ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Учебно-методический комплекс

Витебск
2010

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«Витебский государственный технологический университет»

Е.Ю. ВАРДОМАЦКАЯ

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Учебно-методический комплекс

для студентов специальности переподготовки
1-40.01.73 «Программное обеспечение
информационных систем»

ВИТЕБСК
2010

УДК 004.4 (075)
ББК 73
В 18

Рецензенты:

Бочкин А. И., кандидат физико-математических наук, заведующий кафедрой информатики и информационных технологий учреждения образования «Витебский государственный университет им. П.М. Машерова»

Казаков В. Е. , кандидат технических наук, заведующий кафедрой информатики учреждения образования «Витебский государственный технологический университет»

Рекомендовано в качестве пособия редакционно-издательским советом УО «ВГТУ», протокол № 4 от 17.06.2010.

В18 Вардомацкая. Е. Ю. Основы алгоритмизации и программирования : учебно-методический комплекс / Е. Ю. Вардомацкая. – Витебск : УО «ВГТУ», 2010. – 164 с.

ISBN 978-985-481-215-1

Пособие представляет собой комплекс учебно-методических материалов по дисциплине «Основы алгоритмизации и программирования», включающий в себя курс лекций, методические указания к практическим занятиям, методические указания к выполнению лабораторных работ, вопросы к экзамену, образец задания для тестового контроля знаний, список литературы.УМК предназначен для студентов специальности переподготовки «Программное обеспечение информационных систем», может использоваться студентами других специальностей дневной и заочной формы обучения при изучении соответствующих тем и самоподготовке.

К печатному изданию прилагается диск с электронной версией УМК и разобранными типовыми заданиями.

УДК 004.4 (075)
ББК 73

ISBN 978-985-481-215-1

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Дисциплина «Основы алгоритмизации и программирования» является первой при изучении дисциплин инфрмационного направления слушателями специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем». Назначение этого курса заключается в том, чтобы дать слушателям навыки алгоритмического мышления на примере одного из языков структурного программирования Паскаль.

Данный учебно-меодический комплекс (УМК) представляет собой комплекс методических материалов по дисциплине «Основы алгоритмизации и программирования» для студентов специальности переподготовки «Программное обеспечение информационных систем» и включает в себя:

- тематический план курса,
- курс лекций,
- методические указания к практическим занятиям,
- методические указания к выполнению лабораторных работ,
- вопросы к экзамену,
- образец задания для тестового контроля знаний,
- список литературы.

Учебно-методический комплекс (УМК) предназначен для изучения теоретических материалов, проведения практических занятий и лабораторных работ, активизации самостоятельной работы студентов при изучении дисциплины – при подготовке к занятиям и контролю знаний.

Данный УМК построен таким образом, чтобы максимально активизировать самостоятельную работу студентов под руководством преподавателя при закреплении знаний по базовым темам дисциплины «Основы алгоритмизации и программирования».

В разделах «Практические занятия» и «Лабораторный практикум» УМК предусмотрены варианты заданий для самостоятельного выполнения с целью закрепления полученных знаний и вопросы для подготовки к тестовому контролю знаний. Большинство заданий к практическим занятиям и лабораторным работам относятся к продуктивному и аналитическому уровню. Чтобы разработать алгоритмы решений, требуется выполнить математический или логический анализ. База вопросов составлена таким образом, что разработанные на ее основе тестовые задания могут быть реализованы в любой тестирующей среде для проведения компьютерного тестового контроля знаний.

УМК ««Основы алгоритмизации и программирования» может использоваться студентами других специальностей дневной и заочной форм обучения при изучении соответствующих тем и самоподготовке.

К печатному изданию прилагается диск с электронной версией УМК и разобранными типовыми заданиями.

ТЕМАТИЧЕСКИЙ ПЛАН КУРСА

«ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»

для специальности переподготовки
1-40 01 73 **«Программное обеспечение информационных систем»**

Квалификация специалиста: *«инженер-программист»*
Форма обучения: *очная (вечерняя), заочная*

№ п/п	Наименование тем	Количество часов								Вид контроля знаний
		Всего	В т.ч. с использованием ВТ	Распределение по видам занятий						
				лекции	практические занятия	семинары	лабораторные занятия	выездные занятия	самостоятельная работа	
1	Введение			1						
2	Основы алгоритмизации			3						
3	Основы структурного программирования			1						
4	Элементы языка Pascal. Типы данных			3						
5	Операторы языка Pascal			8	4		12			
6	Подпрограммы языка Паскаль			4	4		4			
7	Регулярные типы. Массивы			4	4		8			
8	Строки и записи			4			4			
9	Разработка библиотек пользователя			4			4			
	ИТОГО			**32**	**12**		**32**			

КУРС ЛЕКЦИЙ

Лекция 1

ОБЩИЕ СВЕДЕНИЯ ОБ ЭВМ

План лекции

- Классификация компьютеров по назначению
- Классификация ПК
- Структурная схема ПК
  - Внутренние устройства ПК
  - Внешние устройства ПК
  - Иерархия памяти ПК
- Конфигурация ПК
- Параметры, влияющие на производительность ПК
- Принцип работы ПК
- Основы алгоритмизации инженерно-технических задач
  - Алгоритмы и их свойства
  - Способы описания алгоритмов
  - Основные управляющие структуры

Компьютер – это электронное вычислительное устройство. Существует достаточно много систем классификации компьютеров. Рассмотрим некоторые из них, наиболее часто упоминающиеся в доступной технической литературе.

Классификация компьютеров по назначению

Это один из наиболее ранних методов классификации, и связан он с тем, как компьютер применяется. По этому принципу различают

• *Большие ЭВМ* – мэйнфреймы, на базе которых создают ИВЦ, включающие в себя несколько отделов или групп. Это большие компьютеры, с высоким быстродействием и большими вычислительными ресурсами, которые могут обрабатывать большое количество данных и выполнять обработку запросов одновременно нескольких тысяч пользователей.

- *Мини-ЭВМ* – для управления производственными процессами, автоматического проектирования, выполнения технологических и финансовых расчетов. Для организации работы мини-Эвм также требуется ВЦ.
- *Микро-ЭВМ* – как правило для выполнения вспомогательных операций на ВЦ.
- *Персональные компьютеры (ПК)* – компьютеры, которые могут использоваться одним человеком автономно, независимо от других компьютеров.

Классификация ПК

ПК в свою очередь условно подразделяются на:

- *бытовые ПК,*
- *профессиональные ПК.*

Но начиная с 1999 г. в области ПК начал действовать международный сертификационный стандарт – сертификация PC99, в соответствии с которым установлены следующие категории ПК:

- *массовые ПК* (Consumer PC),
- *деловые ПК* (Office PC) – минимизированы требования с средствам воспроизведения графики,
- *портативные ПК* (Mobile PC) – обязательно наличие средств для создания соединений удаленного доступа, то есть средств компьютерной связи,
- *рабочие станции* (Workstations PC) – повышены требования к устройствам хранения данных,
- *развлекательные ПК* (Enterteinmemt PC) – повышены требования к средствам воспроизведения графики и звука.

Кроме того, ПК классифицируют *по типоразмерам*:

- *настольные (desktop)* – используются в стационарных условиях комнаты или кабинета и располагаются на рабочем столе. Настольные компьютеры состоят из нескольких блоков – системного блока, монитора, клавиатуры и мышки, которые соединены между собой;
- *портативные (notebook)* – ноутбук имеет жидкокристаллический дисплей, клавиатуру, совмещенную с системным блоком, дисковод для 3,5" дискет и оптический дисковод (CD-ROM, CD-RW или комбинированный DVD+RW). Кроме того, обязательно имеется манипулятор для управления курсором. По размеру ноутбуки такие, что легко помещаются в портфель;
- *карманные (palmtop),* что означает – лежащий на ладони. В этих компьютерах программы занесены в микросхемы. В набор программ входит операционная система, текстовый и графический редакторы, система баз данных и электронные таблицы, программы для работы в Интернете. Эти компьютеры позволяют обрабатывать документы, вести базы данных, производить вычисления, распечатывать документы, записывать их на дискету, работать в Интернете, но установить новые программы нельзя.

Кроме палмтопов есть карманные компьютеры, которые называются *PDA – personal digital assistent* – личный цифровой ассистент. Эти компьютеры не имеют клавиатуры. Они оснащены сенсорным экраном и информация вводится на экран при помощи специальной указки-стека. Общее название карманных компьютеров – handhold computers – компьютеры, которые держат в руках. Переносной и карманный компьютеры удобны для использования в поездках.

ПК классифицируют *по совместимости:*

- *аппаратная совместимость* (существуют две платформы) – IBM PC совместимые ПК и Apple Macintosh совместимые ПК;
- *совместимость на уровне ОС;*
- *программная совместимость;*
- *совместимость на уровне данных.*

Структурная схема ПК

Персональный компьютер включает в свой состав следующие **основные устройства:**

- системный блок, содержащий большинство электроники, ответственной за работоспособность ПК;
- монитор – устройство для отображения визуальной информации;
- клавиатура, позволяющая вводить в компьютер информацию и управляющие команды;
- манипулятор "мышь".

К **периферийным устройствам ПК** относятся: принтер, джойстик, сканер, внешние оптические дисководы, внешние флоппи-дисководы, модем, стриммер, плоттер, дигитайзер, цифровая видеокамера, световое перо, графический планшет и другие устройства, подключаемые к системному блоку через порты.

Системный блок (рис.1) содержит следующие устройства:

- Блок питания – обеспечивает электроэнергией все электронные устройства ПК. Характеризуется максимальной отдаваемой мощностью (200-250 вт).
- Системную (материнскую) плату.
- Комплекс адаптеров для подключения внешних устройств – обеспечивают возможность вывода информации из компьютера на внешнее устройство.
- Дисковод НЖМД – внешнее запоминающее устройство, предназначенное для долговременного хранения информации на компьютере.

- Дисковод НГМД – одно из внешних запоминающих устройств, предназначенное для записи данных на магнитные диски (дискеты).
- Дисковод CD-ROM – устройство, предназначенное для чтения компакт-дисков на компьютере.

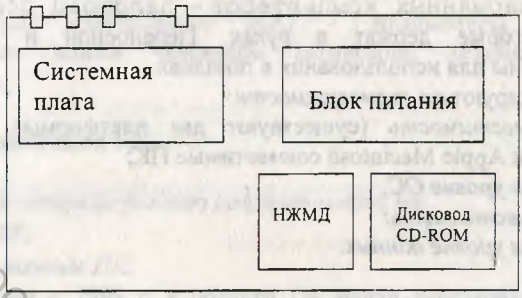


Рисунок 1 – Системный блок

Основным элементом системного блока является системная (материнская) плата.

На системной плате располагаются следующие элементы:

- **микропроцессор** – центральный блок ПК, предназначенный для управления работой всех остальных устройств компьютера и выполнения арифметических и логических операций над информацией.
- **разъемы для плат** оперативной памяти (ОЗУ) и постоянного запоминающего устройства (ПЗУ);
- **микросхемы быстрой памяти** – внешняя Кэш-память – служит для ускорения работы с оперативной памятью. В кэш-память записывается та часть информации оперативной памяти, которая изменяется в данный момент. Таким образом уменьшается время поиска информации в памяти и увеличивается скорость выполнения программы. В настоящее время используется внешняя кэш-память от 1 Мб.
- **микросхема базовой системы ввода-вывода (BIOS).** При включении питания BIOS тестирует состояние компьютера и его элементов и затем передает управление компьютерсм центральному процессору.
- **разъемы для плат расширения (порты),** которые служат для соединения с компьютером периферийных устройств: дисководов, монитора, мышки, клавиатуры, сканера, системой лазерного диска и т.д.);
- **шина расширения и адресная шина** – обеспечивающие сопряжение и связь всех устройств компьютера между собой.

Структурная схема ПК представлена на рисунке 2.

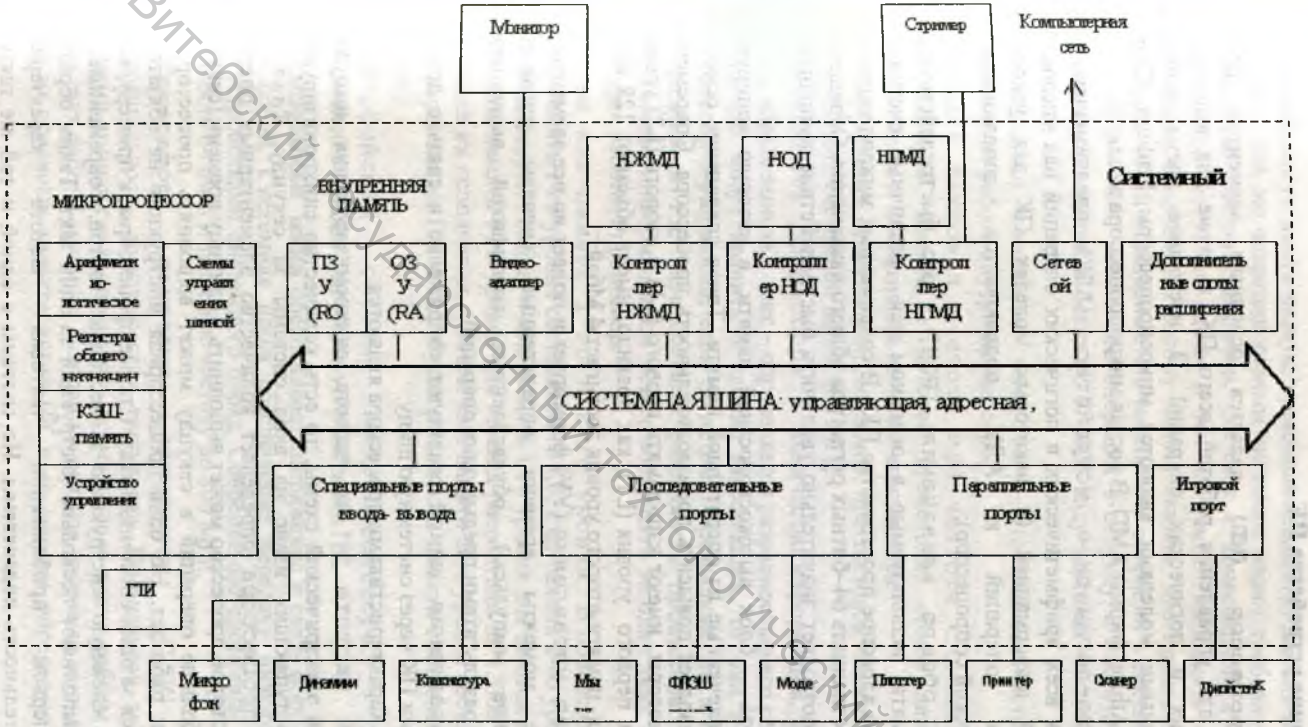


Рисунок 2 – Структурная схема ПК

Внутренние устройства ПК

Микропроцессор (МП) является основным элементом ПК и предназначен для управления работой всего ПК, а также для выполнения арифметических и логических операций. В настоящее время наиболее распространенными моделями являются микропроцессоры Pentium, Celeron фирмы Intel и Athlon фирмы AMD. В состав микропроцессора входят:

- арифметико-логическое устройство (АЛУ) предназначено для выполнения всех арифметических и логических операций над числовой и символьной информацией (в некоторых моделях ПК для ускорения выполнения операций к АЛУ подключается дополнительный математический сопроцессор);
- регистры общего назначения (РОН) – это быстродействующие ячейки памяти, используемые в основном как различные счетчики и указатели на адресное пространство ПК. В современных микропроцессорах имеется шестнадцать 64–битных регистров общего назначения. Обращение к регистрам позволяет значительно увеличить быстродействие выполняемой программы;
- кэш-память – блок высокоскоростной памяти, в которую копируются данные, извлеченные из оперативной памяти. Такое сохранение основных команд позволяет повысить производительность процессора. Современные микропроцессоры имеют кэш-память первого (L1) и второго (L2) уровней. Кэш-память первого уровня (L1), как правило, имеет объем от 128 Кбайт, емкость кэш- памяти второго уровня достигает 4 Мбайт;
- устройство управления (УУ) формирует и подает во все элементы ПК в нужные моменты времени определенные сигналы управления (управляющие импульсы), обусловленные спецификой выполняемой операции и результатами предыдущих операций;
- схемы управления шиной реализуют сопряжение и связь с другими устройствами ПК через системную шину.

Основными характеристиками процессора являются:

- Тактовая частота – это частота синхронизирующих импульсов синхронной электрической схемы, то есть количество синхронизирующих тактов, поступающих извне на вход системы за секунду. Тактовая частота процессора определяет количество элементарных операций (тактов), которое процессор может выполнить в единицу времени (секунду). Т.е. чем больше операций в секунду может выполнять процессор, тем быстрее он работает. Но одна процессорная инструкция не обязательно выполняется за один системный такт. Старые процессоры, к примеру, могли выполнять сложную инструкцию за несколько тактов, современные, как правило, выполняют несколько инструкций за один такт. Таким образом в самом первом приближении тактовая частота характеризует производительность подсистемы. Но системы с одной и той же тактовой

частотой могут иметь разную производительность, так как на выполнение одной и той же операции разным системам может требоваться различное количество тактов (например, системы с конвейерной и параллельной обработкой могут на одних и тех же тактах выполнять одновременно несколько операций). Поэтому нельзя сказать, что процессор с тактовой частотой 40 МГц выполняет 40 миллионов операций в секунду, с частотой 300 МГц — 300 миллионов операций в секунду, с частотой 1 ГГц – 1 миллиард операций в секунду. К 2003 году тактовая частота процессоров достигла 3 ГГц.

- Внутренняя тактовая частота — это тактовая частота, с которой происходит работа внутри процессора.
- Внешняя тактовая частота или частота системной шины — это тактовая частота, с которой происходит обмен данными между процессором и оперативной памятью компьютера.
- В современных процессорах, например, при тактовой частоте процессора 3 ГГц, частота системной шины 800 МГц.
- Разрядность процессора – количество информации, которое может быть обработано за 1 такт – определяется разрядностью его регистров. Если регистры могут хранить 8 единиц информации, то они 8-разрядные, и процессор 8-разрядный, если регистры 16-разрядные, то и процессор 16-разрядный и т.д. Чем большая разрядность процессора, тем большее количество информации он может обработать за один такт, а значит, тем быстрее работает процессор.
- Объем кэш-памяти 1-го и 2-го уровня. В процессоре Pentium III кэш-память 1-го уровня составляла 16 Кб, кэш-память 2-го уровня – 256 Кб. В процессорах Pentium 4 кэш-память 1-го уровня для данных имеет объем 8 Кб, кэш-память 1-го уровня для команд рассчитана на 12000 инструкций в порядке их исполнения, а объем кэш-памяти 2-го уровня составляет 512 Кб. В современных ПК и ноутбуках объем кэш-памяти составляет 1– 4 Мб (и выше).
- интерфейсная система микропроцессора, реализующая сопряжение и связь с другими устройствами ПК.

Системная шина обеспечивает сопряжение и связь всех устройств ПК между собой. Системная шина обеспечивает передачу информации между всеми устройствами системного блока. Современные системные шины имеют разрядность 64 бита и тактовую частоту до 800 МГц.

