. Используйте эти процедуры в основной программе. Используйте X и Y как глобальные переменные.* Откомпилируйте программу. Проверьте правильность решения задачи.

3) *Найти площадь десятиугольника, вершины которого имеют координаты(x1,y1), (x2,y2) , (x3,y3) … (x10,y10).*
Откомпилируйте программу. Проверьте правильность решения задачи на любом примере.

4) *Вычислить разность двух простых дробей: a/b-c/d (a, b, c, d — натуральные числа). Результат получить в виде простой несократимой дроби e/f.*
Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

5) *Вычислить* $y = \frac{\sum_{i=1}^{7} i + \sum_{k=18}^{91} k}{\sum_{j=5}^{47} j}$ *Оформить вычисление суммы в виде подпрограмм.*
Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

6) *Дан одномерный массив, состоящий из 10 элементов. Используя процедуру, найдите максимальный и минимальный элементы этого массива.*
Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

7) Запишите полученные результаты в тетрадь. Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

8) Выйдите из программы Turbo Pascal.

Организация функций

Цель: научиться составлять программы решения задач с использованием функций.

Теоретические сведения:

Функции, это такие подпрограммы, результатом которых обязательно является некоторое значение. Описание функции во многом совпадает с описанием процедуры. Но если имя процедуры используется только для её вызова, то с именем функции, кроме того, связывается её результат. В описании функции заголовок выглядит следующим образом:

Function<имя>(<Параметры>): <Тип_возвращаемого_значения>;
<описания_переменных>;
Begin
<Тело
подпрограмм
ы>;
End;

Как и у процедуры, заголовок функции обязателен.

Функция предполагает обязательную передачу информации из подпрограммы в программу через имя функции. Поэтому раздел операторов обязательно должен содержать хотя бы один оператор, в котором имени функции присваивается значение результата. В противном случае функция не возвратит результат (вернее возвратит произвольный результат).

Порядок выполнения работы:

1) Запустите программу Turbo Pascal.

2) Определить значение выражения m (a,b, c) +m (b, c, d) +m (c, d, a) для вещественных a,b,c,d, где m(x,y,z) - функция определения минимального из трёх чисел.
Откомпилируйте программу. Проверьте правильность решения задачи.

3) Составить программу нахождения выражения $x^{10} + 2x^9 - 5x^7 + 4$*, используя функцию.*
Откомпилируйте программу. Проверьте правильность решения задачи на любом примере.

4) *Даны три квадратных уравнения ax^2+bx+c, dx^2+ex+f, px^2+qx+s. Сколько из них*
имеют вещественные корни?
Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

5) *Вычислить периметр треугольника по его координатам.*
Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

6) Запишите полученные результаты в тетрадь. Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

7) Выйдите из программы Turbo Pascal.

Использование рекурсивных подпрограмм.

Цель: научиться составлять программы решения задач с использованием рекурсивных процедур и функций.

Теоретические сведения:

Рекурсия - это способ описания функции или процессов через самих себя (когда процедура или функция сама себя вызывает). В Паскале можно пользоваться именами лишь тогда, когда в тексте программы этому предшествует их описание. Рекурсия является единственным исключением из этого правила. Имя рекурсивной функции можно использовать сразу же, не закончив его описания.

В Паскале возможно применения рекурсии в процедурах и функциях.

Порядок выполнения работы:

1) Запустите программу Turbo Pascal.

2) *Составить программу нахождения факториала, используя рекурсивную функцию.*
Откомпилируйте программу. Проверьте правильность решения задачи.
{Подсказка: Факториал $a_n=n!$ означает $n!=1*2*3*4*5*6*...*n$. а)$a_1=1$; б) $a_n=n*a_{n-1}$}

3) *Написать программу вычисления членов геометрической прогрессии.*
Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

{Подсказка: Геометрическая прогрессия: а) $a_1=a_0$; б) $a_n=a_{n-1}*q$. При $a_0=2$, $q=2$ имеем степенной ряд 2,4,8,16,32,...; }

4) *Написать программу вычисления членов арифметической прогрессии.*
Откомпилируйте программу. Проверьте правильность решения задачи на конкретном примере.

{Подсказка: Арифметическая прогрессия: а)$a_1=a_0$; б) $a_n=a_{n-1}+d$. При $a_0=1$, $d=1$ имеем натуральный ряд 1,2,3,... }

5) *Составить программу вычисления всех чисел Фибоначчи, используя рекурсивную функцию.*
Откомпилируйте программу. Проверьте правильность решения задачи на любом примере.

{Подсказка: Числа Фибоначчи. Один из наиболее ярких примеров применения рекурсии дают числа Фибоначчи. Они определяются следующим образом: $x_1=x_2=1$, $x_n=x_{n-1}+x_{n-2}$ при $n > 2$

Каждый элемент ряда Фибоначчи является суммой двух предшествующих элементов, т.е.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55,… }

8) Запишите полученные результаты в тетрадь. Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

9) Выйдите из программы Turbo Pascal.

Лабораторная работа №4. Описание и обработка файлов.

Работа с типизированными файлами последовательного доступа.

Цель: научиться работать с файлами в программе Turbo Pascal.

Теоретические сведения:

Файловый тип переменной — это структурированный тип, представляющий собой совокупность однотипных элементов, количество которых заранее не определено.

Структура описания файловой переменной:

Var <имя переменной>: File Of <тип элемента>;

где <тип элемента> может быть любым, кроме файлового.

Для того чтобы начать запись в файл, его следует открыть для записи. Это обеспечивает процедура Rewrite (FV); где FV — имя файловой переменной. При этом указатель устанавливается на начало файла. Если в файле есть информация, то она исчезает.

Запись в файл осуществляется процедурой Write (FV, V); где V — переменная того же типа, что и файл FV. Запись происходит туда, где установлен указатель.

Для чтения элементов файла с его начала следует открыть файл для чтения. Это делает процедура Reset (FV). В результате указатель устанавливается на начало файла. При этом вся информация в файле сохраняется.

Чтение из файла осуществляется процедурой Read (FV,V); где V — переменная того же типа, что и файл FV. Значение текущего элемента файла записывается в переменную V; указатель смещается к следующему элементу.

Для организации связи между файловой переменной и внешним файлом в Турбо Паскале используется процедура назначения: Assign (<имя файловой переменной>, <внешней файл>);

Здесь <внешней файла> — строковая величина;

Работа с файлом в программе завершается его закрытием с помощью процедуры

Close (<имя файловой, переменной>).
Для определения размера файла используется функция: FileSize (<имя файловой переменной>);

Порядок выполнения работы:

1) Запустите программу Turbo Pascal.

2) Составить программу*, которая создает файл, состоящий из N значений целого типа. Прочитать файл и вывести только четные элементы.* Откомпилируйте программу. Проверьте правильность решения задачи.

3) *Сформировать файл F, компонентами которого являются целые числа. Записать в файл C все четные числа файла F, а в файл N - все нечетные. Порядок следования чисел сохраняется.* Откомпилируйте программу. Проверьте правильность решения задачи.

4) *Заполнить файл последовательного доступа N действительными числами, полученными с помощью датчика случайных чисел. Найти максимальный элемент этого файла.* Откомпилируйте программу. Проверьте правильность решения задачи.

5) *Записать в файл f последовательного доступа N целых чисел. Получить в другом файле последовательного доступа все компоненты файла f, кроме тех, которые кратны K.* Откомпилируйте программу. Проверьте правильность решения задачи.

6) Запишите полученные результаты в тетрадь. Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

7) Выйдите из программы Turbo Pascal.

Работа с текстовыми файлами.

Цель: научиться работать с текстовыми файлами в программе Turbo Pascal.

Теоретические сведения:

Текстовый файл можно рассматривать как последовательность символов, разбитую на строки длиной от 0 до 256 символов. Для описания используется стандартный тип Text:

Var F: text; {F - файловая переменная}. Каждая строка завершается маркером конца строки.

Вызов **Read(F, Ww),** где Ww – переменная типа word, осуществляет чтение из файла F последовательности цифр, которая затем интерпретируется в число, значение которого и будет присвоено переменной Ww. В случае если вместо последовательности цифр идет любая другая последовательность символов, использование такого оператора приводит к ошибке выполнения программы. Открытие текстового файла можно произвести двумя стандартными способами:

- поставить в соответствие файловой переменной имя файла (процедура Assign), открыть новый текстовый файл (процедура Rewrite);

- поставить в соответствие файловой переменной имя файла (процедура Assign), открыть уже существующий файл (процедура Reset).

Текстовый файл в силу своей специфики во время работы допускает только один вид операции: чтение или запись. В связи с этим для работы с текстовыми файлами используется еще одна процедура открытия файла: **Append(var F : text);** Эта процедура открывает уже существующий файл и позиционирует указатель обработки на конец файла. После этого в текстовый файл можно только добавлять информацию, причем только в конец файла. Для обработки текстовых файлов используются процедуры Read и Write, обеспечивающие соответственно чтение и запись одной строки и более в текстовый файл. Использование специальных разделителей строк позволило ввести в состав языковых средств еще две процедуры: Readln, выполняющую те же действия, что и Read, и дополнительно – чтение до маркера конца строки и переход к новой строке; Writeln, обеспечивающую запись всех величин с обязательной установкой маркера конца строки в файл.

Процедура Read обеспечивает ввод данных общим потоком из одной строки, а Readln приводит к обязательному переходу к следующей строке текстового файла, т. е. ввод данных осуществляется из различных строк. Все вышесказанное в равной мере относится к операциям записи с помощью процедур Write и Writeln.

При организации операций ввода-вывода используются специальные языковые средства в виде функций **Eoln, Eof, SeekEoln, SeekEof.**

Функция **Eoln(var F: text)** возвращает булевское значение True, если текущая файловая позиция находится на маркере конца строки или вызов Eof(F) вернул значение True. Во всех остальных случаях значение функции будет False.

Функция **Eof(var F: text)** возвращает булевское значение True, если указатель конца файла находится сразу за последним компонентом, и False – в противном случае.

Порядок выполнения работы:

1) Запустите программу Turbo Pascal.

2) *Дано10 слов. Записать их в файл, расположив каждое слово на отдельной строке.*

Откомпилируйте программу. Проверьте правильность решения задачи.

3) *Имеется текстовый файл. Переписать в другой файл те его строки, в которых содержится более 30-ти символов.* Откомпилируйте программу. Проверьте правильность решения задачи.

4) *Имеется текстовый файл. Удалить из него первую строку, оканчивающуюся вопросительным знаком.* Откомпилируйте программу. Проверьте правильность решения задачи.

5) *Имеется текстовый файл. Найти количество строк, начинающихся на букву «А». Напечатать самую длинную строку (если таких строк несколько, вывести первую из них).* Откомпилируйте программу. Проверьте правильность решения задачи.

6) Запишите полученные результаты в тетрадь. Оформите отчет о проделанной работе, который должен содержать тему, цель работы, формулировки задач с решениями.

7) Выйдите из программы Turbo Pascal.