Генератор тактовых импульсов (ГТИ) генерирует последовательность электрических импульсов. Промежуток времени между соседними импульсами определяет время одного такта работы машины или просто такт работы машины. Частота генератора тактовых импульсов является одной из основных характеристик ПК и во многом определяет скорость его

работы, так как каждая операция в машине выполняется за определенное количество тактов. Современные ПК имеют тактовую частоту до 3,6 ГГц.

Внутренняя память предназначена для хранения и обмена информацией. Внутренняя память содержит два вида запоминающих устройств:

- постоянное запоминающее устройство ПЗУ (ПЗУ, в английской литературе ROM – read only memory) служит для хранения программной и справочной информации, которая не изменяется в процессе работы ПК. Это тест-мониторные программы (они проверяют работоспособность компьютера в момент его включения), драйверы и др. ПЗУ не энергозависимая память. Наряду с постоянной памятью используется энергонезависимая *полупостоянная память* (ППЗУ, CMOS), хранящая параметры конфигурации компьютера, она может быть изменена.
- оперативное запоминающее устройство (ОЗУ, RAM – random access memory) – используется для кратковременного хранения текущей информации и допускает изменение своего содержимого в ходе выполнения процессором вычислительных операций – программы пользователя, исходные, промежуточные и конечные данные. Главными достоинствами оперативной памяти являются ее высокое быстродействие и возможность обращения к каждой ячейке памяти отдельно (прямой адресный доступ к ячейке). В качестве недостатка ОЗУ следует отметить невозможность сохранения информации в ней после выключения питания машины (энергозависимость). Современные ПК имеет ОЗУ объемом до 8 Мбайт.

Видеоадаптеры (видеокарты) предназначены для подключения монитора к компьютеру. Физически они осуществляют формирование сигнала для отображения на дисплее данных и синхронизирующими сигналами – горизонтальной и (строчной) и вертикальной (кадровой) развертки. В настоящее время широко используется стандарт SVGA (Super Video Graphics Array). Видеоустройства, поддерживающие стандарт SVGA, способны отображать до 16,8 миллионов цветов и обеспечивают максимально качественное изображение, приближенное к натуральному цвету.

Контроллеры НЖМД, НГМД и НОД представляют собой специализированные устройства, обеспечивающие подключение и функционирование накопителя на жестких магнитных дисках (НЖМД – «винчестер»), накопителя на гибких магнитных дисках (НГМД – дискета) и накопителя на оптических дисках (CD-диск) соответственно.

Сетевой адаптер (сетевая плата) является устройством для подключения ПК к локальной компьютерной сети. Наиболее распространенными в настоящее время являются сетевые адаптеры, функционирующие на основе технологии Ethernet и Fast Ethernet и обеспечивающие скорости передачи данных 10 и 100 Мбит/с соответственно.

Порты (разъемы для плат расширения - интерфейсы) представляют собой совокупность программных и аппаратных средств для подключения внешних периферийных устройств: дисководов, монитора, мышки, клавиатуры, сканера, системой лазерного диска и т.д.). Конструктивно порты на системном блоке представляют собой стандартные разъемы, которые служат для соединения с компьютером. Существуют следующие основные типы портов:

- Параллельный порт – это скоростной порт, который позволяет передавать за один такт целый байт информации и применяется для быстрой связи на небольших расстояниях (LPT - Line Printer). Через параллельные порты с компьютером соединяются принтеры, стримеры и другие устройства, требующие высокую скорость передачи данных. Параллельные порты используют также для соединения двух компьютеров между собой. Скорость передачи данных через параллельный порт – от 800 Кбит/сек до 16 Мбит/сек.
- Последовательный порт (Serial port или COM-port: Communications port) – это порт, через который данные передаются только в одном направлении в каждый момент времени. За один такт последовательный порт передает один бит и, в общем случае, работает медленнее, но позволяет передавать данные на большие расстояния. Следует, однако, отметить, что современные последовательные порты типа USB и IEEE1394 превосходят по скорости параллельные, и поэтому вытесняют последние. Через последовательные порты подключаются устройства, которые не требуют высокой скорости передачи данных – мышки, клавиатуры, модемы. Скорость передачи данных через последовательный порт – 115 Кбит/сек. На схемах параллельные порты обозначают COM1, COM2 и т.д.
- USB (Universal Serial Bus) – универсальный последовательный порт. Это порт, который позволяет подключать практически любые периферийные устройства. В настоящее время производители периферийных устройств выпускают их в двух вариантах – с обычными для этих устройств портами (разными для разных устройств) и USB. Скорость передачи данных через порт USB 1.1– 12 Мбит/сек. Для мышек и клавиатуры – 1,5 Мбит/сек. Скорость передачи данных через порт USB 2 – 480 Мбит/сек.
- PS/2 порт – это параллельные порты для мышки и клавиатуры.
- AT/MIDI порт (Musical Instrument Digital Interface – соединение с цифровыми музыкальными инструментами) – это порты, через которые первоначально (до PS/2) подключались клавиатуры, а в настоящее время в основном подключаются музыкальные клавиатуры и синтезаторы.
- FireWire огненный провод – это последовательный порт, поддерживающий скорость передачи данных в 400 Мбит/сек. Этот порт служит для подключения к компьютеру видеоустройств (видеомагнитофон),

а также других устройств, требующих быстрой передачи большого объема информации, например, внешних жестких дисков.

Слоты расширения представляют собой пустые разъемы на системной плате, куда могут вставляться дополнительные внутренние устройства.

Внешние устройства ПК

Внешние устройства ПК обеспечивают взаимодействие машины с окружающей средой пользователями, объектами управления и другими ЭВМ. Внешние устройства весьма разнообразны и могут быть классифицированы по ряду признаков. В частности, по назначению можно выделить следующие виды внешних устройств:

- внешние запоминающие устройства (ВЗУ) или внешняя память ПК;
- диалоговые средства пользователя;
- устройства ввода информации;
- устройства вывода информации;
- средства связи и телекоммуникации.

Номенклатура внешних устройств все время меняется, и это связано с постоянными попытками упростить процесс общения человека с машиной.

Среди наиболее распространенных в настоящее время внешних устройств ПК отметим сканер, модем, плоттер и стример.

Сканер предназначен для считывания с бумажных носителей и ввода в ПК машинописных текстов, графиков, рисунков, чертежей. Сканеры можно классифицировать по ряду признаков: по способу использования (ручные, планшетные, пакетной обработки); по скорости обработки (количество листов в минуту); методу сканирования (однопроходный, двухпроходный), по разрешающей способности (величина dpi – количество точек на дюйм) и т.д. Сканеры, как правило, позволяют представлять информацию как в текстовом режиме с распознаванием символов, так и в графическом виде. Наиболее распространенными на рынке являются сканеры фирм XEROX, RICOH.

Модем служит для подключения ПК к глобальной компьютерной сети Интернет по телефонным линиям связи. Модемы преобразуют цифровой сигнал в аналоговый с помощью методов аналоговой модуляции. В зависимости от режимов работы различные модемы обеспечивают различные скорости передачи данных: от 1,2 Кбит/с до 56,0 Кбит/с. Наиболее популярными модемами для ПК в настоящее время являются модемы фирм 3Com и ZyXEL.

Плоттер (графопостроитель) предназначен для вывода графической информации (графиков, чертежей, рисунков) на бумажный носитель. Плоттеры бывают векторные (с вычерчиванием изображения с помощью пера, фломастера или карандаша) и растровые: термографические,

электростатические, струйные и лазерные. Конструктивно плоттеры подразделяются на планшетные и барабанные. Основные характеристиками плоттеров являются: скорость вычерчивания (100– 1000 мм/с); возможность цветного изображения и передача полутонов; разрешающая способность и четкость изображения. Лидерами на рынке плоттеров являются фирмы Hewlett Packard и Canon.

Стример (накопитель на магнитной ленте) – это устройство, которое применяется для операций резервного копирования и архивирования данных винчестера на магнитную ленту. Такое копирование происходит, как правило, в экстремальных ситуациях, когда необходимо очень быстро сохранить важную информацию с НЖМД. Все файлы, размещенные на сменной кассете, будут сохраняться без каких-либо потерь независимо от того, включен компьютер или нет. В качестве носителей информации применяются сменные кассеты различного размера с магнитной лентой. Ёмкость таких кассет составляет от 40 Мбайт до 13 Гбайт, скорость передачи данных — от 2 до 9 Мбайт в минуту, длина ленты – от 63,5 до 230 м, количество дорожек – от 20 до 144. Основными производителями стримеров являются фирмы IBM, HPQ, Dell и Tandberg.

Иерархия памяти ПК

Память ПК – это совокупность отдельных устройств, которые запоминают, хранят и выдают информацию. Отдельные устройства памяти называются запоминающими устройствами (ЗУ). Производительность ПК во многом зависит от состава и характеристик запоминающих устройств, которые в свою очередь различаются принципом действия и назначением. Основными операциями с памятью являются процедуры *записи* и *считывания* (выборки). Общее название указанных процедур носит название *обращение к памяти.* Основные характеристики памяти – это емкость и быстродействие (время обращение к памяти).

Емкость ЗУ измеряется в Байтах (1Байт = 8 Бит), Килобайтах (1 Кбайт = 2^{10} Байт), Мегабайтах (1Мбайт = 2^{10} Кбайт), Гигабайта (1Гбайт = 2^{10} Мбайт), Терабайтах (1Тбайт = 2^{10} Гбайт).

Быстродействие измеряется в секундах и в настоящее время находится в пределах от 10^{-2} до 10^{-9} секунд в зависимости от способа доступа к информации.

По способу доступа к хранящейся в них информации ЗУ делятся на: ЗУ с произвольным, прямым и последовательным доступом.

Для эффективного функционирования в ПК память строится по иерархическому принципу, где на разных уровнях иерархии находятся ЗУ, обладающие различными характеристиками. Иерархическая организация памяти позволяет повысить производительность ПК и предоставить пользователю практически неограниченную емкость памяти.

При движении от 1-го до 3-го уровня иерархии быстродействие ЗУ уменьшается, а емкость увеличивается. Иерархическая структура памяти ПК представлена на рисунке 3.

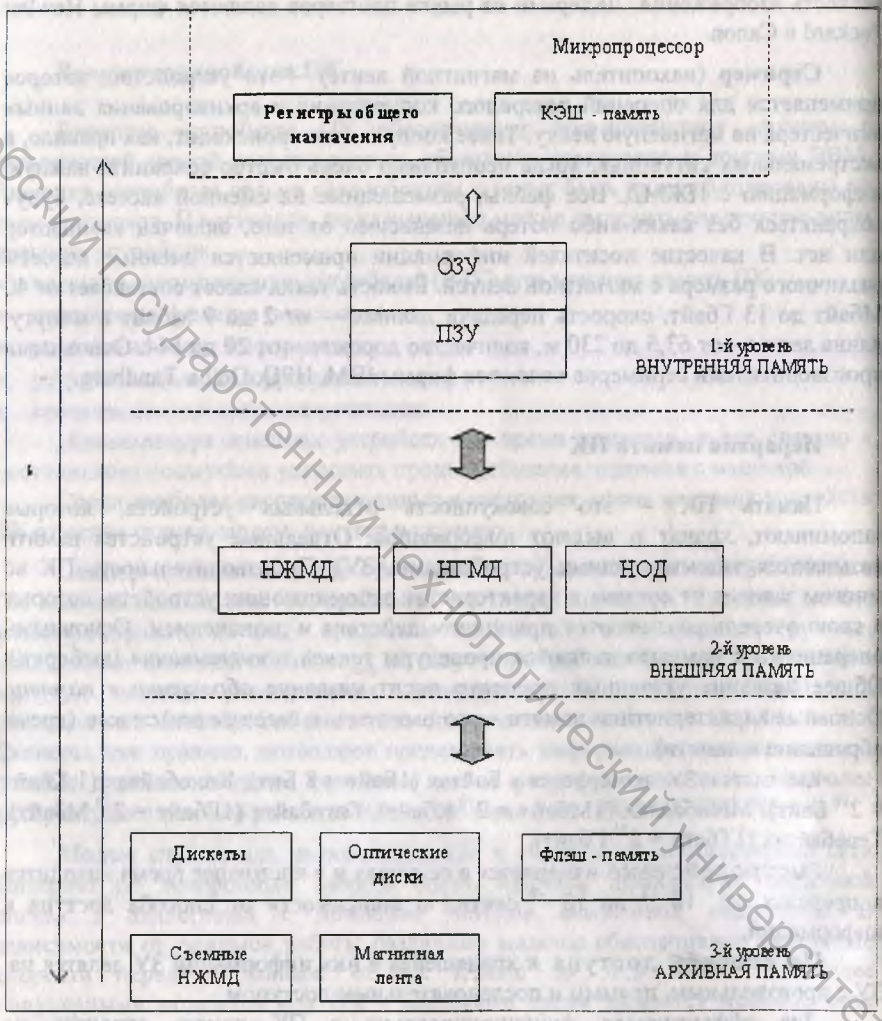


Рисунок 3 – Иерархия памяти ПК

Конфигурация ПК

Конфигурацией ПК называются состав и характеристики устройств, входящих в данный компьютер. Конфигурация подбирается в зависимости от задач, которые необходимо решать ПК.

Конфигурация ПК может быть задана следующим образом:

Intel Core 2 Duo 2 (T5870) 2ГГц (2 Мб), 15.6"(1366x768), Intel GMA X3100, RAM 2 Гб, жесткий диск: 320 Гб (5400), CDRW/DVDRW DL, WiFi, Bluetooth

В приведенной выше конфигурации можно выделить:

- процессор Intel Core 2 Duo (T5870) с тактовой частотой 2ГГц;
- КЭШ процессора 2 Мб
- экран с размером матрицы 15,6 дюймов, разрешающей способностью 1366 на 768 точек;
- графический адаптер Intel GMA X3100;
- ОЗУ DDR2 - SDRAM емкостью 2 Гб (2048 Мб)
- жёсткий диск емкостью 320 Гбайт и скоростью вращения 5400 об/мин;
- оптический привод типа CDRW/DVDRW;
- интерфейсы WiFi, Bluetooth.

Модульность, масштабируемость и стандартизуемость отдельных блоков современных ПК позволяет быстро и гибко менять его конфигурацию

Параметры, влияющие на производительность ПК

Производительность ПК является важнейшей его характеристикой. Все факторы и параметры, влияющие на производительность ПК, можно в общем случае разделить на программные *и* аппаратные.

Влияние программных факторов на производительность ПК определяется правильным выбором и настройкой как операционной системы, так и конкретных программных приложений.

Среди множества аппаратных параметров, влияющих на производительность ПК, наиболее важными являются:

- быстродействие микропроцессора – определяется тактовой частотой ГТИ;
- пропускная способность системной шины – определяется скоростью обмена с внешними устройствами ПК;
- время обращения, как к внутренним, так и внешним ЗУ;

- емкость памяти, как внутренних, так и внешних ЗУ;
- быстродействие внешних устройств, подключаемых к ПК.

Следует также отметить, что на производительность ПК большое влияние оказывает подготовленность и компетентность пользователя.

Принцип работы ПК

Большинство современных процессоров для ПК основаны на той или иной версии циклического процесса последовательной обработки информации, изобретенного Джоном фон Нейманом. Отличительной особенностью архитектуры фон Неймана является то, что инструкции и данные хранятся в одной и той же памяти.

Упрощенно основные элементы ЭВМ и связи между ними могут быть представлены следующей структурой (рис.4).

Рисунок 4 – Структурная схема ЭВМ

Из процессора на *шину адреса* (находится внутри системной шины) выдается адрес очередной команды – число, хранящееся в регистре счетчика команд.

Считанная по этому адресу команда (например из ПЗУ) поступает по *шине данных* (внутри системной шины) в процессор, где она выполняется с помощью АЛУ.

Если эта команда не является командой перехода, процессор увеличивает на единицу (в предположении, что длина каждой команды равна единице) число, хранящееся в счетчике команд, и в результате там образуется адрес следующей команды. Устройство управления процессора определяет адрес следующей выполняемой команды (фактически номер очередной ячейки памяти, где хранится следующая команда и т.д).

Снова выполняется пункт 1.

Данный цикл выполняется неизменно, и именно он называется процессом (откуда и произошло название устройства – процессор).

Во время процесса процессор считывает последовательность команд, содержащихся в памяти, и исполняет их. Такая последовательность команд называется программой и представляет алгоритм работы процессора.

Сигналы, передаваемые по управляющей шине (*шине команд*), синхронизируют работу процессора, памяти, устройств ввода-вывода информации. Порядок выбора адресов из памяти (и очередности выполнения команд) определяет программа, которая может располагаться в ПЗУ, или загружаться в ОЗУ с внешнего запоминающего устройства, по вычислительной сети или вводиться с клавиатуры.

Основы алгоритмизации инженерно-технических задач

Алгоритмы и их свойства

Понятие «алгоритм» является одним из основных в информатике. Это слово происходит от имени узбекского математика аль-Хорезми, что означает «из Хорезма». В IX веке он разработал правила арифметических действий над десятичными числами. Длительное время алгоритмами пользовались только математики, понимая под алгоритмом любое описание процесса решения задачи. Описание процессов решения математических задач предназначалось для человека, поэтому не требовалось большой строгости в описании действий.

С развитием техники появились автоматические устройства, способные воспринимать команды и исполнять соответствующие действия. Для этих устройств последовательность действий должна быть четко сформулирована. Иначе говоря, исполнителю должен быть указан алгоритм решения задачи, представленный на понятном ему языке. Под исполнителем подразумевается как человек, так и компьютер.

Алгоритм решения задачи – это конечная последовательность чётко сформулированных правил решения некоторого класса задач, приводящая к получению результата.

В обыденной жизни мы часто сталкиваемся с алгоритмами как с последовательностью действий, приводящих к достижению поставленной цели. Это правила перехода улицы, рецепты приготовления различных блюд, поиск нужного слова в словаре и др. Алгоритмы в математике – это правила выполнения арифметических действий, нахождения наибольшего общего делителя и наименьшего общего кратного, разложения числа на простые множители и др.

Для адекватности восприятия и понимания исполнителем алгоритмы должны обладать целым рядом свойств: дискретностью, точностью, понятностью, результативностью, массовостью.

- Дискретность – это разбиение алгоритма на ряд отдельных законченных действий – шагов.
- Точность – это четкое указание последовательности шагов.
- Понятность – это однозначное понимание и исполнение каждого шага алгоритма его исполнителем.
- Результативность – обязательное получение результата за конечное число шагов.
- Массовость – применимость алгоритма к решению целого класса однотипных задач.

Если бы человек знал алгоритм решения всех задач, то их исполнение можно было бы поручить машине. Но оказалось, что не все задачи, которые нам хотелось бы решить, имеют алгоритмы решения. Задачи, в принципе не имеющие общего решения, называют алгоритмически неразрешимыми. Это, например, задача о трисекции угла, о квадратуре круга и др.

Способы описания алгоритмов

Существуют различные способы представления алгоритмов. Рассмотрим некоторые из них.

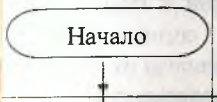
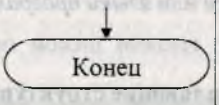
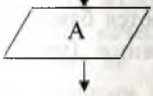
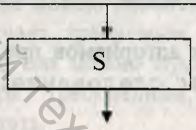
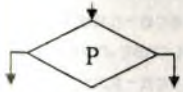
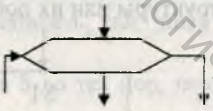
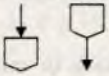
Текстовый – алгоритм решения задачи задается в виде текстового описания шагов решения. Таким способом составляются различные инструкции (например, пользование бытовыми приборами, управление автоматическими игрушками, рецепты приготовления блюд и др.). Эти алгоритмы предназначены для человека, поэтому в качестве команд могут использоваться привычные для человека предложения, фразы. Форма записи должна четко определять порядок исполнения команд.

Математический – такой способ описания алгоритма также предназначен для человека и представляет собой логически связанную последовательность математических формул, реализующих решение задачи.

В виде блок-схем – схема алгоритма представляет собой графическое изображение алгоритма на формализованном языке, описывающего содержание и логические связи задачи. Под *блоком* подразумевается любой конечный этап вычислительного процесса, принимаемый в данной схеме как целое. В блок-схеме все части вычислительного процесса представлены фигурами, содержащими пояснения и соединенными между собой линиями. Применяемые графические символы, отражающие основные операции процесса обработки данных, устанавливает ГОСТ 19.003-80 (обозначение символов соответствует международному стандарту ISO 1028-73). Правила выполнения схем алгоритмов регламентирует ГОСТ 19.002-80 (соответствует международному стандарту ISO 2636-73).

Блок-схема должна иметь начало и конец. Связь между блоками отмечается *линиями потока информации*, которые показывают последовательность решения задачи. Основное направление потока информации идет *сверху вниз* и *слева направо*. Здесь стрелки на линиях можно не указывать. Нумерация блоков производится от начала в порядке их расположения на блок-схеме. Направление линий потока, идущих *снизу вверх* и *справа налево*, показывается стрелками. По отношению к блоку линии потока могут быть входящими или выходящими. Количество входящих линий для блока принципиально не ограничено. Выходящая линия может быть только одна. Исключение составляют *логические блоки* (блоки проверки условия), имеющие не менее двух выходящих линий потока, каждая из которых соответствует одному из возможных исходов проверки логического условия. Над линиями выхода обозначается логическое условие выхода: «Да», если условие истинно, и «Нет», если условие ложно), а также блоки модификации. Наиболее часто используются блоки, представленные в таблице 1.

Таблица 1– Блоки, используемые при построении блок-схем

	Начальный блок		Конечный блок
	Блок ввода – вывода. Используется для обозначения операций обмена информацией между устройствами		Блок вычислений или функциональный блок. В этом блоке осуществляется преобразование информации по заданному действию S
	Блок проверки условия: используется для управления преобразованием информации. В результате анализа выполнения условия P выбирается одно из двух возможных направлений «да» или «нет», и управление передается блоку, записанному на выбранном направлении		Блок модификации. Используется для организации цикла с параметром. Сначала управление передается к блоку по стрелке вниз, который повторяется заданное число раз, затем управление передается к блоку, следующему за блоком модификации
	Соединитель		Межстраничный соединитель

При значительном количестве пересекающихся линий, большой их длине и многократных изменениях направления допускается разрывать линии потока информации, размещая на обоих концах разрыва *соединитель*.

Внутри каждого поля соединителя ставится одинаковая маркировка отдельной буквой или буквенно-цифровой координатой блока, к которому подходит линия потока. Если схема располагается на нескольких листах, переход линий потока с одного листа на другой обозначается с помощью *межстраничного соединителя*. При этом на листе с блоком-источником соединитель содержит номер листа и координаты блока-приемника, а на листе с блоком-приемником – номер листа и координаты блока-источника. Внутри блоков и рядом с ними проставляют записи и обозначения так, чтобы их можно было читать слева направо и сверху вниз независимо от направления потока.

Представление алгоритма на языке программирования. На основании алгоритма, сформулированного одним из перечисленных способов, программистом составляется программа решения поставленной задачи на специальном языке для записи алгоритмов. Такие языки получили название *алгоритмические языки* или *языки программирования высокого уровня*.

Основные управляющие структуры

При изображении алгоритмов при помощи схем используются базовые управляющие структуры: следование, развилка, повторение (рис.5, 6, 7).

Следование – структура, означающая, что действия S1 и S2 должны быть исполнены одно за другим. В качестве действий S1 и S2 могут выступать любые управляющие структуры или их последовательности.

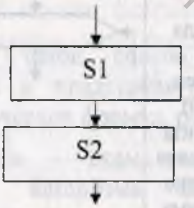


Рисунок 5 – Структура «следование»

Примером использования структуры «следование» на производстве является технологическая последовательность на изготовление изделия. Здесь каждое действие представлено в виде технологически неделимой операции, а сама последовательность есть порядок выполнения этих операций. Например, Вы не сможете проложить отделочную строчку по воротнику, предварительно его не собрав.

Развилка – это действие, осуществляющее анализ условия P (истинно или ложно), и альтернативный выбор дальнейшего направления в последовательности выполнения действий в зависимости от значения P.

Различают:

полную развилку и неполную развилку

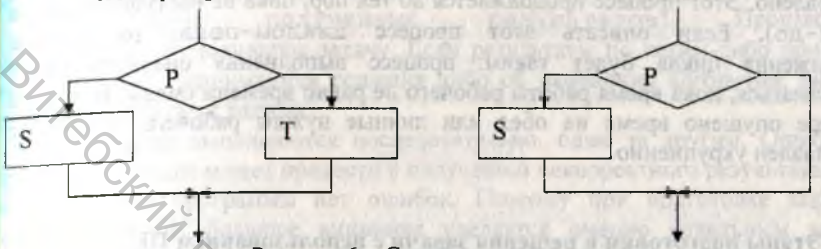


Рисунок 6 – Структура «развилка»

Словесно полная развилка описывается так: если условие P истинно, то исполнять S, иначе T.

Неполную развилку словесно можно описать так: если условие P истинно, то исполнять S.

На производстве полной развилкой можно описать процесс определения качества изделия. Если изделие отвечает требуемым показателям, то оно является качественным, соответственно его можно пускать в продажу. Если изделие не отвечает предъявляемым требованиям, то оно считается бракованным, и его необходимо переделать.

Повторение – это структура, описывающая циклические вычислительные процессы. Различают цикл-пока и цикл-до (см. рис. 7).

Цикл-пока словесно можно описать так: пока условие P истинно, выполнять тело цикла S.

Цикл-до: выполнять тело цикла S до тех пор, пока условие P не станет истинным.

Все пять алгоритмических структур могут комбинироваться одна с другой, как того требует алгоритм.

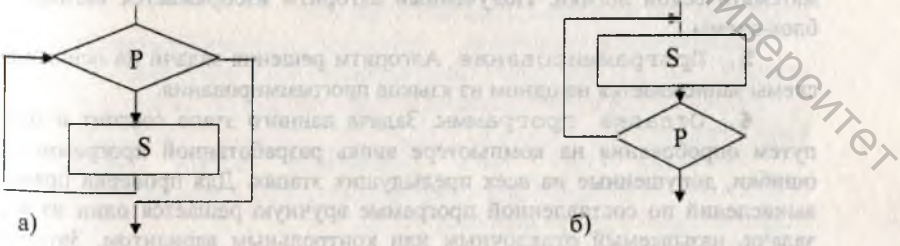


Рисунок 7 – Структура повторение: а) цикл-пока, б) цикл-до

Структурой «повторение» можно описать процесс изготовления изделия на поточной линии. Каждый рабочий выполняет определенную операцию. На потоке изготавливается несколько изделий, следовательно, рабочий должен повторить выполнение заданной операции столько раз, сколько изделий будет изготовлено. Этот процесс продолжается до тех пор, пока не наступит перерыв (цикл-до). Если описать этот процесс циклом-пока, то условие продолжения цикла будет таким: процесс выполнения операции будет продолжаться, пока время работы рабочего не равно времени смены. В данном примере опущено время на обед или личные нужды рабочего, и процесс представлен укрупненно.

Этапы подготовки и решения задачи с использованием ПК

Решение любой задачи на ПК представляет собой сложный и трудный процесс, в котором на различных этапах принимают участие различные специалисты: экономисты, инженеры, математики, программисты, статистики и т.д. Этот процесс можно условно разбить на несколько этапов.

1. Постановка задачи. На этом этапе специалист формирует цели решения задачи. Подробно раскрывает ее содержание, указывает количество и характер всех величин, используемых в задаче в качестве исходных данных.

2. Математическая формулировка задачи. Здесь устанавливаются в окончательном виде те формулы и математические зависимости, которые подлежат решению.

3. Выбор численного метода. Среди всех численных методов решения данной задачи выбирается тот из них, который наилучшим образом обеспечивает выполнение всех требований этой задачи.

4. Разработка логической схемы алгоритма и ее минимизация. Разрабатывается алгоритм решения, т.е. устанавливается необходимая последовательность арифметических и логических действий, с помощью которых может быть реализован выбранный численный метод. Если возможно, то алгоритм упрощается (минимизируется) с помощью операций и формул математической логики. Полученный алгоритм изображается наглядно в виде блок-схемы.

5. Программирование. Алгоритм решения задачи на основании блок-схемы записывается на одном из языков программирования.

6. Отладка программы. Задача данного этапа состоит в том, чтобы путем опробования на компьютере вновь разработанной программы выявить ошибки, допущенные на всех предыдущих этапах. Для проверки правильности вычислений по составленной программе вручную решается один из вариантов задачи, называемый отладочным или контрольным вариантом. Затем этот же вариант по составленной программе просчитывается на ПК. При совпадении

результатов счета на ПК и контрольного варианта считают, что программа составлена правильно.

7. Решение задачи на компьютере. После отладки программы производится непосредственное решение задачи на ПК с целью получения результатов для всех вариантов исходных данных.

8. Анализ полученных результатов. Производится специалистом, поставившим задачу. Если результаты по каким-либо причинам не устраивают, принимаются решения либо об изменении алгоритма, либо об уточнении исходных данных.

Эти этапы выполняются последовательно, один за другим. Ошибка на предыдущем этапе может привести к получению некорректного результата, даже если в тексте программы нет ошибок. Поэтому при подготовке задачи к решению на ПК большое внимание уделяется именно начальным этапам: постановке задачи и ее математической формулировке, выбору метода решения и составлению алгоритма решения задачи.

Лекция 2

ЭЛЕМЕНТЫ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ. ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА ПАСКАЛЬ

План лекции

- Характеристика и назначение структурного программирования
- Основные понятия языка ПАСКАЛЬ
  - Алфавит языка Паскаль
  - Лексемы языка Паскаль
  - Типы данных языка Паскаль
- Арифметические выражения и правила их записи

Характеристика и назначение структурного программирования

Структуризованная программа (или подпрограмма) – это программа, составленная из фиксированного множества базовых конструкций. Рассмотрим основные определения и способы образования этих конструкций в схемах алгоритмов.

Из операций, развилок и слияний строятся базовые конструкции: следование, ветвление, цикл. Применяя только эти три конструкции, можно реализовать алгоритм решения любой задачи.

Конструкция, представляющая собой последовательное выполнение двух или более операций, называется следованием.

Конструкция, состоящая из развилки, двух операций и слияния, называется ветвлением. Одна из операций может отсутствовать.

Конструкция, имеющая линии управления, ведущие к предыдущим операциям или развилкам, называется циклом.

Конструкции следование, ветвление и цикл можно представить как операции, так как они имеют единственный вход и единственный выход.

Произвольную последовательность операций можно представить как одну операцию.

Операция может быть реализована любым оператором языка ПАСКАЛЬ (простым или составным) либо группой операторов, за исключением оператора перехода GOTO.

В языке ПАСКАЛЬ количество базовых конструкций увеличено до шести, это:

-следование;
-ветвление;
-цикл с предусловием;
-цикл с постусловием;
-цикл с параметром;
-вариант.

Далее рассмотрим, как эти базовые конструкции реализуются в языке ПАСКАЛЬ.

Основные понятия языка ПАСКАЛЬ

На протяжении уже многих лет среди программистов достаточно популярен язык программирования Паскаль (Pascal). Этот язык был разработан Никлаусом Виртом первоначально для целей обучения программированию вообще, но, благодаря ряду особенностей, получил достаточно широкое распространение и в практической работе. По своей идеологии Паскаль весьма полно отражает идеи структурного программирования. Он хорошо приспособлен для применения технологии разработки программ методом нисходящего проектирования (пошаговой детализации) и предоставляет весьма гибкие возможности в отношении используемых структур данных.

Как в любом языке программирования, в Паскале присутствуют три составляющие: алфавит, синтаксис и семантика .

Алфавит – это фиксированный для данного языка набор основных символов, из которых может состоять текст на этом языке. Никакие другие символы в тексте не допускаются.

Синтаксис – это система правил, определяющих допустимые конструкции из символов алфавита. С помощью этих конструкций представляются отдельные компоненты алгоритма и алгоритм в целом.

Семантика – это система правил истолкования отдельных языковых конструкций, позволяющих однозначно воспроизвести процесс обработки данных по заданной программе.

Алфавит языка Паскаль

Алфавит языка Паскаль составляют основные символы языка – буквы, цифры и специальные символы. Паскаль включает следующий набор основных символов:

1) 26 латинских прописных и 26 латинских строчных букв:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

2) _ подчеркивание

3) 10 цифр:

0 1 2 3 4 5 6 7 8 9

4) знаки операций:

\+ - * / = <> < > <= >= := @

5) ограничители:

. , ' ( ) [ ] (. .) { } (* *) .. : ;

6) спецификаторы:

^ # $

7) служебные (зарезервированные) слова:

ABSOLUTE	EXPORTS	LIBRARY	SET
ASSEMBLER	EXTERNAL	MOD	SHL
AND	FAR	NAME	SHR
ARRAY	FILE	NIL	STRING
ASM	FOR	NEAR	THEN
ASSEMBLER	FORWARD	NOT	TO
BEGIN	FUNCTION	OBJECT	TYPE
CASE	GOTO	OF	UNIT
CONST	IF	OR	UNTIL
CONSTRUCTOR	IMPLEMENTATION	PACKED	USES
DESTRUCTOR	IN	PRIVATE	VAR
DIV	INDEX	PROCEDURE	VIRTUAL
DO	INHERITED	PROGRAM	WHILE
DOWNTO	INLINE	PUBLIC	WITH
ELSE	INTERFACE	RECORD	XOR
END	INTERRUPT	REPEAT	
EXPORT	LABEL	RESIDENT	

Кроме перечисленных, в набор основных символов входит пробел. Пробелы нельзя использовать внутри сдвоенных символов и зарезервированных слов.

Лексемы языка Паскаль

Лексема – *это минимальная конструкция языка, состоящая из символов алфавита, имеющая смысловое значение. Рассмотрим следующие виды лексем.*

1. **Слова**. Их можно подразделить на *зарезервированные* слова, *стандартные идентификаторы и идентификаторы пользователя.*

Зарезервированные слова являются составной частью языка, имеют фиксированное начертание и раз и навсегда определенный смысл. Всего зарезервированных слов около шестидесяти (*например*, `program`, `begin`, `end`).

Стандартные идентификаторы служат для обозначения заранее определенных разработчиками языка типов данных, констант, процедур и функций. (`Sin`, `Ln`).

Идентификаторы пользователя применяются для обозначения меток, констант, переменных, процедур и функций, определенных программистом.

Существуют общие правила написания идентификаторов [1]:

- идентификатор начинается только с буквы или знака подчеркивания;
- идентификатор может состоять из букв, цифр и знака подчеркивания (пробелы, точки и другие специальные символы недопустимы);
- между двумя идентификаторами должен быть хотя бы один пробел;
- максимальная длина идентификатора 127 символов, все они значимы.

Паскаль не различает идентификаторы, написанные прописными и строчными символами. Например, `SUM2, BL_6` – правильные идентификаторы; `1KL` – ошибка, идентификатор должен начинаться с цифры; `NOMER.DOMA` – ошибка, идентификатор содержит точку.

2. Константы – *это элементы данных, значения которых известны заранее и в процессе выполнения программы не изменяются.* Различают такие виды констант, как литералы, именованные и типизированные константы.

К *константам-литералам* относятся:

- десятичные числа (*например*, 69, -619, 21.14 и т. п.);
- целые шестнадцатеричные числа, которые начинаются со знака $. Например, $B, $3A и т.п.;
- логические (или булевские) константы – принимают 2 значения: True и False;
- символьные константы, например, 'A', '45', '?' и т.п.;
- строковые константы, например, 'FFF', 'DOM', 'BOX' и т.п.

Именованная константа – это идентификатор, которому должно быть присвоено значение. Такие константы должны быть описаны в разделе описания констант программы. В Паскале для описания констант используется зарезервированное слово `Const`.

Формат

```
Const <идентификатор> = <значение константы>;
```

Пример

```
Const Max = 1000;Arg = -54.69; C = 'GRUPPA';
```

В таблице 1 представлен ряд констант, к значениям которых можно обращаться без предварительного определения.

Таблица 1– Зарезервированные константы

Идентификатор	Тип	Значение	Описание
Pi	Real (вещественный)	3,141592..	Число «пи»
True	Boolean (булевский)	True	Истина
False	Boolean (булевский)	False	Ложь
Maxint	Integer (целочисленный)	32767	Максимальное целое число

Типизированная константа – это константа, для которой указывается не только значение, но и тип. Типизированной константе можно присваивать только те значения, которые определяются данным типом. *Например*,

```
Const GR: INTEGER = 2005;
```

где INTEGER – целочисленный тип.

3. Переменная – *это именованный элемент, который может изменять свое значение в процессе выполнения программы.*

Каждая переменная характеризуется *именем*, которое является идентификатором, и *типом*.

Тип констант автоматически распознается компилятором Паскаля без предварительного описания.

Тип переменных должен быть описан перед тем, как с переменными будут выполняться какие-либо действия. Для описания переменных предназначено зарезервированное слово Var.

Формат

```
Var <идентификатор>: <тип>;
```

Пример

```
Var C: integer;
    J:Char;
    A, B:Real;
```

Типы данных языка Паскаль

Понятие типа является одним из фундаментальных понятий любого языка программирования. Объекты (константы, переменные, функции, выражения), которыми оперирует программа, относятся к определенному типу.

Тип – *это множество значений, которые могут принимать объекты программы, и совокупность операций, допустимых над этими значениями [1].*

Например, значения 1 и 2 относятся к целочисленному типу, их можно складывать, умножать и выполнять другие арифметические операции. Значения «монитор» и «Паскаль» носят лингвистический характер, они имеют свой набор допустимых операций. В большинстве широкоупотребительных языков могут использоваться только строго определенные, заранее известные типы. Pascal, наряду со стандартными типами, имеющимися в других языках высокого уровня, позволяет программисту образовывать собственные типы.

Все допустимые в языке Паскаль типы подразделяются на две большие группы: *простые и сложные (структурированные).*

Простые типы в свою очередь подразделяются на *скалярные и ограниченные.*

Скалярные типы подразделяются на *стандартные и определенные пользователем.*

Стандартные типы предлагаются пользователям разработчиками языка Паскаль. К ним относятся целочисленные, вещественные, литерные, булевские типы данных и указатели.

Целочисленные типы данных представляют собой значения, которые могут использоваться в математических выражениях. Данные целочисленных типов могут быть представлены как в десятичной, так и в шестнадцатеричной

системе, для записи которой используется знак $. Характеристика целочисленных типов данных представлена в таблице 2.