Задача. *Дан текстовый файл f. Получить все его строки, содержащие более 60 символов.*

Для решения задачи:

- *формируем тело программы и описываем переменные;*
- *открываем файл для чтения;*
- *в цикле, до тех пор пока не дойдем до конца файла, читаем из файла строки и если их длина более 60 символов, то выводим на экран.*

```
program p55;
uses crt;
var
   f : text;
   a : string;
begin
   clrscr;
   assign(f,'text');
   reset(f);
   repeat
         readln(f,a);
   if (length(a)>60) then
         begin
               write(a);
               if length(a)<>80 then writeln;
     end;
   until eof(f);
   close(f);
   readln;
end.
```

Задача. *Даны два текстовых файла f и g. Определить, совпадают ли строки файла f со строками файла g. Если же нет, то получить номер первой строки и позицию первого символа в этой строке, в которых файлы f и g отличаются между собой.*

Для решения задачи:

- *формируем тело программы и описываем переменные;*
- *открываем файлы f и g для чтения;*
- *в цикле считываем строки из файлов, до тех пор, пока не дойдем до конца файла f или файла g, и совпадают строки этих файлов;*
- *в этом же цикле определяем позицию различий строк;*

- *выводим на экран результат проверки строк файлов.*

```
program p56;
uses crt;
var
   f, g : text;
   i, j : integer;
   c1, c2: char;
   cb : boolean;
begin
   clrscr;
   assign(f,'catalog1.txt');
   {$i-}
   reset (f);
   assign(g,'catalog2.txt');
   {$i-}
   reset(g);
   i := 1;
   j := 1;
   cb := false;
   repeat
         read(f,c1);
         read(g,c2);
         if c1=chr (10) then read(f,c1);
         if c2=chr (10) then read(g,c2);
         if c1<>c2 then cb := true;
         if (c1=chr(13)) and not cb then
         begin
               i := 1;
               j := j + 1
         end
         else
               i := i + 1
   until eof(f) or eof(g) or cb;
   close(g);
   close(f);
   if cb then
   begin
         textcolor(14);
         writeln('Найдено различие между файлами на позиции:');
         writeln;
         textcolor(11);
         write('Строка: ');
         textcolor(15);
         writeln(j);
         textcolor(11);
         write('Столбец: ');
```

```
            textcolor(15);
            writeln(i-1);
        end
        else
        begin
            textcolor(12);
            writeln('Различий между файлами не найдено.')
        end;
        textcolor(15);
        repeat until keypressed
    end.
```

Задача. *Описать процедуру Lines(FileIn, FileOut), которая считывает из входного файла FileIn литеры и построчно записывает их в текстовый файл FileOut, вставляя в начало каждой строки ее порядковый номер (он должен занимать 4 позиции и пробел) и в конец строки число, показывающее количество символов в этой строке.*

Для решения задачи:

- *формируем тело программы и описываем переменные;*
- *описываем процедуру lines, в которой открываем файл filein для чтения, а файл fileout для записи;*
- *в цикле считываем из файла filein все строки, добавляем порядковый номер строки и записываем в файл fileout;*
- *в основной программе в файл filein записываем несколько строк;*
- *вызываем процедуру lines.*

```
program p57;
var
   filein, fileout : text;

procedure lines(filein,fileout : text); {Начало процедуры}
var
   s : string[255];
   n : integer;
begin
   reset(filein);
   rewrite(fileout);
   n := 1;
   while(not eof(filein)) do
   begin
        readln(filein,s);
        write(fileout,n:4);
        write(fileout,' ');
        write(fileout,s);
        write(fileout,' ');
        writeln(fileout,length(s));
```

```
            n := n + 1;
        end;
        close(fileout);
    end;    {Конец процедуры}

    begin   {Начало программы}
        assign(filein,'filein.txt');
        assign(fileout,'fileout.txt');
        rewrite(filein);
        writeln(filein,'текстовый файл');
        writeln(filein,'с литерами');
        close(filein);
        lines(filein,fileout);
        readln;
    end.    {Конец программы}
```

Задание.

1. Записать все символы латинского алфавита в алфавитном порядке построчно в текстовый файл *f*.
2. Подсчитать количество строк в текстовом файле *f*.
3. Вывести на экран все непустые строки текстового файла *f*.
4. Подсчитать количество символов в текстовом файле *f*.
5. Распечатать текстовый файл *f* так, что бы каждое предложение начиналось с новой строки.
6. Дан текстовый файл *f*. Получить самую длинную строку файла. Если в файле имеется несколько строк с наибольшей длинной, то вывести все такие строки.
7. Посчитайте количество знаков препинания в текстовом файле *f*.
8. Дан текстовый файл *f*. Получить все его строки, содержащие более 60 символов.
9. Дан текстовый файл *f*. Получить в файле *h* все строки файла, в которых встречается слово *s*.
10. Дан текстовый файл *f*. Определить, сколько в файле *f* имеется слов, состоящих из одного, двух, трех и т.п. символов (до самого наибольшего по длине слова).
11. Дан текстовый файл *f*. Записать в перевернутом виде строки файла *f* в файл *g*. Порядок строк в файле *g* должен совпадать с порядком исходных строк в файле *f*.
12. Дан текстовый файл *FileIn*, записанный в кодировке *Windows*. Составить процедуру *ANSI_ASCII*(*FileIn*, *FileOut*) преобразования текстового файла в кодировку *DOS*.
13. Описать процедуру *Lines*(*FileIn*, *FileOut*), которая сортирует строки исходного файла по длине, т.е. сохраняет в файле *FileOut* строки файла *FileIn* в порядке увеличения их длины, от самой короткой до самой длинной.
14. Описать процедуру *Lines*(*FileIn*, *FileOut*), которая считывает из входного файла *FileIn* литеры и построчно записывает их в текстовый файл *FileOut*, вставляя в начало каждой строки ее порядковый номер (он должен занимать 4 позиции и пробел) и в конец строки число, показывающее количество символов в этой строке.

15. Пусть текстовый файл *FileIn* содержит непустые строки, состоящие из литер латинского алфавита. Описать процедуру *Lines*(*FileIn*, *FileOut*), которая в файл *FileOut* записывает строки, содержащие одинаковые литеры.
16. Дан текстовый файл *FileIn*. Составить процедуру *RWord*(*FileIn*, *FileOut*), которая удаляет в файле слова и фразы выделенные кавычками.
17. Преобразовать текстовый файл *FileIn* с помощью составленной процедуры *DelWord*(*FileIn*, *FileOut*) удалите слова, содержащие латинские символы.
18. Дан текстовый файл *f*, в котором содержатся сведения о сотрудниках некоторого учреждения в формате: Фамилия, Имя, Отчество. Записать эти сведения в файл g в формате: Фамилия, И.О.
19. Дан текстовый файл *f*. Удалить из файла все слова длиной более 10 символов.
20. Дан текстовый файл *f*, в котором содержатся сведения о зарплате сотрудников некоторого учреждения в формате: Фамилия Имя Отчество : ####.##. Подсчитайте общую сумму зарплат всех работников и распечатайте сотрудника с наибольшей зарплатой.

Ообразец оформления лабораторной работы

Задание 1. Вычислить сумму цифр введенного натурального двухзначного числа.

Используемые переменные: *n* –вводимое натуральное двухзначное число, *a* – первая цифра числа (десятки), *b* – вторая цифра числа (единицы)

Решение:

```
Program  pr1;
Var
        n, a, b: integer;
Begin
       write('n=   ');    readln(n);
       a:=n div 10;
       b:=n mod 10;
      writeln('сумма = ', a+b);
End.
```

Результат выполнения программы:

1. n=48
   сумма=12
2. n=52
   сумма=7

Ручная трассировка:

n=48
a=48 div 10=4
b= 48 mod 10= 8
сумма = 4+8=12

Задание 2. Вводится вещественное число *a*. Не пользуясь никакими арифметическими операциями, кроме сложения, получить $7a$ за четыре операции.

Используемые переменные: *a* –вводимое число, *b,c,d* –вспомогательные переменные

Решение:

```
Program pr2;
Var
  a,b,c,d:real;
Begin
  write('введите a ');
  readln (a);
  b:=a+a;
  c:=b+b;
  d:=b+c;
  a:=d+a;
  writeln('7a=',a:8:2);
```

```
  readln;
End.
```

Результат выполнения программы:

1. введите а 2
   7а= 14.00
2. введите а 3
   7а= 21.00

Ручная трассировка:

a=2
b:=2+2=4;
c:=4+4=8;
d:=4+8=12;
a:=12+2=14;
7a=14.00

Учебно-методическое обеспечение

Методические рекомендации преподавателю

Согласно существующему государственному образовательному стандарту специальности и других нормативных документов целесообразно разработать матрицу наиболее предпочтительных методов обучения и форм самостоятельной работы студентов, адекватных видам лекционных и лабораторных занятий.

Необходимо предусмотреть развитие форм самостоятельной работы, выводя студентов к завершению изучения учебной дисциплины на её высший уровень.

Пакет заданий для самостоятельной работы следует выдавать в начале семестра, определив предельные сроки их выполнения и сдачи.

Организуя самостоятельную работу, необходимо постоянно обучать студентов методам такой работы.

Вузовская лекция — главное звено дидактического цикла обучения. Её цель — формирование у студентов ориентировочной основы для последующего усвоения материала методом самостоятельной работы. Содержание лекции должно отвечать следующим дидактическим требованиям:

- изложение материала от простого к сложному, от известного к неизвестному;
- логичность, четкость и ясность в изложении материала;
- возможность проблемного изложения, дискуссии, диалога с целью активизации деятельности студентов;
- опора смысловой части лекции на подлинные факты, события, явления, статистические данные;
- тесная связь теоретических положений и выводов с практикой и будущей профессиональной деятельностью студентов.

Преподаватель, читающий лекционные курсы в вузе, должен знать существующие в педагогической науке и используемые на практике варианты лекций, их дидактические и воспитывающие возможности, а также их методическое место в структуре процесса обучения.

Лабораторные работы сопровождают и поддерживают лекционный курс.

При проведении промежуточной и итоговой аттестации студентов важно всегда помнить, что систематичность, объективность, аргументированность - главные принципы, на которых основаны контроль и оценка знаний студентов. Проверка, контроль и оценка знаний студента, требуют учета его индивидуального стиля в осуществлении учебной деятельности. Знание критериев оценки знаний обязательно для преподавателя и студента.

Методические указания студентам

Изучение программы курса. На лекциях преподаватель рассматривает вопросы программы курса, составленной в соответствии с государственным образовательным стандартом. Из-за недостаточного количества аудиторных часов некоторые темы не удается осветить в полном объеме, поэтому преподаватель, по своему усмотрению, некоторые вопросы выносит на самостоятельную работу студентов, рекомендуя ту или иную литературу.

Кроме этого, для лучшего освоения материала и систематизации знаний по дисциплине, необходимо постоянно разбирать материалы лекций по конспектам и

учебным пособиям. В случае необходимости обращаться к преподавателю за консультацией. Полный список литературы по дисциплине приведен в пункте 8.1. «Учебно-методическое обеспечение дисциплины».

В целом, на один час аудиторных занятий отводится один час самостоятельной работы.

Контрольные работы. После изучения некоторых разделов практической части курса «Программирование» проводятся контрольные аудиторные работы. Для успешного их написания необходима определенная подготовка. Готовиться к контрольным работам нужно по материалам лекций и рекомендованной литературы. Обычно, контрольная работа имеет 4-6 вариантов.

Лабораторные работы. При изучении курса «Программирование» необходимо выполнять и вовремя сдавать преподавателю индивидуальные лабораторные работы.

Коллоквиум - это устный теоретический опрос. Он проводится в середине семестра с целью проверки понимания и усвоения теоретического и практического материала курса, а также для проверки самостоятельной работы студентов по вопросам программы курса.

При подготовке к коллоквиуму ориентируйтесь на лекции и рекомендованную основную литературу. Дополнительная литература также может помочь при подготовке к теоретическому опросу.

В каждом семестре предполагается проведение трех коллоквиумов.