Таблица 2 – Характеристика целочисленных типов данных

Тип	Диапазон	Требуемая память (байт)
BYTE	0..255	1
SHORTINT	-128..127	1
INTEGER	-32768..32767	2
WORD	0..65535	2
LONGINT	-2147483648..2147483647	4

Очевидно, что использование переменной типа `Byte` по сравнению с типом `Integer` экономит память в два раза. Использование байтового типа целесообразно, если известно, что значение переменной не превысит 255. В арифметических и логических выражениях допустимо смешение типов `byte` и `integer`.

Пример

```
Var Sort, Nomer: integer;
    A, B, C: integer;
    Gran: byte; Min, Max: byte;
```

Вещественные типы данных представляют собой вещественные значения, которые используются в арифметических выражениях и занимают в памяти от 4 до 6 байт. Паскаль допускает представление вещественных значений с плавающей или фиксированной точкой. Вещественные десятичные числа с *фиксированной точкой* записываются по обычным правилам арифметики. Целая часть отделяется от дробной десятичной *точкой*. Если десятичная точка отсутствует, число считается целым. Вещественные десятичные числа с *плавающей точкой* представлены в экспоненциальном виде: mE+p, где m – мантисса (целое или дробное число с десятичной точкой), E обозначает «10 в степени», p – порядок (целое число). Характеристика вещественных типов данных представлена в таблице 3.

Таблица 3 – Характеристика вещественных типов данных

Тип	Диапазон	Мантисса, знаков	Требуемая память (байт)
REAL	2.9*10E-39..1.7*10E38	11-12	6
SINGLE	1.5*10E-45..3.4*10E38	7-8	4
DOUBLE	5.0*10E-324..1.7*10E308	15-16	8
EXTENDED	1.9*10E-4951..1.1*10E4932	19-20	10
COMP	-2E+63+1..2E+63-1	10-20	8

Эффективное использование типов `SINGLE`, `DOUBLE`, `EXTEND`, `COMP` возможно только при включенной директиве {$N+}. По умолчанию она

находится в выключенном состоянии. Для решения инженерно-экономических задач достаточно значений типа REAL.

Пример

```
Var Res, Summa, Itog: real;
    X, Y: real;
```

Булевский тип данных описывается идентификатором BOOLEAN. Переменные и константы этого типа могут принимать только одно из двух значений: TRUE (истина) или FALSE (ложь).

Пример

```
Var Sel1, Sel2: boolean;
A,B,C,D: boolean;
```

Выражения булевского типа занимают в памяти 1 байт и используются в логических выражениях и выражениях отношения, а также для управления порядком выполнения операторов программы.

Литерный (символьный) тип описывается стандартным идентификатором CHAR. Константы и переменные этого типа могут принимать одно из значений кодовой таблицы ASCII. Значение константы или переменной этого типа заключается в апострофы.

Например,

```
Var Bukva, Znak, Simvol: char;
    Bukva:='A'; Znak:='+'; Simvol:='!'
```

Переменные символьного типа занимают в памяти 1 байт. Использование данных типа char в арифметических выражениях запрещено. К литерным значениям могут применяться операции сравнения, результат при этом зависит от номера литерной переменной или константы в кодовой таблице.

Кроме стандартных типов данных, Паскаль поддерживает скалярные типы, *определенные пользователем*. К ним относятся *перечисляемый* и *интервальный* типы. Данные этих типов занимают в памяти 1 байт, поэтому любой пользовательский тип не может содержать более 255 элементов. Их применение значительно улучшает наглядность программы, делает более легким поиск ошибок и экономит память.

Перечисляемый тип задается непосредственно перечислением всех значений, которые может принимать переменная данного типа. Отдельные значения указываются через запятую, а весь список заключается в круглые скобки.

Формат

```
Type <имя типа>=(<значение1, значение2, ..., значениеN>);
Var <идентификатор, ...>: < имя типа>;
```

Пример

```
Type Season =(Spring, Summer, Autumn, Winter);
Var S1, S2: Season;
    Autumn: (September, October, Nowember);
```

В данном примере приведен явно описанный тип данных пользователя Season. Определены их значения – обозначения времен года. Переменные S1 и S2 могут принимать только одно из перечисленных значений. Попытка присвоить им любое другое значение вызовет программное прерывание. Третий тип перечисления – анонимный (не имеет имени) и задается перечислением значений в разделе Var. Autumn является переменной этого типа и может принимать значения September, October, Nowember. Таким образом, может быть задан любой тип, но это не всегда приемлемо. Первый способ, безусловно, более понятен и больше соответствует характеру языка Pascal.

Интервальный тип позволяет задавать две константы, определяющие границы диапазона значений для данной переменной. Компилятор при каждой операции с переменной интервального типа генерирует подпрограммы проверки, определяющие, остается ли значение переменной внутри установленного для нее диапазона. Обе константы должны принадлежать одному из стандартных типов, кроме вещественного. Значение первой константы должно быть обязательно меньше значения второй.

Формат

```
Type <имя типа> = <константа1> .. <константа2>;
    Var <идентификатор>: < имя типа>;
```

Пример

```
Type Days = 1.. 31;
  Var Work_d, Free_d: Days;
```

В этом примере переменные *Work_d, Free_d* имеют тип *Days* и могут принимать любые значения из диапазона 1 . . 31.

Выход из диапазона вызывает программное прерывание.

Можно определить интервальный тип, задав границы диапазона не значениями констант, а их именами:

```
Const Min = 1; Max = 31;
Type Days = Min .. Max;
  Var Work_d, Free_d: Days;
```

Структурированные типы данных базируются на скалярных типах и могут содержать их различные комбинации. Они определяют упорядоченную совокупность скалярных элементов и характеризуются типом своих компонентов [1]. В языке Паскаль представлены следующие структурированные типы данных:

строка – последовательность символов, заключенная в апострофы;

массив – структурированный тип данных, состоящий из фиксированного количества элементов одного и того же типа, доступ к которым осуществляется по индексу;

множество – набор выбранных по какому-либо признаку или группе признаков объектов, которые можно рассматривать как единое целое;

запись – совокупность фиксированного числа компонентов разного типа;

файл – последовательность компонентов одного типа и одной длины.

Еще двум структурированным типам – процедурному и типу object (объектному) – трудно поставить в соответствие данные в обычном представлении.

Рисунок 1– Набор основных типов языка Паскаль

Арифметические выражения и правила их записи

Арифметическое выражение представляет собой набор операндов, соединенных знаками арифметических операций, задающими действия над элементами данных и порядок их выполнения, и возвращает целое или действительное значение. Операндами могут являться константы, переменные и обращения к функциям.

Существует пять категорий операций, упорядоченных по убыванию старшинства:

1) – (унарный минус), @ - (взятие адреса объекта);

2) операция NOT;

3) операции типа умножения : *, /, DIV, MOD, AND, SHL, SHR (см. таблицу 4);

4) операции типа сложения: +, OR, XOR (см. таблицу 5);

5) операции сравнения: =, <>, >, <, >=, <=.

Таблица 4 – Операции типа умножения

Обозначение	Наименование	Тип операторов	Тип операндов
*	умножение	real	real
		integer	integer

		real, integer	real
/	деление	real, integer	real
DIV	целое деление	integer	integer
MOD	остаток от деления	integer	integer
AND	арифметическое И	integer	integer
	логическое И	boolean	boolean
SHL	сдвиг побитовый влево	integer	integer
SHR	сдвиг побитовый вправо	integer	integer

Таблица 5 – Операции типа сложения

Обозначение	Наименование	Тип операторов	Тип операндов
+	сложение	real	real
		integer	integer
		real, integer	real
-	вычитание	real	real
		real, integer	real
		integer	integer
or	арифметическое ИЛИ	integer	integer
	логическое ИЛИ	boolean	boolean
xor	арифметическое побитовое сложение по модулю 2	integer	integer
	логическое исключающее ИЛИ	boolean	boolean

Функция `DIV` возвращает целую часть от деления числа нацело, а функция `MOD` – остаток от деления.

Например,

`56 DIV 5 = 11` – выделяет целую часть полученного частного;

`56 MOD 5 = 1` – выделяет остаток от деления нацело. В арифметических выражениях могут использоваться стандартные функции, представленные в таблице 6.

Таблица 6 – Стандартные функции языка Паскаль

Математическая запись	Запись на Паскале	Примечание
Sin x	`Sin(X)`	
Cos x	`Cos(X)`	
Ln x	`Ln(X)`	x>0
$\sqrt{x}$	`Sqrt(X)`	$x \geq 0$
e^x	`Exp(X)`	
x^2	`Sqr(X)`	
Arctg x	`Arctan(X)`	$-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}$
$\|x\|$	`Abs(X)`	

$tg(x) = sin(x)/cos(x)$

$ctg(x) = cos(x)/sin(x)$

$Arc\sin(x) = Arctg(\sqrt{x/(1-x^2)})$

$Arc\cos(x) = \pi/2 - Arc\sin(x)$

$Arcctg(x) = \pi/2 - Arctg(x)$

Аргументами этих функций могут быть данные любых числовых типов. Кроме них, при решении задач можно использовать следующие функции [3]:

- получение целой части:
  - `Trunc(X)` – результат имеет целочисленный тип;
  - `Int(X)` – результат имеет вещественный тип;
- получение дробной части: `Frac(X)` – результат вещественного типа;
- округление до целого: `Round(X)` – результат имеет вещественный тип;
- `Pred(X)` – возвращает элемент, предшествующий X в списке значений типа. Тип результата совпадает с типом параметра;
- `Succ(X)` – возвращает значение, следующее за X в списке значений типа. Тип результата совпадает с типом параметра.
- получение значения случайного числа:
  - `Random` – генерирует случайное значение из диапазона 0..0.99, результат имеет вещественный тип;
  - `Random(X)` – генерирует случайное значение из диапазона от 0 до X, результат имеет целочисленный тип.
  - `Randomize` -- процедура используется для обновления базы генерации.

Для определения четности/нечетности числа используется функция `ODD`. Ее аргументом может быть переменная только целого типа.

`ODD(X)` – возвращает значение булевского типа, равное `True`, если X – нечетное и `False`, если X – четное.

Например, `ODD(3)` – возвращает значение `True`, так как аргумент нечетный, `ODD(4)` – вернет значение `False`, `ODD(2+7-1)` – также вернет значение `False`.

Если в выражениях имеются нестандартные функции, они должны быть выражены через стандартные, используя правила математики.

$a^x \Rightarrow e^{x*\ln a} \Rightarrow \exp(x*\ln(a))$

$\log_a x \Rightarrow \ln x/\ln a \Rightarrow \ln(x)/\ln(a)$

$\lg x \Rightarrow \ln x/\ln 10 \Rightarrow \ln(x)/\ln(10)$

$tgx \Rightarrow \sin x/\cos x \Rightarrow \sin(x)/\cos(x)$

Пример

$\sqrt[5]{ax+2} \Rightarrow (ax+2)^{\frac{1}{5}} \Rightarrow \frac{1}{5}*\ln(ax+2) \Rightarrow e^{\frac{1}{5}*\ln(ax+2)} \Rightarrow \exp(\frac{1}{5}*\ln(ax+2))$

$e^{\sqrt{2x+4}} = \exp(sqrt(2x+4))$

Лекция 3

ПРОСТЫЕ ОПЕРАТОРЫ ЯЗЫКА ПАСКАЛЬ

План лекции

- Характеристика разделов программы
- Классификация операторов языка Паскаль
  - Простые операторы
  - Операторы ввода-вывода
- Пример

Характеристика разделов программы

Любая программа реализует алгоритм решения задачи. Она объединяет последовательность действий, выполняемых над определенными данными с помощью определенных операций для реализации определенной цели. Основными характеристиками программы является точность полученного результата, время выполнения и объем требуемой памяти.

Синтаксически программа на языке Паскаль состоит из двух частей: *заголовка*, который не является обязательным, и *блока*, который может содержать в себе другие блоки. Блок, в свою очередь, состоит из двух частей: *раздела описаний* и *раздела операторов*. Первая часть может отсутствовать, без второй блок не имеет смысла.

Блок, который не входит ни в какой другой блок, называется *глобальным*. Если в глобальном блоке находятся другие блоки, они называются *локальными*. *Глобальный блок* – это основная программа, он должен присутствовать в любом случае. Локальные блоки – это процедуры и функции, их присутствие необязательно.

В начале программы находится *заголовок*, который в общем случае состоит из зарезервированного слова `Program`, имени программы и параметров, с помощью которых программа взаимодействует с операционной системой. Заголовок программы несет чисто смысловую нагрузку и может отсутствовать. Имя программы является идентификатором и формируется по соответствующим правилам.

Формат заголовка программы
`Program <имя программы>;`
Например, Program Lin;

После заголовка следует *программный блок*, состоящий в общем случае из *разделов описаний* подключаемых библиотечных модулей, меток, констант, типов данных, переменных, процедур и функций, и *раздела операторов*.

Любой раздел, кроме раздела операторов, может отсутствовать. Разделы описаний могут встречаться в программе любое количество раз и следовать в любом порядке. Главное, чтобы все описания объектов программы были сделаны до того, как они будут использованы.

Охарактеризуем разделы программного блока.

■ **Раздел описания подключаемых библиотечных модулей**

Характерной чертой языка Паскаль является наличие большого количества стандартных процедур и функций (общее название – подпрограммы). Все они находятся в библиотечных модулях. *Библиотечный модуль* – это результат компиляции одной или нескольких процедур и функций. Модуль имеет имя, упомянув которое в разделе `Uses`, можно получить доступ к каждой из находящихся в нем процедур или функций.

Например, язык Турбо Паскаль имеет следующие стандартные модули: `System, Dos, Overlay, Graph, Crt, Printer`, которые хранятся в файле `TURBO`.

В модуле `Crt` содержатся стандартные процедуры управления экраном, а модуль `Printer` служит для управления работой принтера.

Рассмотрим некоторые процедуры модуля `CRT`, которые будут использоваться при написании программ:

1. `CLRSCR` – очистка экрана. Курсор перемещается в левый верхний угол экрана.

2. `GOTOXY(N1, N2)` – перемещение курсора в заданную точку экрана. Здесь `N1`, `N2` – параметры:

- N1 – номер столбца на экране дисплея, целое из диапазона от 1 до 80;
- N2 – номер строки на экране дисплея, целое из диапазона от 1 до 25;

Например, `GOTOXY(20,10)` – перемещение курсора в 10 строку и 20 столбец.

3. `DELAY(T)` – задержка выполнения программы на T милисекунд. *Например*, чтобы сделать задержку на 2 сек, следует записать `DELAY (2000)`.

Интегрированная среда разработки `Delphi` имеет свои стандартные модули: `SysUtils, Windows, Graphics` и некоторые другие. Подробнее с назначением и составом каждого из них можно ознакомиться в справочной системе Delphi.

Пример

```
Uses SysUtils, Windows;
```

В модуле SysUtils содержатся процедуры, функции, объекты для обработки и управления разными типами данных, организации и обработки файлов, управления работой программы.

В модуле Windows содержатся библиотеки с набором API- функций ОС, в частности, процедура SetConsoleoutputCP(N), которую следует использовать для отображения русских шрифтов (в формате SetConsoleoutputCP(1251),где 1251 – код страницы, в которой находится кириллица).

Подробнее эта информация изложена в справочной системе программной среды Delphi.

Раздел описания подключаемых библиотечных модулей всегда расположен после заголовка программы.

Формат

```
Uses<имя модуля1>,<имя модуля2>, …;
```

- **Раздел описания меток**

Перед любым оператором языка Паскаль можно поставить метку, что позволит выполнить обращение к этому оператору с помощью оператора перехода goto из любого места программы. Метка состоит из имени и следующего за ним двоеточия. Именем может служить идентификатор или цифра.

Используемые метки должны быть описаны в разделе Label.

Формат

```
Label <имя, . . . >;
```

Пример

```
Label M1, M2, 4, Blok2;
```

Далее в тексте программы выполняется обращение к меткам с использованием оператора Goto.

```
Goto M1;
. . .
M1:  Y:=2*SIN(X);
```

Здесь M1 – это метка оператора.

- **Раздел описания констант**

В этом разделе производится присваивание идентификаторам констант постоянных значений.

Формат

```
Const < идентификатор > = < значение>;
```

Пример

```
Const R1=22;
      St=3.25;
      R2:Integer=4;
```

После того, как константа определена, ей нельзя присвоить какое-либо другое значение. Ряд констант (`True, False, Maxint`) определен стандартно, к ним можно обращаться без предварительного описания.

- **Раздел описания типов данных**

Этот раздел включается в программу тогда, когда программист использует в программе свои собственные типы. Стандартные типы не требуют описания. Синтаксис языка Паскаль не требует обязательного определения идентификатора типа, так как значения типа всегда можно задать перечислением в разделе описания переменных.

Формат

```
Type < имя типа > = < значения типа >;
```

Пример

```
Type T1=1..25;
     T2=Array[1..10] of real;
```

- **Раздел описания переменных**

Каждая встречающаяся в программе переменная должна быть описана. Раздел описания переменных включает в себя описание всех переменных, встречаемых в программе.

Формат

```
Var <идентификатор, . . .>: < тип >;
```

Пример

```
Var A,B,X : real;
    Z,Y : integer;
```

- **Раздел описания процедур и функций**

Стандартные процедуры и функции в программе не описываются. В разделе описания процедур и функций должны быть приведены тексты только тех процедур и функций, которые разработаны самим программистом и используются в данной программе.

Формат

Procedure <имя процедуры>{(список формальных параметров)};
Function <имя функции>{(список формальных параметров)}:<тип>;

Пример
Procedure Sum(a,b:real; Var S:real);
Function Sum(a,b:real):real;

Подробнее процедуры и функции пользователя будут рассмотрены далее.

.

■ **Раздел операторов**

Раздел операторов является основным, так как именно в нем описывается алгоритм решения задачи. Этот раздел начинается со слова `Begin` и заканчивается словом `End` с точкой (`End.`), между которыми располагаются строки операторов, описывающих, в каком порядке выполняются действия над данными, чтобы получить требуемый результат. В конце каждого оператора ставится точка с запятой («;»).

Формат:

```
Begin
   < оператор;>
     . . .
   < оператор;>
End.
```

Операторы языка Паскаль

Оператор – это конструкция языка, которая определяет действия, выполняемые над данными, или порядок выполнения этих действий. Операторы выполняются последовательно в том порядке, в котором они записаны в тексте программы. Разделителем операторов служит точка с запятой. Все операторы языка Турбо Паскаль разделяются на три группы:

1. Простые операторы.
2. Операторы ввода-вывода, которые на самом деле являются процедурами, так как в них можно указывать параметры.
3. Структурированные (сложные) операторы, в состав которых включаются другие операторы по строго определенным правилам.

1 Простые операторы

Это операторы, не содержащие в себе никаких других операторов. К ним относятся *операторы присваивания, безусловного перехода, вызова процедуры и пустой оператор.*

Оператор присваивания (:=) предписывает выполнить выражение, стоящее в его правой части, и присвоить результат идентификатору переменной, стоящей в левой части. Переменная и выражение должны иметь

один и тот же тип. Исключение представляет случай, когда переменная имеет вещественный тип, а выражение – целочисленный. Допустимо присваивание любых типов данных, кроме файловых.

Формат

```
<идентификатор>:=<выражение>;
```

Пример

```
Sort:=1;
Cena:=15.23;
Nazv:='Model N986';
Res:=Sin(A)+Cos(B);
```

Оператор безусловного перехода GOTO означает «перейти к» и применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а какой-нибудь другой, отмеченный меткой оператор. Метка может содержать как цифровые, так и буквенные символы.

Формат

```
GOTO <метка>;
```

Пример

```
...
Label Metka1, Metka2;
...
Metka1: GOTO Metka2;
Metka2: GOTO Metka1;
...
```

При записи оператора `GOTO` необходимо помнить следующее. Метка, на которую передается управление, должна быть описана в разделе описания меток того блока (процедуры, функции, основной программы), в котором эта метка используется. Областью действия метки является блок, в котором она описана. Попытка выйти за пределы блока вызывает программное прерывание.

Обычно оператор `GOTO` применяется для преждевременного выхода из цикла или при отладке программы.

Пустой оператор не содержит никаких символов и не выполняет никаких действий. Он может быть расположен в любом месте программы, где синтаксис языка допускает наличие оператора. Как и все другие операторы, пустой оператор может быть помечен меткой. Чаще всего пустой оператор используется для организации выхода из середины программы или составного оператора.

Пример

```
Begin
  Goto Metka;        {переход в конец блока}
. . .
   Metka;            {пустой оператор помечен меткой}
End.
```

Оператор вызова процедуры служит для активизации предварительно определенной пользователем или стандартной процедуры.

Формат

```
<имя процедуры> {(список параметров)};
```

Пример

```
Program Prim;
Procedure V1;
Begin
. . .                          {тело процедуры V1}
end;
Procedure V2;
Begin
. . .                          {тело процедуры V2}
end;
Procedure Sum(a,b:real; Var S:real);
Begin
. . .                          {тело процедуры Sum}
end;

Begin
V1;                 {вызов для выполнения процедуры V1}
V2;                 {вызов для выполнения процедуры V2}
Sum(x,y,S_xy);      {вызов для выполнения процедуры Sum}
End.
```

2 Операторы ввода-вывода

Решение самой простой задачи не обходится без операций ввода-вывода информации.

Ввод данных – это передача информации от внешнего носителя в оперативную память для обработки.

Вывод данных – обратный процесс, когда данные передаются после обработки из оперативной памяти на внешний носитель.

Для выполнения операций ввода используются операторы `Read` и `Readln`.

Формат

```
Read(список ввода);
```

где в списке ввода могут фигурировать только переменные допустимых типов данных.

Пример

```
Var I: integer;
    J: integer;
    K: char;
Begin
   read(I, J, K);
```

Оператор чтения `Readln` аналогичен оператору `Read`. Отличие состоит в том, что после считывания последнего в списке значения для одного оператора `Readln` данные для следующего оператора `Readln` будут считываться с начала новой строки.

Операторы записи `Write` и `Writeln` производят вывод числовых данных, символов, строк, булевских значений.

Формат

```
Write(список вывода);
```

где список вывода представляет собой набор переменных и выражений.

Например, `Write(y1, y2, sin(3), yn)` – *выражения типа integer,* `byte`, `real`, `char`, `boolean` и т.д.

Для вывода переменных *целого* и *символьного* типов используется следующий формат: `Write(X:P)`, где параметр P определяет, сколько знакомест на экране будет выделено для вывода значения той или иной переменной.

Например, `Write(X:3,C:7)`

Для переменных *вещественного* типа используется обычно следующий формат: `Write(X:P:q)`, где P означает общее число позиций на экране, выделенное для данной переменной, а q – число позиций (из общего числа) после десятичной точки.

Например,

`Write(X:3:1, C:7:4)` – правильный формат,

`Write(X:1:3, C:7:6)` – неправильный формат.

Для вывода результирующей информации на экран иногда требуется вывести текстовую информацию, поясняющую результат.

Пример

```
Write('Сумма ряда =', S:7:3, 'при X=',X:4:2);
```

Оператор `Writeln` аналогичен оператору `Write`, но после вывода последнего в списке значения происходит перевод курсора к началу следующей строки. Оператор `Writeln` без параметров вызывает пропуск пустой строки.

Знание рассмотренных операторов дает возможность программирования алгоритмов линейной структуры.

Пример

Вычислить значение функций

$$a = arctg(x) - \frac{3}{5}e^{x^y} + 0.5\frac{|x+y|}{(x+y)^c} \qquad b = \frac{e^{|x-y|} \cdot tg(z)}{arctg(y) + \sqrt{x}} + \ln x$$

Текст программы:

```
Program Lin1;
{$APPTYPE CONSOLE} -
 Uses SysUtils, Windows;
 Const c=1.2;
 Var x,y,z,a,b1,b2,b:real;
 Begin
   SetConsoleoutputCP(1251);
   Writeln('Введите x,y,z');
   Readln(x,y,z);
   a:=arctan(x)-3/5*exp(x-y)+0.5*abs(x+y)/exp(c*ln(x+y));
   b1:=exp(abs(x-y))*sin(z)/cos(z);
   b2:=arctan(y)+sqrt(x);
   b:=b1/b2+ln(x);
   Writeln('При  x=',x:0:1,'   y=',  y:0:2,'   результаты:
a=',a:0:2,'  b=',b:0:2);
   Sleep(7000);
End.
```

Пояснения к программе:

Директива компилятора `{$APPTYPE CONSOLE}` указывает на генерацию консольного приложения.

Процедура `SetConsoleoutputCP(1251)` из модуля `Windows` используется для правильного отображения русских шрифтов.

Процедура `Sleep(7000)` из модуля `SysUtils` используется для задержки на экране результата на 7 сек.

Лекция 4

СТРУКТУРНЫЕ ОПЕРАТОРЫ ЯЗЫКА ПАСКАЛЬ

План лекции

- Составной оператор
- Условные операторы
  - Оператор условия `IF`
  - Оператор выбора `CASE`
- Операторы повтора
  - Оператор повтора `For`
  - Оператор повтора `Repeat`
  - Оператор повтора `While`

Для программирования сложных алгоритмических структур, таких, например, как ветвления и циклы, необходимо иметь представление о структурных операторах. *Структурные операторы* представляют собой структуры, построенные из других операторов по строго определенным правилам. Все структурные операторы подразделяются на три группы: *составные*, *условные* и *повтора*.

Составной оператор

Составной оператор представляет собой группу из произвольного числа операторов, отделенных друг от друга точкой с запятой и ограниченную операторными скобками `Begin ... End`.

Формат

```
begin
    <оператор;>
    <оператор;>
    . . .
    <оператор>
end;
```

Пример

Записать, используя составной оператор, выражение $R := \sin(\pi \cdot x) + e^{2x}$.

```
Begin
```

```
    A:= Sin(Pi*x);
    B:= exp(2*x);
    Rez:= A + B;
end;
```

Составной оператор может находиться в любой части программы и воспринимается как единое целое. Обычно используется при организации разветвлений или циклов.

Условные операторы

Условные операторы используются для реализации разветвляющихся алгоритмов.

Разветвляющимся называется алгоритм, который содержит несколько ветвей, отличающихся друг от друга содержанием вычислений. Выход вычислительного процесса на ту или иную ветвь алгоритма определяется исходными данными задачи.

В разветвляющихся вычислительных процессах отдельные этапы вычислений (операторы) выполняются не всегда в одном и том же порядке. В зависимости от условий, проверяемых по ходу вычислений, выбираются для исполнения различные их последовательности. Поэтому в программе должно содержаться указание о том, в каком случае надо выбирать для исполнения тот или иной оператор. Это указание формулируется в виде отношения, например, условия $x > y$. Если это условие истинно, то выполняется соответствующий этому условию оператор, в противном случае, если условие ложно, выполняется другой оператор.

Для задания подобного рода разветвляющихся вычислительных процессов служат условные (выбирающие) операторы, которые относятся к числу производных операторов. Такой оператор обеспечивает выполнение или невыполнение некоторого оператора, группы операторов или блока в зависимости от заданных условий. Pascal допускает использование двух условных операторов IF и CASE.

Оператор условия IF является одним из самых популярных средств, изменяющих естественный порядок выполнения операторов программы. Он может принимать одну из следующих форм:

полная форма

```
If <условие> Then <оператор1>
   Else <оператор2>;
```

Если условие истинно, выполняется оператор1, в противном случае оператор2;

неполная форма

```
If <условие> Then <оператор>;
```

Если условие истинно, выполняется оператор, стоящий за Then, в противном случае – выполняется оператор, стоящий за If.

Условие может быть простым и сложным. В последнем случае используются логические операции AND, OR, NOT,

например, If (A>3) And (A<5).

Блок-схемы полной и неполной условной конструкции подробно рассмотрены в в лекции 1.

Пример

```
If X>0 Then Writeln('X больше 0')
       Else Writeln ('X меньше или равно 0');
```

Перед Else точка с запятой не ставится. Если после Then или Else необходимо выполнить не один, а несколько операторов, они оформляются как составной оператор, т.е. заключаются в операторные скобки Begin ... End.

Пример

```
If X<=A Then Begin
                Y:= A*X;
                Z:= Ln(X)+Y;
             End;
```

Один оператор IF может входить в состав другого оператора IF. Такие конструкции называют вложенными. Допускается до 7 вложений оператора IF друг в друга.

Оператор выбора CASE является обобщением оператора IF и позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого *селектором*, и списка параметров, каждому из которых предшествует список констант выбора. Как и в операторе IF, здесь может присутствовать слово ELSE, имеющее тот же смысл. Блок-схема оператора Case представлена на рисунке 1.

Формат

```
Case <выражение-селектор> of
     <список1>: <оператор1;>
     <список2>: <оператор2;>
     . . .
     <списокN>: <операторN>
     else <оператор>
end;
```

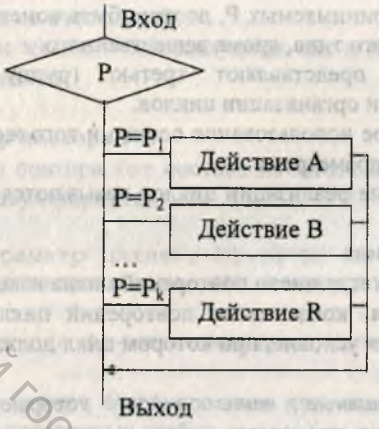


Рисунок1 – Блок-схема оператора Case

Эта структура обеспечивает выбор одного из ряда возможных действий в зависимости от значения, которое принимает выражение (условие) P.

Пример

Селектор целочисленного типа	Селектор интервального типа

Var *I*:integer;
Begin
. . .
Case *I* of
1:Z:=A+B;
2:Z:=A+10;
3:Z:=A+100;
end;
	Var *I*:integer;
	Begin
	. . .
	Case *I* of
	1..2: Writeln('WINTER');
	3..5: Writeln('SPRING');
	6..8: Writeln('SUMMER');
	9..11: Writeln('AUTUMN');
	12: Writeln('WINTER')
	end;

Селектор литерного типа	Селектор перечисляемого типа

Var *Name*:Char;
Begin
. . .
Case *Name* of
'A','a': Writeln('Модуль');
'S','s' :Writeln('Синус');
'C','c': Writeln('Косинус');
end;
	Var
	Season:(Winter,Spring,Summer
	Autumn);
	Begin
	. . .
	Case *Season* of
	Winter: Writeln('WINTER');
	Spring: Writeln('SPRING');
	Summer: Writeln('SUMMER');
	Autumn: Writeln('AUTUMN');
	end;

Количество значений, принимаемых P, должно быть конечным. Селектор P может быть любого скалярного типа, кроме вещественного.

Операторы повтора представляют третью группу структурных операторов и используются при организации циклов.

Цикл – это многократное использование одного и того же вычисления с изменяющимися исходными параметрами.

Конкретные программные реализации циклов называются циклическими программами.

Различают два типа циклов:

- **цикл со счетчиком**, когда число повторений цикла известно заранее;
- **итерационный цикл,** когда число повторений цикла не известно заранее. В этом случае задается условие, при котором цикл должен закончиться или, наоборот, продолжаться.

Можно выделить *основные типы задач*, которые сводятся к программированию циклических процессов:

1. **Вычисление суммы, произведения элементов конечного ряда**. При этом суть циклического процесса заключается в накоплении суммы S или произведения P по зависимости

$$S_n := S_{n-1} + U_n,$$

где S, стоящее справа от оператора присваивания – есть сумма элементов конечного ряда.

U – значение текущего элемента конкретчого ряда,

S, стоящее слева от оператора присваивания, – последующее (определяемое) значение суммы конечного ряда.

В случае вычисления произведения формула имеет вид:

$$P_n := P_{n-1} * U_n,$$

где P, стоящее справа от оператора присваивания, – есть произведение элементов конечного ряда,

U – значение текущего элемента конкретного ряда,

P, стоящее слева от оператора присваивания, – последующее (определяемое) значение.

2. **Определение суммы (произведения) элементов бесконечного ряда**. Условием окончания цикла является дополнительное логическое условие

$$|S_n - S_{n-1}| < \varepsilon,$$

где ε – заданная точность вычислений.

Сумма или произведение при этом вычисляются традиционно:

S:=S+U или P:= P*U.

3. **Табулирование функции** – получение таблицы значений функции на заданном интервале $x \in [x_{min}; x_{max}]$ при дискретном изменении x, равном dx.

Для реализации циклов в Паскале используются три оператора повтора:

`For` – для реализации циклов, число повторений которых известно

заранее (циклов со счетчиком);

Repeat и While – используются как в итерационных циклах, так и в циклах со счетчиком.

Оператор повтора For

Оператор повтора For состоит из заголовка и тела цикла и может быть представлен в двух форматах:

```
For <параметр цикла>:=N1 to N2 do <оператор>;
For <параметр цикла>:=N1 downto N2 do <оператор>;
```

Здесь For ... do – заголовок цикла, <оператор> – тело цикла.

Оператор может быть простым или составным. В последнем случае используются операторные скобки.

Параметр цикла, его начальное и конечное значения должны принадлежать к одному и тому же типу данных. При этом допустим любой скалярный тип, кроме вещественного.

N1 и N2 – соответственно начальное и конечное значения параметра цикла. Параметр цикла в процессе выполнения циклической части программы принимает следующие значения:

для первого формата: N1, N1+1, N1+2,... N2;

для второго формата: N1, N1-1, N1-2,...N2.

Отсюда следует, что в первом случае (при For ... to) значение N1 должно быть меньше или равно N2, во втором (при For ... downto) – значение N1 больше или равно N2, т.е. шаг изменения параметра цикла в операторе For в любом случае равен единице. Однако это не является большим недостатком, так как любой шаг можно задать при использовании операторов While и Repeat. Блок-схема оператора For представлена на рисунке 2.

Рисунок 2 – Блок-схема оператора For

Эта конструкция обеспечивает повторение действия А заданное число раз. При каждом повторении цикла величина К, называемая параметром цикла, увеличивается (или уменьшается) на 1. Выход из цикла происходит тогда, когда величина К превзойдет значение N, которое определяет число повторений цикла.

После нормального завершения оператора For значение параметра цикла равно конечному значению. Если оператор For не выполнялся, значение параметра цикла не определено. В теле оператора For могут содержаться другие операторы For. Это позволяет строить циклы, содержащие внутренние циклы. Такие внутренние циклы называются *вложенными*.

Пример

```
For i:=4 to 7 do
    Write(i:2);
```

Результат: 4 5 6 7.

```
For i:=7 downto 4 do
Write(I:2);
```

Результат: 7 6 5 4.

Оператор повтора Repeat

Часто при программировании циклов заранее невозможно определить число необходимых итераций или приближений. Поэтому при решении таких задач применяется не цикл со счетчиком (или с параметром), а цикл с постусловием, называемый иначе итеративным, или *итерационным*. В языке Паскаль для организации цикла с постусловием имеется специальный оператор `Repeat … Until`, проверяющий условие окончания цикла.

Формат

```
Repeat
    <оператор;>
    <оператор;>
    . . .
    <оператор >
Until <условие>;
```

Блок-схема оператора повтора `Repeat` представлена на рис. 3. Условие – это выражение булевского типа, при котором цикл заканчивается. Операторы, заключенные между словами `Repeat` и `Until`, являются телом цикла. Вначале выполняется тело цикла, затем проверяется условие выхода из цикла. Если результат выражения имеет значение `FALSE`, тело цикла выполняется еще раз, если результат имеет значение `TRUE` – происходит выход из цикла.

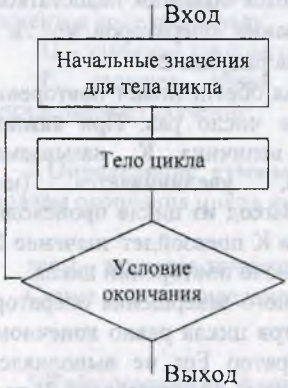


Рисунок 3 – Блок-схема оператора Repeat

Оператор `Repeat` имеет три характерные особенности:

- цикл выполняется по крайней мере хотя бы один раз;
- в теле цикла может находиться произвольное число операторов без операторных скобок `begin … end`;
- тело цикла выполняется, пока условие ложно.

По крайней мере, один из операторов, находящихся в теле цикла, должен влиять на значение условия, иначе цикл будет бесконечным.

Пример:

```
i:=4;
Repeat
Write(I:2);
i:=i+1
 Until I>7;
```

Результат: 5, 6, 7, 8

Таким образом, первой особенностью такого цикла является то, что заранее неизвестно число его повторений. Второй особенностью итерационного цикла является то, что тело цикла должно располагаться до проверки условия окончания (или повторения) цикла.

Оператор повтора While

Оператор While является самым мощным оператором повтора. Любой цикл, построенный с помощью For или Repeat, может быть построен и с помощью While.

Формат

```
While <выражение> Do <оператор>;
```

Этот оператор аналогичен оператору Repeat, но проверка условия выполнения тела цикла производится в самом начале оператора. Здесь выражение представляет собой условие, при котором цикл повторяется. Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат равен True, тело цикла выполняется и снова вычисляется выражение условия. Если результат равен False, то есть условие не выполняется, происходит выход из цикла и переход к первому после While оператору. Если перед первым выполнением цикла значение выражения было False, тело цикла вообще не выполняется и происходит переход к следующему оператору.

Блок-схема оператора While представлена на рис. 4.

Рисунок4 – Блок-схема оператора While

Эта структура обеспечивает многократное повторение некоторого действия A, называемого телом цикла до тех пор, пока выполняется условие P (условие повторения цикла). Если после очередного повторения действия A условие P не выполняется, происходит выход из цикла.

Цикл, который был описан для оператора REPEAT, в случае с WHILE будет иметь вид:

```
i:=4;
While i<=7 do
     Begin
        Write(i:2);
        i:=i+1
     End;
```

Как и в операторе Repeat, программист сам должен позаботиться об изменении переменных, определяющих условие выхода, иначе цикл получится бесконечным. Выйти из цикла можно с помощью оператора goto, минуя вычисление выражения условия.

Для начинающих программистов использование оператора While вызывает гораздо меньшее число ошибок, чем применение оператора Repeat, поэтому при прочих равных условиях лучше пользоваться оператором While.

Таким образом, можно выделить следующие отличия While от Repeat:

1. Условие ставится в начале цикла.
2. Это условие, при котором цикл повторяется.
3. Необходимость использования операторных скобок Begin... End, если в цикле выполняется более одного оператора.
4. Выход из цикла будет произведен тогда, когда условие, стоящее после While, не будет выполняться.

Рассмотренные операторы дают возможность описывать алгоритмы решения различных задач, которые представляют собой комбинацию линейных, разветвляющихся и циклических участков. Линейные участки, которые не содержат разветвлений и циклов, реализуются обычно с помощью оператора присваивания, операторов ввода-вывода.