Тематика самостоятельных работ

I-семестр

Раздел А. Основы классического программирования

1. Платформы персональных компьютеров.
2. Классификация и структура языков программирования.
3. Способы описания синтаксиса языков программирования.
4. Специализированный алгоритмический язык и его конструкции.
5. Структура и компоненты интегрированных систем программирования.
6. Стандартные модули систем программирования, их структура и компоненты.

II-семестр

Раздел Б. Архитектура МП, языки ассемблер и C++

1. Процессоры Intel Pentium в современных разработках.
2. Особенности многоядерных микропроцессоров.
3. Операции со строками и массивами на ассемблере.
4. Структура и способы создания .exe-файлов.
5. Способы определения пользовательских типов в C++-программах.
6. Специализированные функции обработки файлов языка C++.

Литература

1. Симанович С. и др. Специальная информатика: универсальный курс. - М.: АСТ-ПРЕСС, 2000. – 480 с.
2. Вирт Н. Алгоритмы + структуры данных= программы. - М.: Мир, 1985. - 405 с.

3. Бройдо В., Ильина О. Архитектура ЭВМ и систем. – СПб.: Питер, 2006.-718 с.
4. Жмакин А. Архитектура ЭВМ. – СПб.: ВХБ, 2006. – 320 с.
5. Информатика. Учебник под ред. Н. Макаровой. –М.: ФиС, 2000. -284 с.
6. Фаронов В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. –М.: Нолидж, 1998. - 616 с.
7. Сурков Д. и др. Программирования в среде Borland Pascal для Windows. – Минск: Высшая школа, 1996. – 426 с.
8. Абрамов А. и др. Задачи по программированию. - Наука, 1988. – 224 с.
9. Нортон П. Программно-аппаратная организация IBM PC. – М.: Мир, 1996. – 327 с.
10. Абель П. Язык ассемблера для IBM PC и программирования. –М.: Высшая школа, 1992. – 447 с.
11. Магда Ю. Ассемблер для процессоров Intel Pentium. - СПб.: Питер, 2006. – 410 с.
12. Скляров В. Программирование на языке ассемблера. - Минск: Высшая школа, 1999. – 192 с.
13. Юров В. Assembler: практикум. – СПб.: Питер, 2002. – 400 с.
14. Страуструп Б. Язык программирования C++. Специальное издание. –М.: Бином–Пресс, 2006. – 1104 с.
15. Павловская Т. C++. Программирование на языке высокого уровня. –СПб.: Питер, 2005. – 461 с.
16. Культин Н. C++ Builder в задачах и примерах. – СПб.: ВХБ, 2005. - 336 с.

Электронные ресурсы

1. http://www.informatica.ru
2. http://www.intuit.ru
3. http://www.pascaldax.ru
4. http://www.cppstudio.ru
5. http://www.compteacher.ru/programming
6. http://www.citforum.ru/programming

Контрольные и тестовые материалы

1. Этапы решения задач на ЭВМ. Алгоритм. Свойства алгоритмов.
2. Алгоритмическая конструкция ветвления. Типы. Правила отображения.
3. Алгоритмическая конструкция цикла. Типы. Правила отображения.
4. Основные конструкции языка. Операторы.
5. Алфавит. Типы данных языка программирования.
6. Простые типы данных. Целочисленный, символьный, вещественный типы.
7. Структурированные типы данных. Массив. Строка. Структура.
8. Константы, переменные, операции. Определения и свойства, объявление.
9. Управляющие конструкции. Безусловные конструкции. Операторы.
10. Управляющие конструкции. Условные конструкции. Операторы.
11. Управляющие конструкции. Конструкция выбор. Операторы.
12. Управляющие конструкции. Циклические конструкции. Цикл с предусловием. Порядок работы.

13. Управляющие конструкции. Циклические конструкции. Цикл с постусловием. Порядок работы.
14. Управляющие конструкции. Циклические конструкции. Цикл со щетчиком. Порядок работы.
15. Функции. Объявление, определение и обращение к функции. Передача данных в функцию и из функции. Примеры программирования.
16. Стандартные функции ввода-вывода. Свойства, подключаемые библиотеки, форматированный вывод. Примеры.
17. Математические функции. Свойства, подключаемые библиотеки.
18. Функции работы со строками. Свойства, подключаемые библиотеки.
19. Файлы. Указатель на файл. Режимы открытия файла. Функции для открытия и закрытия файла. Примеры программирования.
20. Файлы. Указатель на файл. Функции для чтения и записи данных в файл. Примеры программирования.
21. Блок-схема алгоритма описания числа. Основные особенности алгоритма.
22. Программа для описания числа. Основные особенности.
23. Блок-схема алгоритма определения значения функции. Основные особенности алгоритма.
24. Программа для определения значения функции. Основные особенности.
25. Блок-схема алгоритма поиска сумы ряда. Основные особенности алгоритма.
26. Программа для поиска сумы ряда. Основные особенности.
27. Блок-схема алгоритма ввода-вывода двумерного и одномерного массива. Основные особенности алгоритма.
28. Программа для ввода-вывода двумерного и одномерного массива. Основные особенности.
29. Блок-схема алгоритма поиска минимального (максимального) элемента в массиве. Основные особенности алгоритма.
30. Программа для поиска минимального (максимального) элемента в массиве. Основные особенности.
31. Блок-схема алгоритма поиска локальных максимумов (минимумов) и их количества. Основные особенности алгоритма.
32. Программа для поиска локальных максимумов (минимумов) и их количества. Основные особенности.
33. Сортировка массивов. Сортировка простым выбором. Программа.
34. Сортировка массивов. Сортировка простой перестановкой. Программа.
35. Сортировка массивов. Сортировка пузырьковым методом. Программа.
36. Объектно - ориентированное программирование. Основные положения.
37. Библиотека Компонентов. Общие свойства компонентов.

Задачи.

1. Даны координаты двух полей шахматной доски (4 целых числа от 1 до 8). Если король может перемещаться с одного поля на другое, то вывести True, иначе – False.
2. Из трех данных чисел выбрать наименьшее.
3. Дано целое число в диапазоне 1..10. Вывести словесное описание числа.

4. Даны 2 целых числа A и B. Вывести все целые числа, расположенные между ними.
5. Отсортировать массив в порядке убывания элементов.
6. Дан целочисленный массив размером 3 x 3. Вычислить сумму значений главной диагонали.
7. Определить, является ли данная строка числом и какого типа (целое или вещественное).
8. Написать функцию, рассчитывающую факториал.
9. Создать текстовый файл и записать в него строку, введенную пользователем.
10. Прочитать строку из файла и вывести ее на экран.

Контрольное тестирование

1. Какие операторы записаны правильно?

а) g:=g;
б) M:=2xM;
в) k:=232R;
г) s:=l:=u

2. В каком разделе программы на языке Паскаль можно использовать процедуру вывода
Write?

а) В разделе выполняемых операторов;
б) В разделе описания констант;
в) В разделе описания переменных.

3. Какие процедуры ввода записаны без ошибок?

а) Read (S, 67);
б) Read (T);
в) Read T;
г) Read (a, c);
д) Read ('a');
е) Read ('Введите a',
а).

4. Что будет выведено на экран в результате выполнения программы?

```
Program S;
Var x, y: Integer;
Begin x:=2;  y:=3; x:=x*x; y := y * y;x:=x + y;
WriteLn ('x =', x);
End.
```

а) *x* = 5;
б) c=13;
в) 13;
г) 5.

5. Какая команда позволяет увидеть результат выполнения программы?

а) Debug/Outpu t;
б) File/Save;
в) File/New;
г) Run/Run.

6. Какое расширение должен иметь файл с исходным текстом программы на Паскале?

а) B AS;
б) BAK;
в) C;
г) PAS.

7. Какие имена программ на Паскале допустимы (при условии, что имя программы не совпадает с именем файла)?

а) 112233;
б) MyProgl;
в) Мойрг!;
г) MyPrograml.

8. Какие элементы языка Турбо-Паскаль могут быть обозначены идентификаторами?

а) Переменные;
б) Операторы;
в) Константы

9. Какие из перечисленных типов данных относятся к простым?

а) Логический;
б) Символьные;
в) Целые;
г) Строковые.

10. Какие из представленных здесь констант относятся к целым?

а) '555'; б) 1188; в) 1.0; г) -345.

11. Операторы в программе на языке Pascal отделяются друг от друга:

а) двоеточием;
б) точкой с запятой;
в) пробелом;
г) запятой.

12. Какой оператор позволяет выводить пустую строку в документе?

а) writeln ; б) print; в) output; г) readln.

13. Укажите условие выбора чисел, кратных 3, но не кратных 6

а) (X mod 3 =0) AND (X mod 6 <>O);
б) (X/3 =0) OR (X / 6≠0));
в) (X mod 3 =0) OR (X mod 6 <>0);
г) (X mod 6 <>0).

14. Какое значение примет переменная С в результате выполнения программы:

```
VAR A, B, C : INTEGER;
BEGIN
A:=-4;  B:=A*3-6; IF B>2*A THEN A := 2;
IF B<2*A THEN A := 5;  IF B=2*A THEN A := B - A;
C :=A*B+A-B; WRITELN ('C=', C);
END.
```

а) 29; б) 21; в) 32; г) 12.

15. Какой из операторов организует в программе множественный выбор?

а) If условие Then действие1 Else действие2;
б) If условие Then действие;
в) Case ключ of ;
г) goto n.

16. С помощью какого оператора осуществляется переход к метке?

а) If условие Then действие;
б) Case ключ of;
в) goto n;
г) If условие Then действие1 Else действие2.

17. При каком значении X при исполнении программы будет получен ответ «ДА»:

```
Program T21;
Var X: Integer;
Begin
Readln(X);  If X MOD 2=0 Then Writeln('ДА') Else Writeln('НЕТ');
End.
```

а) 3; б) 15; в) 4; г) 7.

18. Какое значение будет принимать переменная Y после выполнения фрагментов программы?

```
Var Y, X: integer;
Begin Y: = l;  X: = 10;
Repeat
Y:=X*X;          X:=X-2;
Until X<=0;
Writeln ('У=',У);     End.
```

а) 4; б) 0; в) 2; г) 100.

19. Какие из приведенных операторов организуют цикл со счетчиком?

а) For i := nl downto n2 do оператор;
б) repeat <операторы > until условие;
в) while условие do <оператор>;
г) for i = nl to n2 do оператор.

20. Какое значение будет принимать переменная Y после выполнения фрагментов программы?

y:=0; FOR x := 1 TO 5 do У := У * x; Writeln (y);

а) 0; б) 12345; в) 120; г) 00000.

21. Тело цикла - это...

а) группа команд, не входящих в циклическую структуру;
б) группа команд, повторяющихся некоторое число раз;
в) произвольная группа команд;
г) команды, заключенные в операторные скобки.

22. Какой из перечисленных заголовков циклов не содержит ошибок?

а) FOR X:=3 TO 12 DO;
б) FOR К = 1 TO 5.5 DO;
в) FOR K = 2DOWNTO 10 DO;
г) FOR J := 10 TO -2 DO.

23. Что определяет индекс массива?

а) Индекс определяет положение элемента массива данных относительно его конца;

б) Индекс определяет положение элемента массива данных относительно друг друга;

в) Индекс определяет положение элемента массива данных относительно его начала.

24. Укажите правильное описание массива

а) Var a:array[1..1000] of integer;

б) Var A: ARRAY [1..50 OF REAL];

в) Var A, B, C: ARRAY [1 ..50] OF REAL.