Разветвления в программе реализуются с помощью оператора If, а циклические участки – с помощью одного из операторов цикла – For, While и Repeat. Использование оператора goto для этих целей нежелательно, так как это нарушает логику программы.

Лекция 5

ПОДПРОГРАММЫ ЯЗЫКА ПАСКАЛЬ

План лекции

- Понятие подпрограммы. Виды подпрограмм
- Подпрограммы-процедуры пользователя
- Параметры и переменные в подпрограммах
- Подпрограммы-функции пользователя
- Рекурсия
- Передача имен процедур и функций в качестве параметров

Алгоритм решения задачи проектируется путем декомпозиции всей задачи в отдельные подзадачи. Обычно подзадачи реализуются в виде подпрограмм.

Подпрограмма – это последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы. Каждая подпрограмма определяется уникальным именем. В языке ПАСКАЛЬ существуют два типа подпрограмм – процедуры и функции.

Процедура и функция – это именованная последовательность описаний и операторов. При использовании процедур или функций ПАСКАЛЬ – программа должна содержать текст процедуры или функции и обращение к процедуре или функции. Тексты процедур и функций помещаются в раздел описаний процедур и функций.

Понятие подпрограммы. Виды подпрограмм

В практике программирования часто встречаются ситуации, когда одну и ту же группу операторов, реализующих определенную цель, необходимо повторить без изменений в нескольких других местах программы. Чтобы избавить программиста от столь нерационального занятия, была предложена концепция подпрограмм, впервые описанная М.Уилксом в 1957 году. Она получила широкое распространение во всех языках программирования. В соответствии с этой концепцией программа представляется в виде одного глобального модуля – основной программы и вложенных в нее модулей – подпрограмм.

Подпрограммой называется именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения по имени любое количество раз из различных мест программы.

В языке Паскаль для организации подпрограмм используются подпрограммы-процедуры и подпрограммы-функции, которые, в зависимости от целей программирования, могут размещаться следующим образом:

- головная программа и подпрограммы расположены в одном программном модуле;
- подпрограммы расположены в отдельных файлах и включаются в основной файл посредством директивы компилятора;
- подпрограммы организуются как оверлейные структуры и поочередно загружаются в одно и то же место памяти;
- подпрограммы пишутся на машинном коде и затем включаются в программу;
- подпрограммы располагаются во внешней библиотеке и вызываются из основной программы.

Все процедуры и функции языка Паскаль разделяются на встроенные и описанные пользователем. *Встроенные* (стандартные) процедуры и функции являются частью языка и могут вызываться по имени без предварительного определения в разделе описаний. Различают 9 основных групп встроенных процедур и функций. Для обозначения типов данных аргументов используются сокращения: IBR – целочисленный (I), байтовый (B), вещественный тип (R); S – любой из скалярных типов, кроме вещественного.

1. Арифметические: `Abs(IBR)`, `Arctan(IBR)`, `Cos(IBR)` и т.д.
2. Скалярные: `Odd(I)`, `Pred(S)`, `Succ(S)`.
3. Преобразования типов: `Round(R)`, `Trunc(R)`. Результат имеет целочисленный тип.
4. Управление строками на экране: `ClrScr`, `DelLine`, `InsLine`.
5. Специальные функции: `Delay(T)`, `Exit`, `Halt`.
6. Функции обработки строк.
7. Функции и процедуры обработки файлов.
8. Функции управления памятью.
9. Графические процедуры.

Процедуры и функции пользователя организуются самим программистом в соответствии с синтаксисом языка и представляют собой локальный блок. Предварительное описание процедур и функций пользователя обязательно.

Подпрограммы-процедуры пользователя

Использование подпрограмм пользователя обеспечивает возможность организации работы нескольких программистов над одной программой, проведение отладки отдельных блоков и только потом всей программы в целом, значительно экономит память, т.к. многократно используемый участок

заносится в память только один раз, упрощает внесение изменений в программу.

Процедура пользователя – это именованная группа операторов, реализующая определенную часть общей задачи. Процедура вызывается по имени из любой части головной программы.

Описание процедуры включает заголовок и тело процедуры. Заголовок состоит из зарезервированного слова Procedure, идентификатора (имени) процедуры и необязательного списка формальных параметров, заключенного в круглые скобки. Для каждого параметра из списка указывается его тип.

Формат

```
Procedure <Имя процедуры>[(формальные параметры)]; {заголовок}
    <разделы описаний>
    Begin
      <раздел операторов>      {тело процедуры}
    End;
```

Пример заголовка

```
Procedure Sort(A:integer; B:real;var S:real);
Procedure Sum;
```

Имя процедуры – это идентификатор, записанный по правилам записи идентификаторов и уникальный в пределах программы. Тело процедуры – это локальный блок, по структуре аналогичный программе.

Для обращения к процедуре используется оператор вызова процедуры. Он состоит из идентификатора (имени) процедуры и списка фактических параметров, отделенных друг от друга запятыми и заключенных в круглые скобки.

Например, SORT(A1,B1,S1); SORT(14,25,S); SUM; .

Параметры и переменные в подпрограммах

Параметры, описанные в подпрограммах, могут иметь любой тип. Следует различать формальные и фактические параметры. Параметры, описанные в заголовке подпрограммы при ее определении, называются формальными. Например, `Procedure Sort(A:integer; B:real, var S:real);` – A, B и S – формальные параметры.

Параметры, которые указываются в заголовке подпрограммы при обращении к ней, называются фактическими.

Например, `SORT(A1,B1,S1);` – A1, B1 и S1 – фактические параметры – значения переменных.

`SORT(14,25);` – 14 и 25 – фактические параметры – непосредственные значения;

`SUM;` – фактические параметры не указаны, т.к. в вызываемой процедуре нет формальных параметров.

Когда параметры передаются по значениям, формальный параметр является переменной, локальной в блоке. Фактический параметр может быть любым выражением, переменной или идентификатором того же типа, что и соответствующий ему формальный параметр. Такие параметры называются параметрами-значениями. Их главная отличительная черта заключается в том, что изменение формального параметра не влечет за собой изменения фактического. Если же любое изменение формального параметра влечет за собой изменение фактического, то такие параметры называются параметрами-переменными. Для их описания служит зарезервированное слово Var.

Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке описания процедуры устанавливается взаимнооднозначное соответствие в результате их перебора слева направо. Количество и тип формальных параметров соответственно равны количеству и типу фактических параметров.

Если процедура возвращает в программу какие-то значения, соответствующие переменные должны быть описаны как параметры-переменные с использованием слова `Var`.

Например, `Procedure One(X:integer; var Y:real);`

параметр-значение параметр-переменная

Таким образом, формальные параметры делятся на два типа: *параметры-значения*, которые передаются в подпрограмму, и *параметры-переменные*, которые передаются из подпрограммы в вызывающую часть.

Например,
```
Procedure One(X:integer; var Y:real);
  Begin
    Y:=X*Sin(X);
  End;
```
Вызов: `One(2.5, Z);`

Если в программе используется несколько процедур и одна из них вызывает другую, то вызываемая процедура должна быть описана обязательно до вызывающей.

Различные особенности описания процедур и способы их использования рассмотрим на примере следующей задачи [3].

Пример 1. По задаваемым вещественным значениям x и y вычислить

`U = max(x+y, x*y), V = max(0.5, U).`

Алгоритм решения этой задачи сводится к поиску максимального значения их двух. Составим программу, реализующую решение без использования подпрограммы-процедуры:

```
Program Max_A;
  Var x,y,u,v:real;
```

```
    Begin
  Writeln('Введите x,y');
  Readln(x,y);
         If x+y>x*y Then U:=x+y Else U:=x*y;
         If 0.5>U Then V:=0.5 Else V:=U;
  Writeln('U=',U:3:1,'  V=',V:3:1);
End.
```

В программе фигурируют два условных оператора `If`, выполняющих одну и ту же частичную задачу: поиск максимального из двух значений. Поэтому алгоритм выполнения поиска целесообразно объявить процедурой.

Если программа имеет модульную структуру и состоит из ряда вложенных друг в друга блоков, то идентификаторы, описанные в определенных модулях, называются *локальными* и могут быть использованы только в той части программы, где они описаны. Идентификаторы, описанные в основной программе, являются *глобальными* и могут использоваться во всех вложенных блоках (процедурах и функциях). Все параметры, используемые в подпрограммах пользователя, подчиняются определенным правилам.

1. Каждый идентификатор должен быть описан перед тем, как он будет использован.

2. Областью действия идентификатора является блок, в котором он описан.

3. Идентификаторы в блоке должны быть уникальными.

4. Один и тот же идентификатор может быть по-разному определен в каждом отдельном блоке.

5. Если процедура или функция пользователя совпадает по имени со стандартной процедурой или функцией, то будет выполняться подпрограмма пользователя, а стандартная игнорироваться (`Sin`, `Cos`).

Процедура без параметров. Обратимся к условию примера 1 и, чтобы подчеркнуть сходство двух условных операторов, запишем программу иначе:

```
...
    a:=x+y; b:=x*y;
    If a>b Then S:=a Else S:=b;
    U:=S;
    a:=0.5; b:=U;
    If a>b Then S:=a Else S:=b;
    V:=S;
...
```

Таким образом, условные операторы `If a>b Then S:=a Else S:=b;` полностью совпадают друг с другом, то есть один и тот же условный оператор присутствует в программе дважды. Объявим его процедурой.

```
Procedure Max_1A;
  Begin
    If a>b Then S:=a Else S:=b;
  End;
```

Для вызова этой процедуры в нужном месте программы достаточно

записать оператор процедуры Max_1A .

Программа с этой процедурой выглядит следующим образом:

```
Program Max_B;
  Var x,y,u,v,a,b,s:real;
(*******************************)
  Procedure Max_1A;
   Begin
    If a>b Then S:=a Else S:= b;
   End;
(*******************************)
  Begin
   Writeln('Введите x,y');
   Readln(x,y);
   a:=x+y; b:=x*y;
   Max_1A; U:=S;
   a:=0.5; b:=u;
   Max_1A; V:=S;
   Writeln('U=',u:3:1,'  V=',v:3:1);
  End.
```

В этой программе переменные a, b, x, y, U, V, S описаны как глобальные и используются и в основной программе, и в подпрограмме-процедуре.

Процедура с параметрами. Предыдущий вариант не очень удобен, так как назначение процедуры зафиксировано слишком жестко. Следовательно, исходными данными для нее могут быть только значения переменных a и b. Поэтому при каждом обращении к процедуре этим переменным нужно присваивать те значения, из которых должно быть выбрано наибольшее. Этого можно избежать, оформив процедуру с параметром, и передавать значения a и b в качестве параметров-значений, а S – в качестве параметра-переменной.

```
Program Max_C;
  Var x,y,u,v:real;
(*************************************)
  Procedure Max_2A(a,b:real;var s:real);
   Begin
    If a>b Then S:=a Else S:= b;
   End;
(*************************************)
  Begin
   Writeln(' Введите x,y');
   Readln(x,y);
   Max_2A(x+y,x*y,U);
   Max_2A(0.5,U,V);
   Writeln('U=',u:3:1,'  V=',v:3:1);
  End.
```

Здесь переменные x, y, U, V описаны как глобальные и, являясь фактическими параметрами при обращении к процедуре Max_2A

используются только в основной программе. Параметры a, b, S – формальные и используются при описании процедуры, причем, параметры a и b – параметры-значения, S – параметр-переменная.

Подпрограммы-функции пользователя

Хотя набор встроенных функций Паскаля достаточно широк, он вряд ли может удовлетворить требованиям каждого программиста. Поэтому пользователю предоставляется возможность самому реализовывать нужные ему алгоритмы в виде функций и обращаться к ним из программ по мере необходимости. Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок состоит из ключевого слова Function, имени функции, необязательного списка формальных параметров, заключенного в круглые скобки с указанием типа каждого параметра и типа возвращаемого результата.

Формат

```
Function <имя>{(формальные параметры)}:тип; {заголовок}
    <разделы описаний>
    Begin
     <раздел операторов>;                    {тело функции}
End;
```

Например.
```
Function One(X:integer):real;
Function Sum:real;
```

В разделе операторов должен присутствовать, по крайней мере один оператор, присваивающий идентификатору функции значение. Если таких операторов несколько, то результатом работы функции будет значение последнего оператора присваивания. Обращение к функции производится по имени, после которого в круглых скобках следует необязательный список фактических параметров.

Имя функции – уникальный в пределах блока идентификатор. Раздел, относящийся к формальным параметрам для функции, аналогичен разделу описания формальных параметров для процедур. Возвращаемый результат может иметь любой скалярный тип.

Тело функции представляет собой локальный блок, по структуре аналогичный программе.

Особенности, присущие функциям:

1. Обязательно задается тип возвращаемого результата.
2. В результате работы функции получается только один результат.
3. В разделе операторов блока должен присутствовать хотя бы один оператор, влияющий на значение функции.
4. Обращение к функции должно происходить только из какого-либо выражения, *например*, `Y:=One(X);` а не `One(X)`.

Функция без параметров. Обратимся к условию рассмотренного ранее примера, в котором необходимо вычислить большее из двух значений U и V, где U = max(x+y), V = max(0.5, U).

Основываясь на вышеприведенных рассуждениях, составим программу с использованием подпрограммы-функции без параметров.

```
Program Max_D;
  Var x,y,u,v,a,b:real;
(*************************************)
  Function Max:real;
   Var S:real;
   Begin
    If a>b Then S:=a Else S:= b;
    Max:=S;
   End;
(*************************************)
  Begin
   Writeln('Введите x,y');
   Readln(x,y);
   a:=x+y; b:=x*y;
   U:=Max;
   a:=0.5; b:=u;
   V:=Max;
   Writeln('U=',u:3:1,'  V=',v:3:1);
  End.
```

В этой программе переменные a, b, x, y, U, V описаны как глобальные и используются и в основной программе, и в подпрограмме-функции Max, переменная S – локальная, так как используется только в подпрограмме-функции Max.

Функция с параметрами. Обратимся опять к условию предыдущего примера, в котором необходимо вычислить большее из двух значений U и V, где U = max(x+y), V = max(0.5, U) и составим программу с использованием подпрограммы-функции с параметрами.

```
Program Max_E;
     Var x,y,u,v:real;
(*****************************************)
 Function Max(a,b:real):real;
  Var S:real;
  Begin
   If a>b Then S:=a Else S:= b;
   Max:=S;
  End;
(*****************************************)
 Begin
 Writeln(' Введите x,y');
 Readln(x,y);
```

```
U:=Max(x+y,x*y);
V:=Max(0.5,U);
Writeln('U=',u:3:1,'  V=',v:3:1);
End.
```

Здесь x, y, U, V – глобальные и, являясь фактическими параметрами подпрограммы-функции Max, используются только в основной программе. Параметры a, b – формальные параметры, используемые при описании подпрограммы-функции Max, переменная S – локальная, так как используется только в подпрограмме-функции Max.

Рекурсивные подпрограммы

Во многих случаях оптимизация алгоритма решения задачи требует вызова для выполнения подпрограммы из раздела операторов той же самой подпрограммы, т.е. подпрограмма вызывает сама себя. Такой способ вызова называется *рекурсией*. Рекурсия полезна в тех случаях, когда основную задачу можно разделить на подзадачи, имеющие ту же структуру, что и первоначальная задача. Программы, реализующие рекурсию, называются *рекурсивными*.

Прямая рекурсия заключается в том, что подпрограмма в своем блоке использует саму себя.

Пример

```
Program A;
    Procedure B;
        Begin
        Writeln('Пример');
     B;
        End;
    Begin
      B;
    End.
```

Здесь программа A содержит процедуру B, которая будет вызывать сама себя и печатать слово «Пример» бесконечное число раз.

Классическим примером использования рекурсии является вычисление факториала целого положительного числа [5]. Проиллюстрируем на программе.

```
Program DemoRecurs;
 Var N:integer;
  Function Fact(A:integer):integer;
  Begin
   If A=0 Then Fact:=1 Else Fact:=A*Fact(A-1);
  End;
  Begin
```

```
While True do
 Begin
 Writeln('Введите число N');
 Readln(N);
  Case N of
   0   : Writeln('0! = 0');
  1..9: Writeln(N,'!=',Fact(N));
    10: Halt
  Else Writeln('Число должно быть в диапазоне 0..9');
 End;
End;
End.
```

Здесь рекурсия используется, чтобы решить подзадачи, затем сложить результаты их решений и получить таким образом решение задачи.

Часто встречается и другой вариант рекурсии, когда первая подпрограмма вызывает вторую, которая в момент вызова еще не определена. Такая ситуация называется косвенной рекурсией. Для реализации косвенной рекурсии используется так называемое предварительное описание процедур и функций.

Передача имен процедур и функций в качестве параметров

Во многих задачах, особенно в задачах вычислительной математики, необходимо передавать имена процедур и функций в качестве параметров. Для этого в TURBO PASCAL введен новый тип данных – процедурный или функциональный, в зависимости от того, что описывается.

Описание процедурных и функциональных типов производится в разделе описания типов:

```
type
      FuncType = Function(z: Real): Real;
      ProcType = Procedure (a,b: Real; var x,y: Real);
```

Функциональный и процедурный тип определяется как заголовок процедуры и функции со списком формальных параметров, но без имени. Можно определить функциональный или процедурный тип без параметров, например:

```
type
      Proc = Procedure;
```

После объявления процедурного или функционального типа его можно использовать для описания формальных параметров – имен процедур и функций.

Кроме того, необходимо написать те реальные процедуры или функции, имена которых будут передаваться как фактические параметры. Эти процедуры и функции должны компилироваться в режиме дальней адресации с ключом {$F+}.

Лекция 6

СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ. МАССИВЫ

План лекции

- Понятие массива. Способы описания массивов
- Действия над элементами массивов
- Примеры обработки массивов
- Сортировка массивов
  - обменные сортировки
  - сортировки выбором
  - сортировка «вставками»
  - сортировка «слияниями» (алгоритм фон Неймана)

Понятие массива. Способы описания массивов

До сих пор рассматривались лишь *простые* (скалярные), типы данных, где каждым значением любого из этих типов является отдельное данное, то есть тривиальная структура. Значения *структурированных* или *производных* типов в общем случае представляют собой нетривиальную структуру, то есть имеют более чем одну компоненту. При этом каждая компонента структуры может быть как отдельным данным, так и нетривиальной структурой, то есть значением любого из производных типов.

На языке Паскаль можно обрабатывать не только отдельные переменные, но и их совокупности. Одной из таких совокупностей и наиболее часто используемым производным типом является *регулярный* тип, а его значения обычно называют массивами. *Массив* – это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип.

Тип элементов массива называется *базовым*. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется. Доступ к каждому отдельному элементу осуществляется путем индексирования элементов массива. Индексы представляют собой выражения любого скалярного типа, кроме вещественного. Тип индекса определяет границы изменения значений индекса. Например, массив A состоит из элементов a_1, a_2, a_3, . . . a_n. Здесь a – *имя элемента* массива, n – его *индекс*, т.е. порядковый номер в массиве, по которому можно определить (найти) каждый элемент.

Для описания массива в языке Паскаль предназначено словосочетание ARRAY OF (массив из...). Описание массива возможно двумя способами.