25. Что производит следующий фрагмент программы?

```
s:=0; for i:=l to n do
for j:=l to n do s: =s+a[i, j]; end;
writeln('s=',s);
```

а) Находит сумму элементов каждой строки и выводит их на экран;

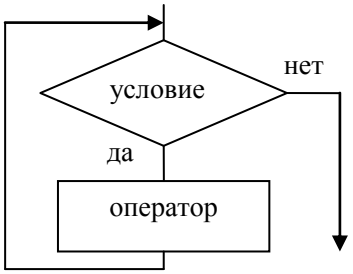
б) Находит сумму всех элементов двумерного массива и сумму элементов каждой строки и выводит их на экран;

в) Находит сумму всех элементов двумерного массива и выводит их на экран.

АЛГОРИТМЫ

1. Задание

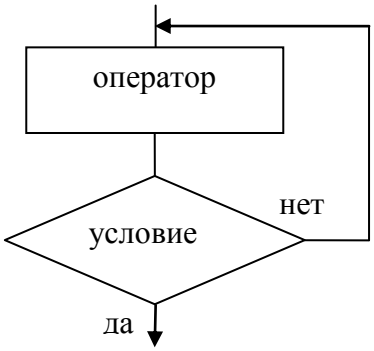
Укажите вид циклической кострукции, изображенной на рисунке



- ☑ цикл с предусловием
- ☐ цикл с постусловием
- ☐ цикл с параметром

2. Задание

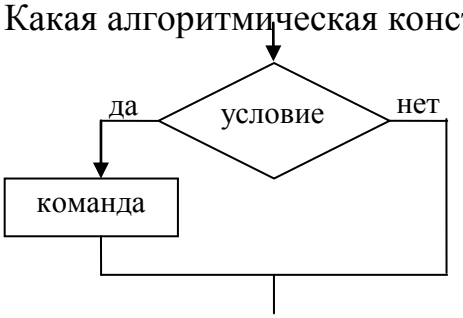
Укажите вид циклической конструкции, изображенной на рисунке



- ☑ цикл с постусловием
- ☐ цикл с предусловием
- ☐ цикл с параметром

3. Задание

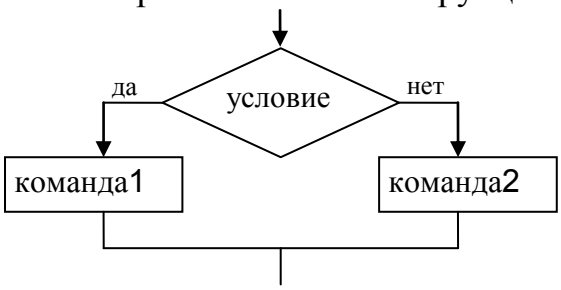
Какая алгоритмическая конструкция изображена на рисунке



- ☑ неполное ветвление
- ☐ полное ветвление
- ☐ цикл с предусловием
- ☐ цикл с параметром

4. Задание

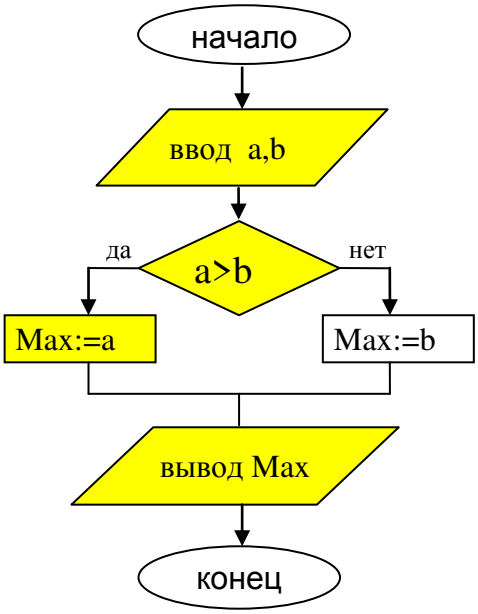
Какая алгоритмическая конструкция изображена на рисунке



- ☐ неполное ветвление
- ☑ полное ветвление
- ☐ цикл с предусловием
- ☐ цикл с параметром

5. Задание

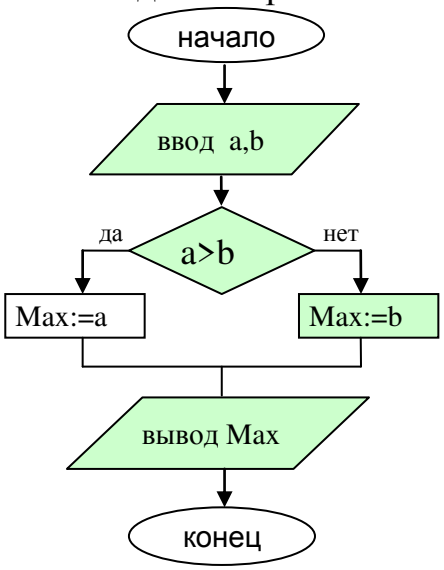
При каких значениях a и b выполнение алгоритма пойдет по левой ветви (цветом выделены выполняемые команды)



- ☑ a=7 b=4
- ☐ a=7 b=7
- ☐ a=5 b=8

6. Задание

При каких значениях a и b выполнение алгоритма пойдет по правой ветви

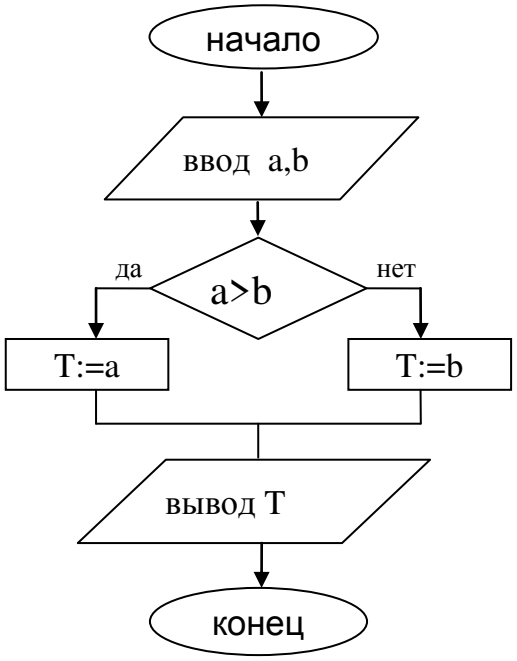


(цветом выделены выполняемые команды)

- ☐ a=7 b=4
- ☑ a=7 b=7
- ☑ a=5 b=8
- ☐ a=6 b=1

7. Задание

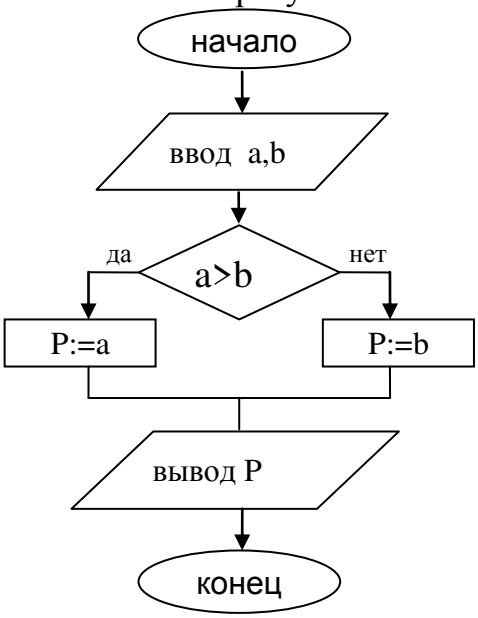
При a=18 и b=25 переменная T примет значение...



Правильные варианты ответа: 25;

8. Задание

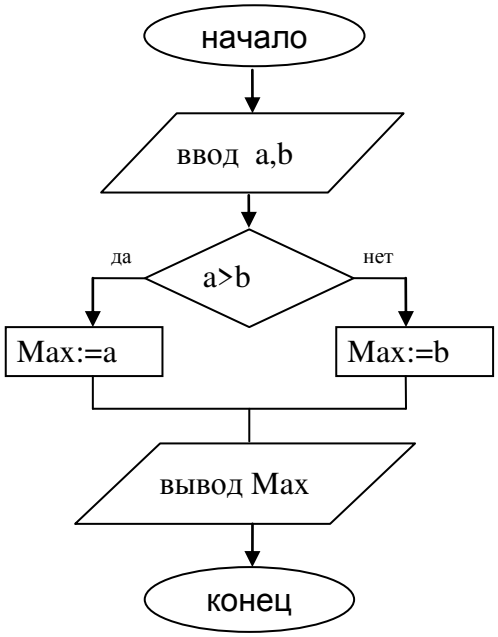
При a=17 и b=11 результат выполнения алгоритма...



Правильные варианты ответа: 17;

9. Задание

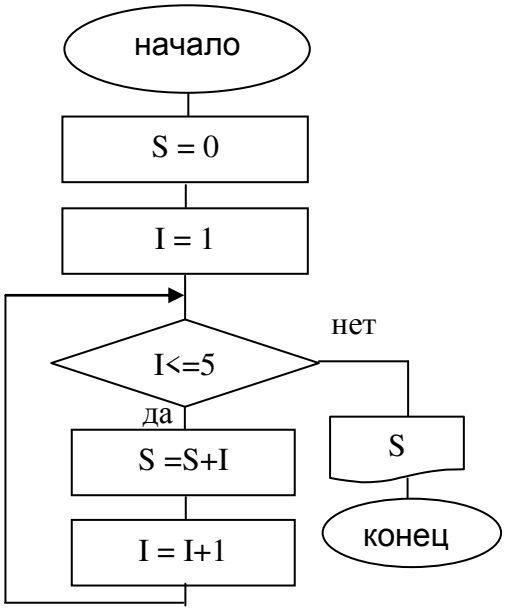
При a=8 и b=8 результат выполнения алгоритма...



Правильные варианты ответа: 8;

10. Задание

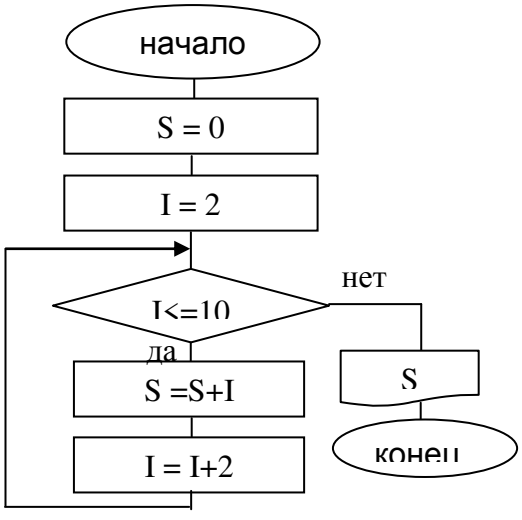
В результате выполнения алгоритма переменная S примет значение...



Правильные варианты ответа: 15;

11. Задание

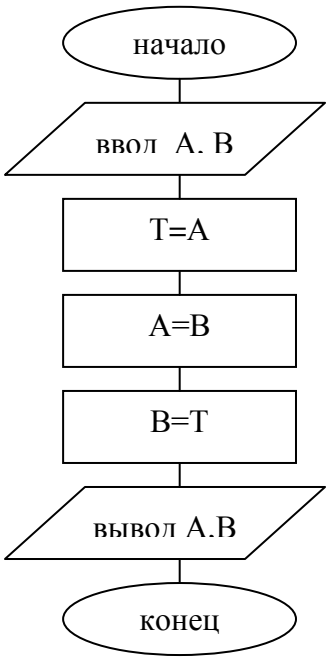
В результате выполнения алгоритма переменная S примет значение...



Правильные варианты ответа: 30;

12. Задание

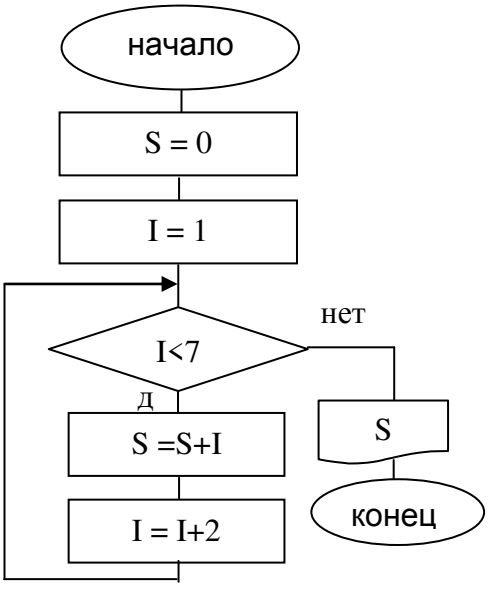
При A=5 и B=17 в результате выполнения алгоритма переменные A и B примут значения



- ☑ A= 17 B=5
- ☐ A= 5 B=5
- ☐ A= 5 B=17
- ☐ A= 17 B=17

13. Задание

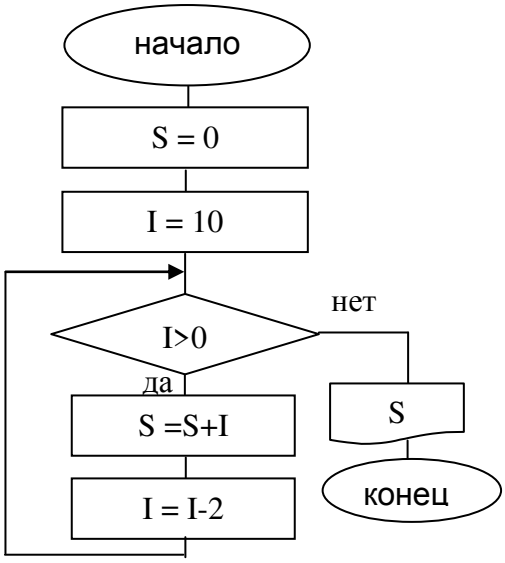
В результате выполнения алгоритма переменная S примет значение...



Правильные варианты ответа: 9;

14. Задание

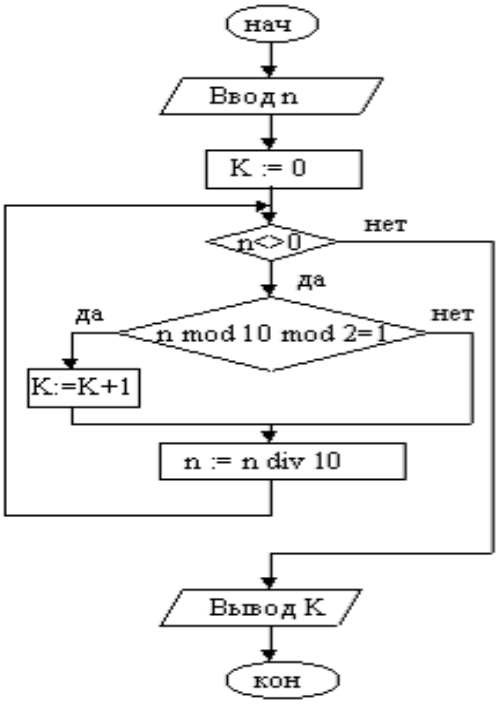
В результате выполнения алгоритма переменная S примет значение...



Правильные варианты ответа: 30;

15. Задание {{ 15 }} № 197

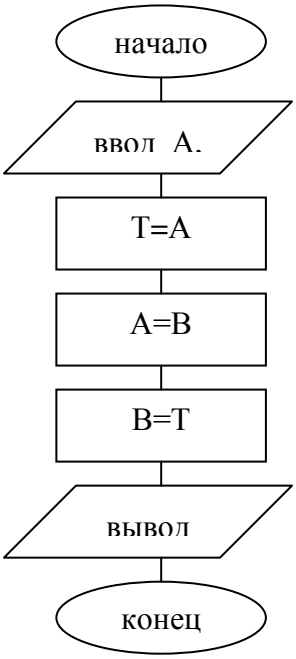
При n=3257 в результате выполнения алгоритма переменная K примет значение...



Правильные варианты ответа: 3;

16. Задание

При A=15 и B=7 в результате выполнения алгоритма переменные A и B примут значения



- ☐ A= 15 B=7
- ☐ A= 15 B=15
- ☐ A= 7 B=7
- ☑ A= 7 B=15

17. Задание

Дополните

Алгоритм, в котором действия выполняются последовательно сверху вниз от начала до конца называется...

Правильные варианты ответа: линейный; линейным;

18. Задание

Отметьте правильные ответы

Основными способами записи алгоритма являются...

- ☑ словесно-формульный
- ☑ графический
- ☑ на алгоритмическом языке
- ☐ знаковый
- ☐ числовой

19. Задание

Отметьте правильный ответ

Свойство алгоритма, определяющее, что решение задачи должно быть представлено в виде последовательности отдельных действий, называется....

- ☑ дискретностью
- ☐ определенностью
- ☐ результативностью
- ☐ массовостью
- ☐ понятностью

20. Задание

Отметьте правильный ответ

Свойство алгоритма, определяющее, что каждый шаг алгоритма должен восприниматься однозначно и не допускать произвольной трактовки, называется....

- ☐ дискретностью
- ☑ определенностью
- ☐ результативностью
- ☐ массовостью
- ☐ понятностью

21. Задание

Отметьте правильный ответ

Свойство алгоритма, определяющее, что решение задачи должно быть получено за определенное конечное число шагов, называется....

- ☐ дискретностью
- ☐ определенностью
- ☑ результативностью
- ☐ массовостью
- ☐ понятностью

22. Задание

Отметьте правильный ответ

Свойство алгоритма, определяющее, что алгоритм должен решать некоторый класс задач, отличающихся исходными данными, называется....

- ☐ дискретностью
- ☐ определенностью
- ☐ результативностью
- ☑ массовостью
- ☐ понятностью

23. Задание

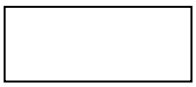
Отметьте правильный ответ

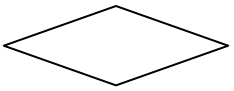
Алгоритм - это...

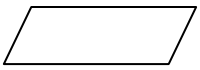
- ☑ четко определенная последовательность действий, которые необходимо выполнить для решения задач.
- ☐ набор данных
- ☐ результат решения задачи
- ☐ поиск решения задачи
- ☐ набор данных, которые необходимо задать для решения задачи

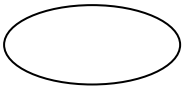
24. Задание

Установите соответствие между элементами групп

Действие 

Условие 

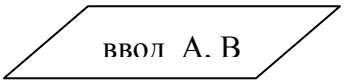
Ввод/вывод данных 

начало/конец алгоритма 

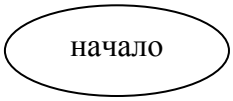
вспомогательный алгоритм

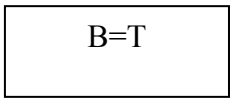
25. Задание

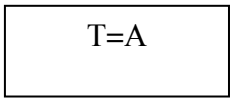
Установите правильную последовательность блоков блок-схемы для решения задачи обмена значений переменных

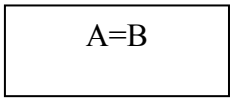
6: 

2: 

1: 

5: 

3: 

4: 

7: 

ОПРЕРАТОРЫ ПРИСВАИВАНИЯ, ВВОДА И ВЫВОДА

46. Задание

Переменная D после выполнения команд: D:=3; D:=D*D; D:=D*D; примет значение...

Правильные варианты ответа: 81;

47. Задание

Результат выполнения команд: C:=14; C:=C mod 3; Writeln(C);

Правильные варианты ответа: 2;

48. Задание

Укажите правильные формы записи оператора вывода

- ☑ write (x, y);
- ☑ write (x, x+1, x+2);
- ☐ write (x; y; z);
- ☑ write (x:7:3);
- ☐ write (x-2; 2);

49. Задание

Операторы в языке PASCAL отделяются друг от друга...

- ☐ Пробелом
- ☑ Точкой с запятой
- ☐ Точкой
- ☐ Запятой

50. Задание

Переменная X после выполнения команды X:=SQR(4)/4*2 примет значение ...

- ☐ 4
- ☐ 2
- ☐ 6
- ☑ 8

51. Задание

Команда ввода значений переменных в PASCAL

- ☑ READLN
- ☐ GET
- ☐ APPEND
- ☐ WRITELN

52. Задание

Дополните

Команда, позволяющая переменной A присвоить значение 38, ...

Правильные варианты ответа: A:=38;; A:= 38;; A := 38;; A :=38;;

53. Задание

Выберите правильный ответ

Результат выполнения следующего фрагмента кода:

```
X:= 5; Y:= X+1;
Writeln('X=', X, ' Y=',Y);
```

☐ X=6 Y=5
☐ X=5 Y=5
☑ X=5 Y=6
☐ X=6 Y=6

54. Задание

Укажите правильный ответ

Результат выполнения следующего фрагмента кода:

```
X:= 5;  Y:= 8;
T:= X;  X:=Y;  Y:= T;
Writeln('X=',X, ' Y=',Y);
```

☐ X=5 Y=8
☑ X=8 Y=5
☐ X=5 Y=5
☐ X=8 Y=8

55. Задание

Переменная X после выполнения команд:

```
X:=2; X:=X+1; X:=X*X;
```

примет значение...

Правильные варианты ответа: 9;

56. Задание

Переменная X после выполнения команды X:=SQRT(16)/2 примет значение ...

☐ 4
☑ 2
☐ 6
☐ 8

57. Задание

Укажите правильный ответ

Результат выполнения следующего фрагмента кода:

```
X:=10;  S=(X+5)/2;
Writeln('S=', S:6:2);
```

☐ S=7.500
☑ S= 7.50
☐ S= 7.5
☐ S=7.500000

58. Задание

Введите правильный ответ

Переменная X после выполнения команд: X:=3; X:=X+X; X:=X+X; примет значение...

Правильные варианты ответа: 12;

59. Задание

Введите правильный ответ

Результат выполнения команд: a=5; a:=a*a+1; Writeln('a=',a);

Правильные варианты ответа: a=26;

60. Задание

Отметьте правильные формы записи

Команда вывода в PASCAL

- ☑ WRITELN
- ☐ PRINT
- ☐ SAVE
- ☐ READLN
- ☑ WRITE

61. Задание

Укажите правильную последовательность команд, позволяющих произвести обмен значений переменных X и Y

2: T:=X;
1: READLN(X,Y);
4: Y:=T;
5: WRITELN('X=',X,'Y=',Y);
3: X:=Y;

62. Задание

Укажите правильную последовательность команд, позволяющих ввести значение переменной A и вычислить 7A, используя только операции сложения

3: C:=B+B;
1: READLN(A);
6: WRITELN('7A=', A);
5: A:=D+A;
2: B:=A+A;
4: D:=B+C;

63. Задание

Отметьте правильные ответы

Какие из последовательностей символов в языке Паскаль являются операторами присваивания?

- ☑ a:=b
- ☐ a+b:=c
- ☑ c:=a+b
- ☐ a=:b
- ☐ a+b=:c

64. Задание

Отметьте правильные ответы

Какие последовательности символов в языке Паскаль являются операторами ввода?