Способ 1. Используя раздел описания типов

```
Type <имя типа> = array [тип индекса] of <тип компонент>;
     Var <идентификатор>: <имя типа>;
```

Например,
```
Type Mas = array[1..100] of real;
     Var  M1,M2:Mas;
```

Способ 2. Используя раздел описания переменных (Var) без представления типа в разделе описания типов данных.

```
Var
<идентификатор>: array [тип индекса] of <тип компонент>;
```

Например,
```
Var M1, M2: array[1..100] of real;
    V1: array[1..50] of integer;
```

Если в качестве базового типа взят другой массив, образуется структура, которую принято называть *многомерным* массивом.

Например,
```
Type Vector = array [1..4] of integer;
     Massiv = array [1..4] of Vector;
Var M1:Massiv;
```

Ту же структуру можно получить, используя другую форму записи:

```
Var M1: array[1..4,1..4] of integer;
```

Таким образом, многомерный массив задается при помощи индексированных выражений, следующих друг за другом. Если индексное выражение одно, то массив называется *одномерным*, если два – *двумерным* (матрица), если n – *n-мерным*. Одномерные массивы в Паскале обычно используются для представления векторов, двумерные – для представления матриц.

Например,
```
Var Vector = array [1..5] of real;
{одномерный массив-вектор из 5-ти элементов типа real}
    Matr = array [1..4,1..3] of integer;
{матрица из 4-х строк и 3-х столбцов с элементами типа integer}
```

При описании массивов для задания границ индексов можно использовать предварительно определенные константы. Например,

```
Const G1=4;G2=5;
Var   Matr = array [1..G1,1..G2] of integer;
```

Элементы массива располагаются в памяти последовательно, по строкам. Для массива Matr:

```
Matr[1,1]   Matr[1,2]   Matr[1,3]
Matr[2,1]   Matr[2,2]   Matr[2,3]
Matr[3,1]   Matr[3,2]   Matr[3,3]
Matr[4,1]   Matr[4,2]   Matr[4,3]
```

При обращении к элементам массива на Паскале индекс указывается в квадратных скобках: A[i], B[i,j].

При работе с массивами необходимо помнить следующее:

массив – это структура данных, представляющая собой совокупность элементов одного типа;

при объявлении массива указывается количество элементов массива;

доступ к элементу массива осуществляется путем указания индекса (номера) элемента массива.

Тип индекса элемента массива может быть только скалярным. Наиболее часто в качестве типа индекса используется ограниченный тип [1], причем в большинстве случаев это ограниченный целый тип.

Так, например, массив из 100 компонент вещественного типа может быть задан как `array [1..100] of real`.

Ограниченный целый тип `1..100` определяет количество компонент – `100` и их упорядоченность – от `1`-ой до `100`-ой.

В большинстве задач нумерация компонент начинается от 1 и ограничивается положительным целым числом. Но это необязательно. Если, например, представить в виде массива численность г. Москвы [1] в отдельные годы, то нумерацию удобно начинать с 1147 – года основания Москвы и до 2006 , т.е. такой массив можно задать как `array[1147..2006] of integer;`, а для Рима, основанного в 754 г. до н.э. – `array[-754..-1] of integer;`.

Здесь –1 –последний год до н.э.

Каждый из рассмотренных массивов может быть задан двояко:

1) используя описание типа:

```
Type  vector = array[1..100] of real;
      ch_M = array[1147..2006] of integer;
      ch_R = array[-754..-1] of integer;
Var   a, b: vector;
      c: ch_M; d: ch_R;
```

Обращение к элементам массивов `c` и `d`: `c[1147]`, `c[1999]` и т.п., то есть `c[i]`, `d[i]`;

2) непосредственно при описании переменной:

```
Var  a,b: array[1..100] of real;
     c: array[1147..2006] of integer;
     d: array[-754..-1] of integer;
```

Использование *ограниченного целого типа* для индексов дает достаточно широкие возможности для описания массивов.

Множество значений перечисляемого типа также образует *ограниченное перенумерованное множество*. Следовательно, перечисляемый тип также может быть использован в качестве типа индексов [1]. Значения перечисляемого типа упорядочены (порядок задается порядком перечисления имен, являющихся значениями этого типа), и число их конечно (определяется количеством имен в этом типе).

Например, необходимо проанализировать среднемесячную температуру воздуха за год [1]. Можно в качестве индекса использовать ограниченный целый тип 1..12, но можно для наглядности ввести перечисляемый тип, состоящий из имен месяцев года:

```
Type Month = (Jan, Feb, Mar, Apr, … , Dec);
```

Тогда переменные, являющиеся векторами среднемесячных температур, можно определить следующим образом: `Var t,r: array[Month] of real;`

Частичные компоненты массива, обозначающие температуру каждого месяца, запишутся следующим образом: `t[Yan]`, `t[Feb]`, `t[apr]` и т. п. Если ввести переменную `m` типа `Month (m: Month)`, то можно обращаться `t[m]`, где `m` меняется от `Jan` до `Dec`.

В качестве типа индексов могут использоваться также стандартные типы `Boolean` и `Char`. Например,

```
Type priznak = array[Boolean] of integer;
     kod_s = array[Char] of integer;
Var k: priznak; s: kod_s;
```

Частичные компоненты таких массивов выглядят следующим образом:

```
k[False], k[True];          s['d'], s['h'].
```

Для работы с массивом как с единым целым используется идентификатор массива без указания индекса в квадратных скобках. Если массивы описаны одинаково, то с ними можно проводить следующие операции [5]:

Проверка на равенство: `A=B`. Результатом является значение «истина», если каждый элемент массива A равен соответствующему элементу массива B.

Проверка на неравенство: `A<>B`. Результатом является значение «истина», если значение хотя бы одного элемента массива A не равно значению соответствующего элемента массива B.

Присваивание. `A:=B`. Все значения элементов массива B присваиваются элементам массива A. Значения элементов массива B остаются без изменения.

Действия над элементами массивов

После объявления массива каждый его элемент можно обработать, указав идентификатор массива и индекс элемента в квадратных скобках. Рассмотрим типичные ситуации, возникающие при работе с массивами. Для этого опишем четыре одномерных массива: два (`A` и `B`) из элементов целочисленного типа, два (`C` и `D`) из элементов вещественного типа и два двумерных массива (`V` и `W`), содержащих элементы вещественного типа.

```
Var A, B: array[1..5] of integer;
    C, D: array [1..10] of real;
    V, W: array [1..4,1..4] of real;
```

Инициализация массива

Инициализация – это присваивание каждому элементу массива какого-либо (или одного и того же) значения типа. Например, `A[1]:=0; A[2]:=0; ... A[5]:=0` или `V[1,1]:=0; V[1,2]:=0;` и т.д. Но при большом количестве элементов такой способ инициализации нерационален. Гораздо удобнее получить такой же результат, используя оператор FOR:

```
For I:=1 To 5 Do
    A[I]:= 0;
```

```
For I:=1 To 4 Do
For J:=1 To 4 Do
V[I,J]:=0;
```

Ввод элементов массива

Паскаль не имеет средств ввода-вывода элементов массива сразу, поэтому ввод и вывод значений производится поэлементно. Чаще всего они вводятся с помощью оператора `Read (Readln)` с использованием оператора цикла For:

```
For I:=1 To M Do Begin
Writeln('Введите',I,'
    элемент массива');
Read(A[I]);
End;
```

```
For I:=1 To M Do
 Begin
Writeln('Введите ',I,'строку');
For J:=1 To N Do
Read(V[I,J]);
End;
```

Вывод элементов массива

производится аналогично, но с использованием оператора `Write (Writeln)`.

```
Writeln('Вывод массива A');
For I:=1 To M Do
Write(A[I]:4:2,' ':2);
Writeln;
```

```
Writeln('Вывод массива V');
For I:=1 To M Do
Begin
 For J:=1 To N Do
   Write(V[I,J]:4:2,' ':2);
 Writeln; End;
```

Копирование элементов из одного массива в другой

Для выполнения этого действия необходимо, чтобы массивы, участвующие в этой операции, были описаны одинаково. Например, `B:=A;` - верно, так как B и A имеют одинаковый тип компонент; `A:=C;` – неверно, т.к. C имеет компоненты типа `real`, а A – типа `integer` и массивы A и C разной длины). Или, используя цикл:

```
For I:=1 To M Do Begin
  B[I]:=A[I];
  C[I]:=B[I] End;
```

```
For I:=1 To M Do
For J:=1 To N Do
W[I,J]:=V[I,J];
```

Поиск в массиве элементов, удовлетворяющих некоторым условиям

Пусть надо выяснить, сколько элементов в массивах C и W имеют нулевое значение.

```
K:=0;
```

```
K:=0;
For I:=1 To M Do
```

```
For I:=1 To M Do
If C[I]=0 Then K:=K+1;
```

```
For J:=1 To N Do
If W[I,J]=0 Then K:=K+1;
```

После выполнения цикла переменная K будет содержать количество элементов с нулевым значением.

Поиск минимального (максимального) элемента массива рассмотрим на примере поиска минимального элемента в одномерном массиве C и максимального – в двумерном массиве W.

```
Min:=C[1];
For I:=1 To M Do
If C[I]<Min
      Then Min:=C[I];
```

```
Max:=W[1,1];
For I:=1 To M Do
For J:=1 To N Do
If W[I,J]>Max Then Max:=W[I,J];
```

Вычисление суммы (произведения) элементов массива

Найти сумму (S) и произведение (P) элементов массивов C и W.

```
S:=0; P:=1;
For I:=1 To M Do
Begin
S:=S+C[I];
P:=P*C[I];
End;
```

```
S:=0; P:=1;
For I:=1 To M Do
For J:=1 To N Do
Begin
S:=S+W[I,J];
P:=P*W[I,J];
End;
```

Перестановка элементов массива

осуществляется с использованием вспомогательной переменной того же типа, что и базовый тип массивов.

Поменять местами первый и четвертый элемент массива A

```
Per:=A[1];
A[1]:=A[4];
A[4]:=Per;
```

Поменять местами первый, первый и четвертый, четвертый элемент массива W

```
Per:=W[1,1];
W[1,1]:=W[4,4];
W[4,4]:=Per;
```

Суммирование элементов матриц по строкам или столбцам

По строкам

```
For I:=1 To M Do
 Begin
  Str[I]:=0;
  For J:=1 To N Do
   tr[I]:=Str[I]+V[I,J];
 End;
```

По столбцам

```
For J:=1 To N Do
 Begin
  Stl[J]:=0;
  For I:=1 To M Do
   Stl[J]:=Stl[J]+V[I,J];
End;
```

Поиск минимального (максимального) элемента строки(столбца)

Результатом этого поиска является одномерный массив длиной M (для строк) или N (для столбцов).

```
For I:=1 To M Do
Begin
MIN[I]:=V[I,1];
For J:=2 To N Do
If V[I,J] < MIN[I]
Then MIN[I]:=V[I,J];
End;
```

```
For J:=1 To N Do
Begin
MIN[J]:=V[1,J];
For I:=2 To M Do
If V[I,J] < MIN[J]
Then MIN[J]:=V[I,J];
End;
```

Примеры обработки массивов данных

Рассмотрим основные действия над элементами массивов на примерах.

Пример 1. *Обработка одномерного массива.*

Вычислить % дней в месяце, когда температура опускалась ниже 0^0C.

Решение

Для упрощения вычислений введем следующие обозначения:

`all_d` – всего дней в месяце (примем значение `all_d =31`);

`cold_d` – количество холодных дней;

`t` – массив температур за месяц;

`pr` – процент дней с отрицательной температурой.

Текст программы `Cold` на языке Паскаль

```
Program Lk_6_1;
  {$APPTYPE CONSOLE}
uses
 SysUtils,
 Windows;

Type days = array[1..31] of real;
Var t: days;
    pr: real;
    cold_d,all_d:integer;
    i: integer;
Begin
  SetconsoleoutputCP(1251);
  Writeln('Введите количество дней в месяце');
  Readln(all_d);
  Writeln('Введите массив температур за месяц - t');
  For i:= 1 to all_d do
    Read(t[i]);
```

```
  cold_d:=0;
  For i := 1 to all_d do
    If t[i]<0 Then cold_d:=cold_d+1;
  pr:=cold_d/all_d*100;
  Writeln('Холодных дней',cold_d:2,' это ',pr:5:2,' %');
Sleep(10000)
End.
```

Пример 2. *Обработка двумерного массива.*
Найти след матрицы A(MxM).

```
Program Sled;
Var A:array[1..10,1..10] of  real;
    S1:Real;
     Begin
Writeln('Введите размер матрицы A -- M');
Readlm(M);
Writeln('Введите матрицу A');
For i:= 1 to M do
For j:= 1 to M do
  Readln(A[I,j]);
S1:=0;
For I:=1 To M Do
  S1:=S1+A[I,I];
Writeln('След матрицы A=',S1:0:2);
End.
```

Сортировка массивов

Упорядочивание элементов совокупности позволяет упростить дальнейший поиск или представить полученные результаты в более понятном виде. Рассмотрим основные определения.

Сортировкой называется процесс упорядочивания набора данных одного типа по возрастанию или убыванию значения какого-либо признака [1].

Массив, например, массив $A=\{a_1, a_2, \ldots, a_n\}$, является отсортированным (упорядоченным) по возрастанию, если для всех индексов i из интервала $\overline{i=1,n+1}$ выполняется условие $a_i \le a_{i+1}$.

Сортировка строк – это ранжирование данных строкового типа по алфавиту. Фактически символы, составляющие строку, упорядочиваются по их номеру в кодовой таблице ASCII. Наиболее часто сортировку строк выполняют при выводе различных списков (фамилий, названий).

Различают обменные сортировки, сортировки выбором, сортировку вставками, сортировку слиянием.

Предположим, что имеется исходный массив $A=\{a_1, a_2, \ldots, a_{10}\}$, который необходимо упорядочить по возрастанию.

Обменные сортировки

Суть обменных сортировок заключается в следующем. Сначала последовательно просматриваются элементы a_1, ..., a_n и находится наименьшее i, такое, что $a_i > a_{i+1}$. Затем элементы a_i и a_{i+1} меняются местами, и просмотр возобновляется с a_{i+1} элемента, и т.д. Тем самым, наибольшее число передвигается на последнее место. Следующие просмотры начинаются опять сначала, уменьшая на единицу количество просматриваемых элементов. Массив будет упорядочен после просмотра, в котором участвовали только первый и второй элементы. Процесс сортировки по убыванию будет аналогичен.

В пределах этого вида можно выделить такие методы сортировок, как метод «пузырька»; метод улучшенной «пузырьковой» сортировки; метод быстрой сортировки.

Для сортировки *методом «пузырька»* массив последовательно просматривается несколько раз. Каждый просмотр состоит из сравнения каждого элемента массива a_i с элементом a_{i+1} и обмена этих двух элементов, если они расположены не в нужном порядке.

***Пример 1*.** Пусть задан массив A, состоящий из элементов: A={25; 57; 48; 37; 12; 92; 86; 33}. Упорядочить его по возрастанию значений элементов.

Решение

На первом просмотре делаем следующие сравнения:

a_1 с a_2 (25 и 57) – нет обмена;

a_2 с a_3 (57 и 48) – обмен;

a_3 с a_4 (57 с 37) – обмен;

a_4 с a_5 (57 с 12) – обмен;

a_5 с a_6 (57 с 92) – нет обмена;

a_6 с a_7 (92 с 86) – обмен;

a_7 с a_8 (92 с 33) – обмен.

Таким образом, после первого просмотра элементы массива будут располагаться в таком порядке:

25 48 37 12 57 86 33 92.

После первого просмотра наибольший элемент 92 находится в нужной позиции. В общем случае элемент $a_{(n-i+1)}$ будет находиться в нужной позиции после i-той итерации. Этот метод называется *«методом пузырька»* потому, что каждое число, словно пузырек, медленно всплывает вверх в нужную позицию.

После второго просмотра получим последовательность:

25 37 12 48 57 33 86 92. Элемент 86 на нужном месте.

Поскольку каждая итерация помещает новый элемент в нужную позицию, то для сортировки некоторого массива, состоящего их n элементов, требуется не более n-1 итерации. Полный набор итераций для массива A выглядит следующим образом:

1-ая итерация 25 48 37 12 57 86 33 92;

```
2-ая итерация 25 37 12 48 57 33 86 92;
3-ая итерация 25 12 37 48 33 57 86 92;
4-ая итерация 12 25 37 33 48 57 86 92;
5-ая итерация 12 25 33 37 48 57 86 92;
6-ая итерация 12 25 33 37 48 57 86 92;
7-ая итерация 12 25 33 37 48 57 86 92.
```

Метод улучшенной «пузырьковой» сортировки основан на методе «пузырька» с той лишь разницей, что количество итераций меньше. Рассмотрим его на примере.

Пример 2. Упорядочим по возрастанию методом «пузырька» массив A, представленный в примере 13.1, а также проиллюстрируем метод улучшенной «пузырьковой» сортировки на примере упорядочивания данного массива по убыванию.

Решение

Создаем вспомогательные массивы B и C. В массиве B будут сохраняться результаты сортировки по убыванию, в массиве C – по возрастанию. Исходный массив A и отсортированные массивы B и C выведем на экран, используя подпрограмму-процедуру.

```
Program Sort1;
Uses CRT;
Type Mas=Array[1..50] Of Integer;
Var A,B,C:Mas;
    i,j,p,N:integer;
{Процедура вывода элементов массива}
Procedure Vyvod(M:Integer; Var V:Mas);
Begin
  For i:=1 to M do
    Write(V[i]:2,' ':2);
  Writeln
End;
Begin
Writeln('Введите количество элементов массива A');
Readln(N);
Writeln('Введите массив A');
 For i:=1 to N do
    Readln(A[i])
Writeln('Исходный массив');
Vyvod(N,A);
{Сортировка массива по возрастанию методом «пузырька»}
C:=A;
For i:=1 to N do
  For j:=1 to N-1 do
    If C[j]>C[j+1] Then
      Begin
```

```
            P:=C[j];
            C[j]:=C[j+1];
            C[j+1]:=P
          End;
    Writeln('Отсортированный по возрастанию массив A');
    Vyvod(N,C);
```
{Сортировка массива по убыванию методом улучшенной «пузырьковой» сортировки. При использовании метода улучшенной «пузырьковой» сортировки i изменяется от 1 до n-1, j от 1 до n-i}
```
    B:=A;
  For i:=1 to N-1 do
    For j:=1 to N-i do
      If B[j]<B[j+1] Then
        Begin
          P:=B[j];
          B[j]:=B[j+1];
          B[j+1]:=P
        End;
    Writeln('Отсортированный по убыванию массив A');
    Vyvod(N,B);
    Readln
    End.
```

Сеанс работы с программой:

Введите количество элементов массива A

8

Введите массив A

25 57 48 37 12 92 86 33

Результаты:

Исходный массив A

25.0 57.0 48.0 37.0 12.0 92.0 86.0 33.0

Отсортированный по возрастанию массив A

12.0 25.0 33.0 37.0 48.0 57.0 86.0 92.0

Отсортированный по убыванию массив A

92.0 86.0 57.0 48.0 37.0 33.0 25.0 12.0

Метод *быстрой сортировки* является еще одной реализацией обменных сортировок. Пусть имеется массив A, состоящий из N элементов. Для обозначения индексов элементов воспользуемся двумя переменными – i и j. Полагаем i=1, j=n. Если a_i меньше a_j, то j уменьшаем на единицу и снова сравниваем a_i и a_j. После первой перестановки i увеличивается на единицу, и выполняется еще одна перестановка. Затем j уменьшаем на единицу. Процесс сравнения продолжается до тех пор, пока исходный набор не будет упорядочен по возрастанию.