- ☑ read (x)
- ☑ read (x, y, z)
- ☐ read (x; y)
- ☐ read (x+y, z)
- ☐ read (x; y; z)

65. Задание

Установите соответствие между элементами групп

<имя_переменной> :=< выражение>;	команда присваивания
Readln(<список переменных>);	команда ввода
Writeln(<список вывода>);	команда вывода
begin <список операторов> end;	составной оператор
	команда выбора

66. Задание

Укажите правильную последовательность команд, позволяющих ввести значение радиуса, вычислить и вывести длину окружности

4: Writeln('L=', L:6:2);
3: L:=2*PI*R;
1: Write('R=');
2: Readln(R);

67. Задание

Укажите правильную последовательность команд, позволяющих ввести значение радиуса, вычислить и вывести площадь круга

3: S:=PI*SQR(R);
2: Readln(R);
4: Writeln('S=', S:6:2);
1: Write('R=');

68. Задание

Укажите правильную последовательность команд, позволяющих ввести стороны треугольника A, B, C и вывести его площадь S

4: Writeln('S=', S:6:2);
2: P:=(A+B+C)/2;
1: Readln(A, B, C);
3: S:=Sqrt(P*(P-A)*(P-B)*(P-C));

69. Задание

Укажите правильную последовательность команд, позволяющих ввести катеты прямоугольного треугольника A, B и вывести его периметр P

1: Readln(A, B);
2: C:=SQRT(A*A+B*B);
3: P:=A+B+C;
4: Writeln('P=', P:6:2);

70. Задание

Установите соответствие между командами и результатом их выполнения:

A:=1; B:=5; Writeln(A, '+', B,'=',A+B);	1+5=6
A:=1; B:=5; Writeln('A+ B=',A+B);	A+B=6
A:=1; B:=5; Writeln('A+ B=',(A+B):5);	A+B= 6
A:=1; B:=5; Writeln(A+B);	6
	A+B=6.00

71. Задание

Установите соответствие между командами и результатом их выполнения

A:=4; B:=8; Writeln(A, '/', B,'=',A/B);	4/8=5.0000000000E-01
A:=4; B:=8; Writeln('A/B=',A/B:4:2);	A/B=0.50
A:=4; B:=8; Writeln('A/ B=',A/B:6:1);	A/B= 0.5
A:=4; B:=8; Writeln(A/B);	5.0000000000E-01
	4/8=0.5

72. Задание

Установите соответствие между командами и результатом их выполнения

A:=14; B:=5; Writeln(A, '-', B,'=',A-B);	14-5=9
A:=14; B:=5; Writeln('A-B=',A-B:1);	A-B=9
A:=4; B:=8; Writeln('A-B=',A-B:6);	A-B= 9
A:=14; B:=5; Writeln(A-B);	9
	14-5=9.00

73. Задание

Введите правильный ответ

Переменная X после выполнения команд:

```
X:=2; X:=X*X; X:=X*X;
```

примет значение...

Правильные варианты ответа: 16;

74. Задание

Введите правильный ответ

Переменная Z после выполнения команд:

```
X:=2; Z:=X*X; Z:=Z+X;
```

примет значение...

Правильные варианты ответа: 6;

75. Задание

Отметьте правильный ответ

После выполнения команды WRITELN...

- ☑ курсор переводится на новую строку
- ☐ курсор остается на прежней строке
- ☐ выводится строка пробелов

76. Задание

Отметьте правильный ответ

При выполнении команды READLN(A,B,C) вводимые значения переменных разделяются...

- ☑ пробелом
- ☐ запятой
- ☐ точкой с запятой
- ☐ ничем не разделяются

77. Задание

Укажите правильный ответ

Результат выполнения следующего фрагмента кода:

```
X:=8;  Y:=5; Sr=(X+Y)/2;
Writeln('Sr=', S:6:3);
```

☐ Sr= 6.50
☐ Sr= 6.5
☐ Sr=6.500000
☑ Sr= 6.500

78. Задание

Выберите правильный ответ

Результат выполнения следующего фрагмента кода:

```
X:= 18; Y:= X mod 5; Y:=Y*Y;
Writeln('Y=',Y);
```

☐ Y=5
☑ Y=9
☐ Y=18
☐ Y=3

79. Задание

Укажите правильный ответ

Результат выполнения следующего фрагмента кода:

```
X:= 15;  Y:= 38;
R:= X;  X:=Y;  Y:= R;
Writeln('X=',X, ' Y=',Y);
```

☑ X=38 Y=15
☐ X=38 Y=38
☐ X=15 Y=15
☐ X=15 Y=38

80. Задание

Результат выполнения следующего фрагмента кода:

```
X:= 5;
T:= X+1;  X:=X+T;
Writeln(X);
```

Правильные варианты ответа: 11;

81. Задание

Результат выполнения следующего фрагмента кода:

```
Z:= 12;  Y:= 8;
Z:= Z+Y;  Y:= Z-Y;
Writeln(Y);
```

Правильные варианты ответа: 12;

82. Задание

Переменная A после выполнения команд:

```
A:=5; A:= A-1; A:=2*A;
```

примет значение...

Правильные варианты ответа: 8;

ОПЕРАТОРЫ ЦИКЛА

83. Задание

Многократно повторяемые участки вычислений называют ... .

Правильные варианты ответа: циклами; циклом; цикл;

84. Задание

Оператор, реализующий в Паскале цикл с предусловием...

- ☐ FOR...
- ☐ REPEAT...
- ☑ WHILE...
- ☐ WRITE...

85. Задание

Оператор, реализующий в Паскале цикл с постусловием...

- ☐ FOR...
- ☑ REPEAT...
- ☐ WHILE...
- ☐ WRITE...

86. Задание

Оператор, реализующий в Паскале цикл с параметром...

- ☑ FOR...
- ☐ REPEAT...
- ☐ WHILE...
- ☐ WRITE...

87. Задание

Оператор, реализующий в Паскале цикл с параметром...

- ☑ FOR...
- ☐ REPEAT...
- ☐ WHILE...
- ☐ WRITE...

88. Задание

Какой из перечисленных операторов цикла всегда выполняется хотя бы один раз

- ☐ FOR...
- ☑ REPEAT...
- ☐ WHILE...

89. Задание

Цикл WHILE выполняется…

- ☐ всегда многократно
- ☑ может не выполниться ни разу
- ☐ всегда выполняется хотя бы один раз

90. Задание

Цикл FOR выполняется…

- ☐ всегда многократно
- ☑ может не выполниться ни разу
- ☐ всегда выполняется хотя бы один раз

91. Задание
Параметр цикла FOR x:=1 to 15 do меняется с шагом...
Правильные варианты ответа: 1;
92. Задание
Параметр цикла FOR x:=15 downto 1 do меняется с шагом
Правильные варианты ответа: -1;
93. Задание
Переменная S в результате выполнения команд s:=0; for k:=1 to 4 do s:=s+k; получит значение...
Правильные варианты ответа: 10;
94. Задание
Переменная S в результате выполнения команд s:=0; for k:=5 downto 2 do s:=s+k; получит значение...
Правильные варианты ответа: 14;
95. Задание
Переменная P в результате выполнения команд P:=1; for k:=1 to 4 do P:=P*k; получит значение...
Правильные варианты ответа: 24;
96. Задание
Переменная Y в результате выполнения команд Y:=1; for k:=3 to 6 do Y:=Y+k; получит значение...
Правильные варианты ответа: 19;
97. Задание
Цикл REPEAT выполняется…
- ☐ всегда многократно
- ☐ может не выполниться ни разу
- ☑ всегда выполняется хотя бы один раз

98. Задание
Переменная s в результате выполнения команд s:=0; a:=2; while a<8 do begin s:=s+a;a:=a+2; end; получит значение...
Правильные варианты ответа: 12;
99. Задание
Переменная s в результате выполнения команд s:=0; a:=5; while a<4 do begin s:=s+a;a:=a+2; end; получит значение...
Правильные варианты ответа: 0;
100. Задание
Переменная a в результате выполнения команд k:=1; a:=0; repeat a:=a+k; k:=k+1;until k>4; получит значение...
Правильные варианты ответа: 10;
101. Задание
Переменная k в результате выполнения команд n:=3456; k:=0; repeat a:=n mod 10; k:=k+1; n:=n div 10; until n=0; получит значение...
Правильные варианты ответа: 4;
102. Задание

При каком значении X произойдет выход из цикла X:=1; While X<=7 do X:=X+2;

Правильные варианты ответа: 9;

103. Задание

При каком значении K произойдет выход из цикла K:=2; REPEAT K:=K+2 UNTIL K>8;

Правильные варианты ответа: 10;

104. Задание

При каком значении K произойдет выход из цикла FOR K:=2 to 10 do;

Правильные варианты ответа: 11;

105. Задание

Установите соответствие между элементами групп

цикл с параметром	FOR <переменная>:=<нач.знач> to <кон.знач> do
цикл с предусловием	WHILE <условие> do <оператор>
цикл с постусловием	REPEAT <операторы> UNTIL <условие>

106. Задание

Установите в правильной последовательности команды, позволяющие вычислить факториал введенного натурального числа N

3: FN:=FN*I;
4: Writeln(N,'!=', FN);
1: Readln(N); FN:=1;
2: For I:=1 to N do

107. Задание

Установите в правильной последовательности команды, позволяющие вывести таблицу значений функции y=sin x для x от 0 до 2 с шагом 0.2

5: x:=x+0.2;
3: y:=sin(x);
6: Until x>2;
4: writeln('x=', x:3:1, 'y=',y:6:3);
1: x:=0;
2: Repeat

108. Задание

Установите в правильной последовательности команды, позволяющие найти сумму цифр введенного натурального числа N

6: end;
7: Writeln('S=', S);
2: S:=0;
3: While N<>0 do begin
4: A:=N mod 10; S:=S+A;
1: Readln(N);
5: N:=N div 10;

109. Задание

Установите в правильной последовательности команды, позволяющие определить порядковый номер первого положительного члена арифметической прогрессии -17, -13,...

1: A:=-17; N:=1;
3: A:=A+4; N:=N+1;
5: Writeln('N=', N);
2: While A<=0 do begin
4: end;

110. Задание

Выберите неправильную форму записи оператора цикла

- ☐ FOR I:=1 TO 10 DO WRITELN(I);
- ☐ FOR I:=10 DOWNTO 1 DO WRITELN(I);
- ☑ FOR I:=10 DOWNTO 1 DO STEP -1 WRITELN(I);

111. Задание

Выберите неправильную форму записи оператора цикла

- ☐ WHILE I<5 DO I:=I+1;
- ☐ WHILE I<5 DO BEGIN I:=I+1; WRITELN(I); END;
- ☑ WHILE I<5 TO BEGIN I:=I+1; WRITELN(I); END;

112. Задание

Выберите правильную форму записи оператора цикла

- ☑ REPEAT I:=I+1; UNTIL I>9;
- ☐ UNTIL I>9 I:=I+1; REPEAT
- ☐ UNTIL I:=I+1; REPEAT I>9;

113. Задание

Укажите правильные ответы

Оператором цикла языка Паскаль являются

- ☑ while x < 0 do x:= x + 0.5;
- ☐ while x < 0 then x:=x - 100;
- ☐ while 0 < x < 1 do x:= sqr (x) + 0.01;
- ☑ while x = y do begin x:= x - 1; y:= y + 1; end;
- ☐ while x := 0 do y:= 2 * y;

114. Задание

Установите в правильной последовательности команды, позволяющие вычислить сумму первых N натуральных чисел

4: Writeln('S=', S);
1: Readln(N); S:=0;
3: S:=S+I;
2: For I:=1 to N do

115. Задание

Установите в правильной последовательности команды, позволяющие вывести таблицу значений функции y=tg x для x от 1 до 2 с шагом 0.1

5: x:=x+0.1;
2: Repeat
3: y:=sin(x)/cos(x);
4: writeln('x=', x:3:1, 'y=',y:6:3);

1: x:=1;
6: Until x>2;

116. Задание

Установите в правильной последовательности команды, позволяющие найти количество цифр введенного натурального числа N

5: N:=N div 10;
7: Writeln('K=', K);
2: K:=0;
4: A:=N mod 10; K:=K+1;
3: While N<>0 do begin
6: end;
1: Readln(N);

117. Задание

Установите в правильной последовательности команды, позволяющие определить порядковый номер первого отрицательного члена арифметической прогрессии 14, 11,...

5: Writeln('N=', N);
3: A:=A-3; N:=N+1;
1: A:=14; N:=1;
2: While A>=0 do begin
4: end;

118. Задание

Переменная S в результате выполнения команд

```
S:=0; For k:=5 downto 2 do S:=S+1;
```

получит значение...

Правильные варианты ответа: 4;

119. Задание

Переменная T в результате выполнения команд

```
T:=1; For K:=1 to 3 do T:=T+2*K;
```

получит значение...

Правильные варианты ответа: 13;

120. Задание

Сколько раз будет выведено слово 'PASCAL' в результате выполнения данного фрагмента программы:

```
For K:=1 to 3 do
    For T:=1 to 4 do
        Writeln('PASCAL');
```

Правильные варианты ответа: 12;

121. Задание

Переменная K в результате выполнения фрагмента программы

```
K:=0; For I:=1 to 4 do
            For J:=2 to 5 do
                 K:=K+1;
```

примет значение...
Правильные варианты ответа: 16;

122. Задание

Укажите результат выполнения данного фрагмента программы:

```
For M:=1 to 3 do
  begin
     S:=0;
     For N:=M to 4 do
         S:=S+N;
     Write('S=',S:4);
  end;
```

- ☑ S=10 S=9 S=7 S=4
- ☐ S=10 S=10 S=10 S=10
- ☐ S=1 S=2 S=3 S=4
- ☐ S=4 S=7 S=9 S=10
- ☐ S=0 S=0 S=0 S=0

123. Задание

Установите правильную последовательность выполнения команд для вывода N членов последовательности, заданной формулой Ai=3*i+2

3: begin
2: For i:=1 to N do
4: A:=3*i+2;
1: Readln(N);
5: Writeln(A);
6: end;

ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ, ОПЕРАТОР ВЫБОРА

124. Задание

Укажите правильные формы записи
Условный оператор в языке Паскаль

- ☐ IF a>0 TO a:=1;
- ☑ IF a>0 THEN a:=1;
- ☐ IF a>0 ELSE a:=1;
- ☑ IF a>0 THEN a:=1 ELSE a:=0;
- ☐ IF a>0 TO a:=1 ELSE a:=0;

125. Задание

Укажите правильные ответы
Логическими являются следующие выражения…

- ☑ 2<>10
- ☐ a:=a+1
- ☐ sin(x+1)

- ☑ 2=3
- ☑ x>=1
- ☑ sin(x+1)=0

126. Задание

Укажите правильный ответ

Логическое выражение может принимать значения

- ☐ любые
- ☑ true, false
- ☐ and, or, not
- ☐ целочисленные

127. Задание

Укажите правильный ответ

AND - это…

- ☐ логическое НЕ
- ☐ логическое ИЛИ
- ☑ логическое И

128. Задание

Укажите правильный ответ

OR - это…

- ☐ логическое НЕ
- ☑ логическое ИЛИ
- ☐ логическое И

129. Задание

Укажите правильный ответ

Какое из перечисленных логических выражений принимает значение TRUE

- ☐ (3>7) AND (6=2+4)
- ☑ (7<=7) OR (2/9>10)
- ☐ (2+6<>8) AND (0<-7)
- ☐ (2=8) OR (0>7)

130. Задание

Укажите правильный ответ

Какое из перечисленных логических выражений принимает значение FALSE

- ☐ (10>7) AND (6=2+4)
- ☐ (2+6=8) AND NOT(0<-7)
- ☑ (7<7) OR (1/9>1)
- ☐ (6<=8) OR NOT(2<7)

131. Задание

Укажите правильные ответы

Какие из перечисленных логических выражений принимают значение TRUE

- ☑ (3>0.7) AND (6=2+4)
- ☑ (7<=17) OR (2/9>10)
- ☐ (2+6<>8) AND (0<-7)
- ☐ (12=8) OR (0>7)

132. Задание

Укажите правильные ответы

Какие из перечисленных логических выражений принимают значение FALSE

- ☑ (10>7) AND (6<>1+5)
- ☐ (5+6=11) AND NOT(0<-7)
- ☑ (14<7) OR (1/9>1)
- ☐ (6<=8) OR NOT(2<7)

133. Задание

Укажите правильные формы записи условного оператора в языке Паскаль

- ☑ IF a>0 THEN a:=1 ELSE begin a:=0; b:=b+1 end;
- ☐ IF a>0 THEN a:=1 ELSE a:=0 end;
- ☐ IF a>0 THEN a:=1 ELSE begin a:=0; b:=b+1;
- ☑ IF a>0 THEN begin a:=1; b:=b+1; end ELSE a:=0;

134. Задание

Введите правильный ответ

Переменная X после выполнения команд: X:=-2; If X >=0 then x:=x*2 else x:=abs(x); примет значение ...

Правильные варианты ответа: 2;

135. Задание

Введите правильный ответ

Переменная X после выполнения команд:

X:=22; If X mod 2=0 then X:=X div 2;

примет значение ...

Правильные варианты ответа: 11;

136. Задание

Укажите правильный ответ

Логическое выражение, принимающее значение TRUE, если значение X попадет в интервал [2,6]

- ☐ X>=2 AND X<=6
- ☑ (X>=2) AND (X<=6)
- ☐ (X<=2) OR (X>=6)
- ☐ 2<=X<=6

137. Задание

Укажите правильную форму записи условного оператора в языке Паскаль

- ☐ IF A>5 OR B<3 THEN WRITELN(A) ELSE WRITELN(B);
- ☑ IF (A>5) OR (B<3) THEN WRITELN(A);
- ☐ IF (A>5) OR (B<3) THEN WRITELN(A); ELSE WRITELN(B);
- ☐ IF (A>5) OR (B<3) THEN WRITELN(A); END ELSE WRITELN(B);

138. Задание

Сопоставьте название оператора с его видом.

Оператор присваивания

<ИДЕНТИФИКАТОР>:=<ВЫРАЖЕНИЕ>

Условный оператор	IF<ВЫРАЖЕНИЕ>THEN<ОПЕРАТОР-1>ELSE<ОПЕРАТОР-2>
Составной оператор	BEGIN<ОПЕРАТОР-1>;<ОПЕРАТОР-2>;...<ОПЕРАТООР-N>END
Оператор безусловного перехода	GOTO<МЕТКА> BEGIN GOTO<ОПЕРАТОР>

139. Задание

Какие из последовательностей символов являются условным оператором языка Паскаль?

- ☑ if x<y then x:=0 else y:=0;
- ☐ if x>y then x:=0 else 1;
- ☑ if x>=y then begin x:=0; y:=0 end else write (z);
- ☐ if x<y then 100 else z:=5;
- ☐ if x<y<z then z:=z+1 else end;

140. Задание

Какие из последовательностей символов являются условными операторами языка Паскаль?

- ☐ if a>b then a-b;
- ☐ if a<b<c then c:=c+1;
- ☑ if a<3.17 then b:=b+1;
- ☑ if a<>b then c:=c+1;
- ☑ if a<=b then a:=b+1;

141. Задание

Какие из вложенных условных операторов языка Паскаль допустимы?

- ☑ if x+y<z then x:=x+1 else if y>z then z:=0 else y:=0;
- ☐ if x+y<z then if y>z else z:=0;
- ☑ if x+y<z then if y>z then z:=0 else y:=0 else z:=0;
- ☐ if x+y<z then if y>z then z:=0;
- ☐ if x+y<z then if y>z then z:=0 else y:=0 then x=z else z:=0;

142. Задание

Результат выполнения программы при N=1...

```
Program Prim;
  Var N, A, B: integer;
BEGIN
  A:=7; B:=3;
  Write('N='); Readln(N);
  CASE N OF
     1 : Writeln(A+B);
     2 : Writeln(A-B);
     3 : Writeln(A*B);
     ELSE  Writeln(A=B);
   END;
```

```
END.
```
Правильные варианты ответа: 10;

143. Задание

Результат выполнения программы при N=2...

```
Program Prim;
  Var N, A, B: integer;
BEGIN
  A:=14; B:=3;
  Write('N='); Readln(N);
  CASE N OF
     1 : Writeln(A+B);
     2 : Writeln(A-B);
     3 : Writeln(A*B);
     ELSE  Writeln(A=B);
   END;
END.
```
Правильные варианты ответа: 11;

144. Задание

Результат выполнения программы при N=4...

```
Program Prim;
  Var N, A, B: integer;
BEGIN
  A:=11; B:=5;
  Write('N='); Readln(N);
  CASE N OF
     1 : Writeln(A+B);
     2 : Writeln(A-B);
     3 : Writeln(A*B);
     ELSE  Writeln(A=B);
   END;
END.
```
Правильные варианты ответа: false;

145. Задание

Укажите правильные ответы

Логическими являются следующие выражения…

- ☑ 2<>10
- ☑ x>=1
- ☑ sin(x+1)=0
- ☐ cos(x+y)
- ☐ x:=x+2
- ☐ a:=1

146. Задание

Введите правильный ответ

Переменная X после выполнения команд:

```
X:=12; If X MOD 2<>0 then X:=X-2 else X:=0;
```

примет значение ...

Правильные варианты ответа: 0;

147. Задание

Установите правильную последовательность элементов, составляющих условный оператор для выбора максимального из чисел A и B

1: IF
4: Max:=A
2: A>B
3: THEN
6: Max:=B
5: ELSE

148. Задание

Установите правильную последовательность элементов, составляющих условный оператор для выбора мингимального из чисел A и B

1: IF
5: ELSE
6: Min:=B
3: THEN
2: A<B
4: Min:=A

149. Задание

Установите правильную последовательность команд, позволяющих ввести числа A, B, C и выбрать из них минимальное

3: If C<Min then Min:=C;
2: If A<B then Min:=A else Min:=B;
1: Readln(A, B, C);
4: Writeln('Min=', Min);

150. Задание

Установите правильную последовательность команд, позволяющих ввести числа A, B, C и выбрать из них максимальное

1: Readln(A, B, C);
3: If C>Max then Max:=C;
4: Writeln('Max=', Max);
2: If A>B then Max:=A else Max:=B;

151. Задание

Результат выполнения программы при K=1...

```
Program Prim;
  Var K, A, B, C: integer;
BEGIN
  A:=11; B:=4;
```

```
  Write('K='); Readln(K);
  CASE K OF
     1 : C:=A+B;
     2 : C:=A-B;
     3 : C:=A*B;
     ELSE  C:=-1;
   END;
  Writeln(C);
END.
```

Правильные варианты ответа: 15;

152. Задание

Результат выполнения программы при K=2...

```
Program Prim;
  Var K, A, B, C: integer;
BEGIN
  A:=11; B:=4;
  Write('K='); Readln(K);
  CASE K OF
     1 : C:=A+B;
     2 : C:=A-B;
     3 : C:=A*B;
     ELSE  C:=-1;
   END;
  Writeln(C);
END.
```

Правильные варианты ответа: 7;

153. Задание

Результат выполнения программы при K=3...

```
Program Prim;
  Var K, A, B, C: integer;
BEGIN
  A:=11; B:=4;
  Write('K='); Readln(K);
  CASE K OF
     1 : C:=A+B;
     2 : C:=A-B;
     3 : C:=A*B;
     ELSE  C:=-1;
   END;
  Writeln(C);
END.
```

Правильные варианты ответа: 44;

154. Задание

Результат выполнения программы при K=5...

```
Program Prim;
  Var K, A, B, C: integer;
BEGIN
  A:=11; B:=4;
  Write('K='); Readln(K);
  CASE K OF
     1 : C:=A+B;
     2 : C:=A-B;
     3 : C:=A*B;
     ELSE  C:=-1;
   END;
  Writeln(C);
END.
```

Правильные варианты ответа: -1;

155. Задание

Введите правильный ответ

Переменная X после выполнения команд:

```
X:=12; If (X >=20) OR ( X MOD 2=0) then X:=X-2 else X:=X*2;
```

примет значение ...

Правильные варианты ответа: 10;

156. Задание

Введите правильный ответ

Переменная X после выполнения команд:

```
X:=SQR(9); If (X <=10) OR ( X MOD 2=0) then X:=X-2 else X:=X+2;
```

примет значение ...

Правильные варианты ответа: 83;

157. Задание

Введите правильный ответ

Переменная X после выполнения команд:

```
X:=ABS(-9); If (X >=1) AND ( X <=9) then X:=SQR(X) else
X:=SQRT(X);
```

примет значение ...

Правильные варианты ответа: 81;

158. Задание

Введите правильный ответ

Переменная X после выполнения команд:

X:=16; If (X >=10) AND (X MOD 2=1) then X:=X+1 else X:=X-1;

примет значение ...
Правильные варианты ответа: 15;
159. Задание
Введите правильный ответ
Переменная X после выполнения команд: X:=-2; If X >=0 then x:=x*2 else x:=abs(x); примет значение ...
Правильные варианты ответа: 2;

Файловая папка презентаций

Лекционные презентации

ЛКП 1.1
ЛКП 1.2
ЛКП 2.1
ЛКП 2.2
ЛКП 2.3
ЛКП 2.4
ЛКП 2.5
ЛКП 2.6
ЛКП 3.1
ЛКП 3.2

Практические презентации

ПКП 1.1
ПКП 1.2
ПКП 2.1
ПКП 2.2
ПКП 2.3
ПКП 2.4
ПКП 2.5
ПКП 2.6
ПКП 2.7
ПКП 2.8
ПКП 3.1

Лабораторные презентации

ЛБП 1.1
ЛБП 1.2
ЛБП 2.1
ЛБП 2.2
ЛБП 3.1
ЛБП 3.2
ЛБП 4.1
ЛБП 4.2

Глоссарий

Delphi – объектно-ориентированный язык программирования, созданный на основе языка программирования Pascal и среда разработки программных продуктов компании Borland.

Microsoft Visual Studio – интегрированная среда разработки программных продуктов компании Microsoft, которая, в том числе, поддерживает языки программирования для платформы Microsoft .NET Framework.

Pascal – один из наиболее известных языков программирования высокого уровня, который широко используется в целях обучения программированию. Объектный Паскаль (Object Pascal) является объектно-ориентированным расширением Паскаля, на основе которого создан язык программирования Delphi.

Автоматизированное программирование- совокупность методов и инструментальных средств для проектирования и сопровождения прикладных программ.

Алгоритм – порядок действий, которые необходимо выполнить для решения определенной задачи.

Алгоритм линейной структуры(Следование) - Алгоритм, в котором все действия выполняются последовательно друг за другом.

Алгоритмизация - Техника составления алгоритмов и программ для решения задач на ЭВМ.

Алфавит языка программирования - Набор символов, с помощью которого могут быть образованы величины, выражения и операторы данного языка.

Ассемблер (Assembler) – язык программирования низкого уровня, инструкции которого соответствуют инструкциям машинного кода. Также, ассемблером называют программу – транслятор с языка программирования низкого уровня в машинный код.Блок-схема – графическая нотация для описания алгоритмов. Используется программистами в процессе разработки и анализа логики работы программных компонентов.

Ветвление - Схема, в которой предусмотрено разветвление указанной последовательности действий на два направления в зависимости от итога проверки заданного условия.

Вещественный Тип(REAL) - Элементы подмножеств вещественного типа.

Выражение - Совокупность операций и операндов.

Графический способ записи Алгоритмов - Описание алгоритма с помощью графических символов.

Дискретность - Свойство алгоритма разчленять предопределённый алгоритмом вычислительный процесс, на отдельные этапы, элементарные операции.

Загрузочный модуль– программный модуль, представленный в форме, пригодной для загрузки в основную память для выполнения

Идентификатор - Последовательность символов, начинающаяся с буквы, для наименования объектов.

Инструментальное программное обеспечение син.Инструментальные программные средства англ.Software tools - программное обеспечение, используемое в ходе разработки, корректировки или развития других программ: редакторы, компиляторы, отладчики, вспомогательные системные программы, графические пакеты и др.

Интерпретатор - Програмный продукт, выполняющий предъявленную программу путём одновременного её онализа и реализации предписанных ею действий.

Исходный модуль– программный модуль на исходном языке программирования, обрабатываемый транслятором

Итерация - Повторение последовательности операторов, включающим проверку условия в начале каждого прохода цикла.

Компилятор - Разновидность транслятора, обеспечивающая перевод программ с языка высокого уровня на язык более низкого уровня или машинозависимый язык.

Константа - Элемент данных, сохраняющий неизменное значение в течении всего времени выполнения программы.

Логический тип(BOOLEAN) - Одно из двух истиностных значений, обозначаемых предопределёнными именами false и true.

Массовость - Свойство алгоритма, позволяющее решать однотипичные задачи с различными исходными дынными по одному алгоритму.

Метка - Произвольный идентифакатор, позволяющий именовать некоторый оператор программы и таким образом ссылаться на него.

Объектно-ориентированное программирование (ООП) –парадигма программирования. Суть объектно-ориентированного программирования в представлении обрабатываемой информации в виде объектов – экземпляров классов. Класс – это новый (по отношению к процедурному программированию) тип данных, который объединяет в себе и структуры данных и параметризованные процедуры.

Объектный модуль– программный модуль, получаемый в результате компиляции исходного модуля

Оператор выбора - Оператор для программирования алгоритмов с множиственным выбором (CASE-OF-ELSE-END)

Оператор перехода - Оператор передачи управления соответствующему меченому оператору(GOTO)

Оператор повторений WHILE-DO - Оператор для программирования алгоритмов циклической структуры с предпроверкой условия.

Оператор присваивания - Оператор, присваивающий переменной или имени функции значение выражения, стоящего справа от знака присваивания.

Определённость - Свойство алгоритма исключать произвольность толкования любого из предписаний и заданного порядка исполнения.

Основные структуры алгоритмов - Ограниченный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий.

Переменная - Объект, имеющий фиксированное имя, фиксированный тип и изменяющееся в зависимости от применяемых действий значение.

Подпрограмма – программа, являющаяся частью другой программы и удовлетворяющая требованиям языка программирования

Правила хорошего стиля - результат соглашения между программистами. Считается, что в соответствии с правилами хорошего стиля программный код должен поддерживать: - очевидную логику; - естественные выражения; - осмысленные имена; - аккуратное форматирование; развернутые комментарии; - отсутствие хитрых трюков и необычных конструкций.

Прикладная программа– программа, предназначенная для решения задачи или класса задач в определенной области применения системы обработки информации

Прикладное программирование – процесс разработки программного обеспечения, предназначенного для решения прикладных задач в определенной сфере деятельности. Такое программное обеспечение называют прикладным, и оно характеризуется тем, что не использует вычислительные ресурсы аппаратного обеспечения напрямую, а делает это посредством операционной системы.

Программа – данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма

Программирование - Процесс определения последовательности инструкций, которые должен выполнить компьютер для решения определённой задачи.

Программирование - англ.Programming - процесс подготовки задач для их решения с помощью компьютера; итерационный процесс составления программ.

Программист (англ.Programmer) - специалист, занимающийся разработкой и проверкой программ. Различают системных и прикладных программистов.

Программное обеспечение– совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ

Программный модуль– программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими программными модулями для загрузки в оперативную память

Программный способ записи алгоритмов - запись алгоритма на языке программирования.

Результативность - Свойство алгоритма через определённое число шагов приводит к выдаче результатов или сообщения о невозможности решения задачи.

Рекурсивная подпрограмма– подпрограмма, которая может обращаться к себе самой

Символьный(литерный) тип - Элементы конечного и упорядоченного множества символов.

Синтаксическая диаграмма - Направленый граф для описания синтаксиса языка, в соответствии с которым строится синтаксически правильная программа.

Система программирования – система, образуемая языком программирования, компиляторами или интерпретаторами программ, представленных на данном языке, соответствующей документацией, а также вспомогательными средствами для подготовки программ к форме, пригодной для выполнения

Системная программа – программа, предназначенная для поддержания работоспособности системы обработки информации или повышения эффективности ее использования в процессе выполнения прикладных программ

Скалярные переменные - Переменные, имеющие в качестве текущего значения только одну величену.

Словесный способ записи алгоритмов - Запись Последовательности действийв произвольном изложении на естественном язке

Составной оператор - Последовательность произвольных операторов программы, заключённая в операторные скобки(BEGIN-END)

Стандартные типы данных - Изначало определённые типы данных, встроенные в ЭВМ

Стиль программирования - набор приемов или методов программирования, которые используют программисты, чтобы получить правильные, эффективные, удобные для применения и легкочитаемые программы.

Структурированная переменная - Переменная, состоящая из нескольких элементов или компонент, на которую можно ссылаться как на единный объект.

Структурно-стилизованный способ записи алгоритмов - Запись алгоритма путём использования ограниченного набора типовых синтаксических конструкций.

Теоретическое программирование - раздел информатики, изучающий описание процессов обработки данных.

Технология программирования - дисциплина, изучающая технологические процессы программирования и порядок их прохождения.

Тип данных - Множество значений, которые может принимать переменная и совокупность операций, выполняемых с этими данными.

Транслятор - Программа, осуществяющая перевод текстов с одного языка на другой.

Управляющая программа– системная программа, реализующая набор функций управления, в который включают управление ресурсами и взаимодействием с внешней средой системы обработки информации, восстановление работы системы после появления неисправностей в технических средствах

Уровень языка программирования - Смысловая ёмкость его конструкции и его ориентация на программиста-человека.

Условный оператор - Оператор с ключевыми словами IF-THEN-ELSE для программирования алгоритмов разветвляющейся структуры.

Целый тип(INTEGER) - Элементы подмножества целых чисел.

Язык высокого уровня– язык программирования, понятия и структура которого удобны для восприятия человеком

Язык программирования - Система обозначений для точного описания алгоритмов для ЭВМ.

Язык программирования, син.Алгоритмический язык (англ.Programming language; Algorithmic language) - искусственный (формальный) язык, предназначенный для записи алгоритмов. Язык программирования задается своим описанием и реализуется в виде специальной программы: компилятора или интерпретатора.