Рассмотрим этот метод на примере сортировки строк.

Пример 3. Имеется список бригады рабочих из 10 человек. Упорядочить его по возрастанию методом быстрой сортировки и вывести на экран.

Решение

Обозначим через переменную `w` строку длиной не более 20 символов (фамилия рабочего), `v` – массив из 10 элементов (количество рабочих), `p` – вспомогательная переменная строкового типа для перестановки элементов, `i` и `j` – индексы строк в массиве. Программа на языке Паскаль представлена ниже.

```
Program Sort_String;
Uses Crt;
Type W=String[20];
     V=Array[1..10] Of W;
Var Fio:V;
    I,J:Integer;
    P:W;
Begin
Clrscr;
Writeln('Ввод списка бригады');
For I:=1 To 10 Do
 Begin
   Write(i,' рабочий - ');
    Readln(FIO[I])
 End;
For I:=1 To 10 Do {Метод быстрой сортировки}
  For J:=10 Downto 1 Do
   If (FIO[I]>FIO[J]) And (I<=J) Then
     Begin
       P:=FIO[I];
       FIO[I]:=FIO[J];
       FIO[J]:=P;
     End;
Writeln('Отсортированный список бригады');
For I:=1 To 10 Do
  Writeln(Fio[I]);
Readln
End.
```

Сортировки выбором

Наиболее распространены следующие методы этого вида сортировки: с поиском минимального значения; с использованием бинарных деревьев; с использованием метода «турнира с выбыванием»; «пирамидальная» сортировка.

Рассмотрим подробнее первый метод этого вида сортировки – *с поиском минимального значения*. Суть его заключается в следующем. Сначала в массиве необходимо найти минимальный элемент, затем поменять его местом с первым,

затем в усеченном (исключая первый элемент) массиве опять найти минимальный элемент и поставить его на второе место и так далее.

Пример 4. Отсортировать по возрастанию массив элементов 1 3 0 9 2.

Решение

Процесс сортировки выбором по методу поиска минимального значения пройдет через следующие шаги:

0:	1	3	0	9	2	min=a[3]=0	переставляем a[1] < -- > a[3]
1:	0 |	3	1	9	2	min=a[3]=1	переставляем a[2] < -- > a[3]
2:	0	1 |	3	9	2	min=a[5]=2	переставляем a[3] < -- > a[5]
3:	0	1	2 |	9	3	min=a[5]=3	переставляем a[4] < -- > a[4]
4:	0	1	2	3	9	Готово	

Здесь знак | отделяет отсортированную часть массива от несортированной. Ниже приводится программа, реализующая данный метод.

```
Program Sort2;
 Uses Crt;
  Var A: Array[1..50] Of Real;
      I,J,N,K: Integer;
      Min,W: Real;
{Процедура вывода элементов массива}
Procedure Vyvod(M:Integer; Var V:Mas);
Begin
  For i:=1 to M do
    Write(V[i]:2,' ':2);
  Writeln
End;
   Begin
     Clrscr;
     Writeln('Введите количество элементов массива A');
     Readln(n);
     Writeln('Введите массив A');
     For i:=1 to n do
     Read(a[i]);
     Writeln;
     Writeln('Исходный массив A');
     Vyvod(N, A);
 {Сортировка массива выбором}
     For i:=1 to n do
      Begin
       min:=a[i]; k:=i;
       For j:=i+1 to n do
         If a[j]<=min then
         Begin min:=a[j]; k:=j end;
 {Перестановка i-го и min элементов}
         w:=a[i];
```

```
            a[i]:=a[k];
            a[k]:=w;
        End;
      Writeln('Отсортированный массив A');
      Vyvod(N, A);
      readkey
    End.
```

Сеанс работы с программой:

Введите количество элементов массива A

5

Введите массив A

1 3 0 9 2

Исходный массив A

1.0 3.0 0.0 9.0 2.0

Отсортированный массив A

0.0 1.0 2.0 3.0 9.0

Аналогичным данному методу и одним из самых простых методов сортировки является метод *прямого выбора*. Для его реализации выбирается максимальный элемент. Выбранный элемент меняется местами с последним элементом; процесс повторяется с оставшимися n-1, n-2 элементами и т.д., пока не останется один, самый малый по значению элемент.

Сортировка «вставками»

Наиболее распространены такие методы этого вида сортировки, как сортировка «простыми» вставками; сортировка «бинарными» вставками; сортировка Шелла; сортировка с вычислением адреса.

Суть сортировки «*простыми*» вставками заключается в последовательном переборе массива a_2, ..., a_n с тем, чтобы каждый новый элемент a_i вставлять на подходящее место в уже упорядоченной совокупности a_1, ..., a_{i-1}. Это место определяется последовательным сравнением a_i с упорядоченными элементами a_1, ..., a_{i-1}.

При сортировке «*бинарными*» вставками место, на которое надо вставить элемент a_i в уже упорядоченную совокупность a_1, ..., a_{i-1}, определяется алгоритмом деления пополам. Таким образом, получается алгоритм сортировки «*бинарными*» вставками, который понимается как «вставка делением пополам».

Пример 5. Вставить в упорядоченный по возрастанию массив A = {3; 4; 7; 9} новый элемент, равный 5, таким образом, чтобы сохранилась упорядоченность.

Решение

Алгоритм решения задачи следующий:

1. Найти в массиве элемент, больший вставляемого. Для этого следует просмотреть все элементы, начиная с первого. Это элемент a[3] = 7.

2. Увеличить длину массива на 1. Результат – массив A = {3 4 7 9 X}.

3. Все элементы, стоящие справа от найденного, включая его самого, то есть от 3-го, сдвинуть вправо. Результат – массив A = {3 4 7 7 9}.

4. На освободившуюся позицию, то есть на место элемента a[3], вставить новый элемент, то есть 5. В итоге получаем массив A = {3 4 5 7 9}.

При решении задач такого рода следует учитывать, что, если все элементы массива меньше вставляемого, новый элемент надо вставить в конец массива, и если все элементы массива больше вставляемого, то новый элемент надо вставить в начало массива.

Ниже приведена программа Sort3, реализующая данный алгоритм:

```
Program Sort3;
 Uses CRT;
  Var a: array[1..50] of real;
      i,j,n,k: integer;
      g:real;
{Процедура вывода элементов массива}
Procedure Vyvod(M:Integer; Var V:Mas);
  Begin
  For i:=1 to M do
    Write(V[i]:2,' ':2);
    Writeln
End;
    Begin
      Clrscr;
      Writeln('Введите количество элементов массива A');
      Readln(n);
      Writeln('Введите массив A');
      For i:=1 to n do
      Read(a[i]);
      Writeln;
      Writeln('Введите число, которое нужно вставить');
      Readln(g);
      Writeln('Исходный массив A');
      Vyvod(N, A);
      {1. Ищем элемент больше вставляемого}
      k:=1; {k - индекс сравниваемого элемента}
      While (k<=n) and (g>=a[k]) do {если k не превышает n и
вставляемый элемент меньше или равен a[k]}
          k:=k+1;{то переходим к следующему элементу}
      {2. Увеличиваем длину массива на 1}
      n:=n+1;
      {3.Сдвигаем элементы, начиная с k-го вправо}
      For i:=n downto k+1 do
        a[i]:=a[i-1];
      {4.В a[k] заносим g}
      a[k]:=g;
      Writeln('Новый массив A');
```

```
    Vyvod(N, A);
    readkey
  End.
```

Сеанс работы с программой:

Введите количество элементов массива A
4
Введите массив A
3 4 7 9
Введите число, которое нужно вставить
5
Исходный массив A
3.0 4.0 7.0 9.0
Новый массив A
3.0 4.0 5.0 7.0 9.0

Сортировка «слияниями» (Алгоритм фон Неймана)

Суть заключается в упорядочении массива $a_1, \ldots, a_n$ по неубыванию. Метод основан на многократных слияниях уже упорядоченных групп элементов массива. Сначала весь массив рассматривается как совокупность упорядоченных групп по одному элементу в каждом. Слиянием соседних групп получаются упорядоченные группы, каждая из которых содержит два элемента (кроме последней группы, которой не нашлось парной). Далее полученные группы укрупняются тем же способом и т.д. Здесь приходится использовать вспомогательный массив $b_1, \ldots, b_n$.

Лекция 7

СТРОКИ И ЗАПИСИ

План лекции

- Строки
  - Строковые выражения
  - Процедуры обработки строк
  - Функции обработки строк
  - Пример использования средств обработки строк
- Записи
  - Описание типа
  - Доступ к полям записи

Строки

Кроме массивов, к структурированным типам данных относятся *строки* и *записи*.

Строка – это последовательность символов. При использовании в выражениях строка заключается в апострофы. Количество символов в строке (длина строки) может изменяться от 0 до 255. Если это значение не указывается, то по умолчанию длина строки равна 255 байт. Для определения данных строкового типа используется идентификатор `string`, за которым следует заключенное в квадратные скобки значение максимально допустимой длины строки данного типа.

Переменную строкового типа можно определить через описание типа в разделе описания типов или непосредственно в разделе описания переменных.

Формат

1. Через раздел описания типов

```
Type <имя типа>=string[максимальная длина строки];
Var <идентификатор, …>:<имя типа>;
```

2. Через раздел описания переменных

```
Var <идентификатор, …>:string[максимальная длина строки];
```

Например,

```
Type FIO =string[35];
Var F1,F2,F3:FIO;
```

или `Var F1,F2,F3:string[35];`

или `Var F1,F2,F3:string;`-по умолчанию длина строки 255 символов.

`F1:='Чехов Антон Павлович';` – переменной F1 присвоено значение строкового типа `FIO`.

В Паскале переменные типа `string[N]` занимают `N+1` байт памяти. Так в приведенном примере строка типа `FIO` будет занимать `36 байт`.

Если строковые данные используются в программе в качестве констант, они должны быть описаны по правилам описания констант:

`Const Adres='ул. Короленко, 5';` – строковая константа,

Недопустимо применение строковых переменных в качестве селектора в операторе CASE.

Строковые выражения

Выражения, в которых операндами служат строковые данные, называются *строковыми выражениями*. Они могут состоять из строковых констант, переменных, указателей функций и знаков операций.

Над строковыми данными допустимы следующие операции:

- *сцепления* («+») – для сцепления нескольких строк в одну результирующую строку;
- *отношения* – проводит сравнение двух строковых операндов. Допустимы такие операции отношения, как равенство («=»), неравенство («<>»), более чем («>»), менее чем («<»), больше либо равно («>=»), меньше либо равно («<=»).

Вначале выполняются операции сцепления, если они присутствуют, потом реализуются операции отношения. Сравнение производится слева направо до первого несовпадающего символа и та строка считается большей, в которой первый несовпадающий символ имеет больший номер в стандартной таблице обмена информацией. Результат выполнения операций отношения над строковыми операндами всегда имеет булевский тип и принимает значение True, если выражение истинно, и False, если выражение ложно.

Например,

условие `'OLD'<'Old'` будет иметь значение True, так как символ 'L' имеет меньший номер, чем символ 'l'.

условие `'student'>'STUDENT'` также будет истинным, поскольку символ 's' имеет больший номер в таблице ASCII, чем символ 'S'.

Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная;

Строки считаются равными, если они полностью совпадают по длине и содержат одни и те же символы.

Для присваивания строковой переменной результата строкового выражения используется оператор присваивания (:=). Например,

```
Str1:='Студенты группы ИС7';
Str2:=Str1+' ФПК';
```

Если значение переменной после выполнения оператора присваивания превышает по длине максимально допустимую при описании величину длины строки, все лишние символы отбрасываются.

Пример

```
Program Stroki;
Uses Crt;
Var St1,St2,St3: String[25];
          St4: String[40];
Begin
Clrscr;
Writeln('Введите St1');
Readln(St1);
Writeln('Введите St2');
Readln(St2);
Writeln('Введите St3');
Readln(St3);
St4:=St1+St2+St3;
Writeln(St4);
Readln
End.
```

Сеанс работы с программой:

Введите St1
Программное_
Введите St2
обеспечение_
Введите St3
информационных систем

Результат:
Программное обеспечение информационных с

Результатом будут являться строка St4, содержащая текст «Программное обеспечение информационных с», так как максимально допустимая длина этой строки равна 40 символов. Для исправления этой ошибки следует увеличить длину строки, предположим, до 50 символов.

К отдельным символам строки можно обратиться по номеру (индексу) данного символа в строке. Индекс определяется выражением целочисленного типа, которое записывается в квадратных скобках сразу за идентификатором строковой переменной или константы. Например, `St1[3]` обеспечит доступ к третьему символу строки `St1`, описанной выше. Этим символом будет буква «о».

Процедуры обработки строк

`Delete(St,Poz,N)` – удаление N символов строки `St`, начиная с позиции `Poz`. Если значение `Poz` > 255, возникает ошибка.

Значение St	Выражение	Результат
'абвгде'	Delete(St,4,2)	'абве'
'река Волга'	Delete(St,1,5)	'Волга'
'NB HP6820s'	Delete(St,6,5)	'NB HP'

`Insert(St1,St2,Poz)` – вставка строки `St1` в строку `St2`, начиная с позиции `Poz`.

Пример:

```
Var SI,S2,S3:string[10];
SI :='HP'; S2 := 'NB 6820'
S3:=Insert(S1,S2,4);
```

В результате выполнения последнего выражения значение строки `S3` станет равным 'NB HP6820'.

`Str(IBR, St)` – преобразование числового значения величины `IBR` и помещение результата в строку `St`. После результата можно задавать формат.

Значение IBR	Выражение	Результат
1500	Str(IBR:6, St)	' 1500'
4.8E+03	Str(IBR:10,St)	' 4800'
-78854	Str(-IBR:3,St)	'-76854'

`Val(St,IBR,Cod)` – преобразует значение `St` в величину целочисленного или вещественного типа и помещает результат в `IBR`. Значение `IBR` не должно содержать незначащих пробелов. `Cod` – целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, значение `Cod=0`. Если ошибка обнаружена, `Cod` будет содержать номер позиции первого ошибочного символа, а значение IBR не определено.

Значение St	Выражение	Результат
'1450'	Val(St,IBR,Cod)	1450 Cod=0
'14.2E+02'	Val(St,IBR,Cod)	1420 Cod=0
'14.2A+02 '	Val(St,IBR,Cod)	? Cod=5

Функции обработки строк

Length(St) – вычисляет длину в символах строки St. Результат имеет целочисленный тип.

Значение St	Выражение	Результат
'123456789'	Length(St)	9
'System 370'	Length(St)	10

Copy(St,Poz,N) – выделяет из St подстроку длиной N символов, начиная с позиции Poz. Если Poz > Length(St), то результатом будет пробел; если Poz > 255, возникнет ошибка при выполнении. Функция Length описана выше. Poz, N – целочисленные выражения.

Значение St	Выражение	Результат
'ABCDEFG'	Copy(St,2,3)	'BCD'
'ABCDEFG'	Copy(St,4,10)	'DEFG'

Concat(St1,St2,... ,StN) – выполняет сцепление строк St1, St2... ,StN в том порядке, в каком они указаны в списке параметров. Сумма символов всех сцепленных строк не должна превышать 255.

Выражение	Результат
Concat('AA', 'XX', 'Y')	'AAXXY'
Concat('Индекс ','603000')	'Индекс 603000'

Pos(St1,St2) – обнаруживает первое появление в строке St2 подстроки St1. Результат имеет целочисленный тип и равен номеру той позиции, где находится первый символ подстроки St1. Если в St2 подстроки St1 не найдено, результат равен 0.

Значение St1	Выражение	Результат
'abcdef'	Pos('de',St1)	4
'abcdef'	Pos('r',St1)	0

UpCase(Ch) – преобразует строчную букву в прописную. Параметр и результат имеют литерный тип. Обрабатывает буквы только латинского алфавита.

Значение	Выражение	Результат
'a'	UpCase(Ch)	'A'
'v'	UpCase(Ch)	'V'

Для примера использования средств обработки строк приведем текст часто применяемой при форматном выводе строк функции, которая удаляет в строке все пробелы слева, если они имеются.

Пример 1

```
Program DemoLeftDelete;
  Var Str : string[20];
  Function DelLeft (Stroka : Str):Str;
   Var Dlina : byte;
    begin
     Dlina := Ord(Stroka[0]);
     while ((Dlina > 0) and (PrStr[1] = ' ')) do
     Delete( Stroka, 1,1);
     DelLeft : = Stroka
    end;
Begin
writeln(DelLeft('  Строка выводится без пробелов слева));
End.
```

Пример 2

Найти, сколько раз в списке группы встречаются фамилии *с окончанием «ова»*.

Например, Иван*ова*, Петр*ова* и т.д. (но не *Ов*алов, Заг*ов*алов).

```
Program Name;
Uses SysUtils,Windows;
Var Fam:Array[1..30] Of String[25];
    i,n,k,j,dl:integer;
Begin
SetconsoleoutputCP(1251);
Writeln('Введите количество студентов в группе');
Readln(n);
For i:=1 to n do
   Begin
     Write('Введите фамилию студента - ');
     Readln(Fam[i])
   End;
k:=0;
```

```
For i:=1 to n do{цикл для организации последовательного
                 перебора фамилий студентов}
 Begin
   dl:=length(Fam[i]);{Вычисление количества символов фамилии
                       студента}
     If Copy(Fam[i],dl-2,3)='ова' Then k:=k+1
{If (Fam[i,dl]='a') And (Fam[i,dl-1]='в') And (Fam[i,dl-
2]='o')}
   End;
Writeln('Количество студентов, чья фамилия заканчивается на
-ова ',k:2);
Writeln('Список группы');
For i:=1 to n do
Writeln(i,' ':3,Fam[i]);
Readln
End.
```

Результат выполнения программы:

```
Количество студентов, чья фамилия заканчивается на -ова, - 4
    Список группы
    1   Иванов
    2   Иванова
    3   Петров
    4   Петрова
    5   Заговалов
    6   Повалий
    7   Овалов
    8   Овалова
    9   Мухова
    10  Ковалева
```

Записи

Описание типа

Ранее рассматривались структурированные типы, состоящие из компонентов только одного типа. Однако часто требуется группировать данные разного типа, логически относящиеся к одному объекту. Например, целесообразно объединить данные о владельце автомобиля (номер, марка машины, имя, фамилия владельца, адрес) в одной структуре. Для этого в языке ПАСКАЛЬ предназначен комбинированный тип, его также называют записью. Записи могут использоваться в любых случаях, где желательно группировать логически связанные элементы данных в один блок.

Запись – это структурированный тип данных, состоящий из фиксированного числа компонентов разного типа. Определение типа записи

начинается идентификатором record и заканчивается зарезервированным словом end. Между ними заключен список компонентов, называемых *полями*, с указанием *идентификаторов полей* и *типа каждого поля*.

Формат

```
Type
<имя типа> = record
<идентификатор поля>:<тип компонента>;
… … … …
<идентификатор поля>:<тип компонента> end;
Var
<идентификатор,. ..> :   <имя типа>;
```

Пример

```
Type Mash = record
       Nomer : integer;          {номер}
       Marka : string[15];       {марка автомобиля}
       FIO   : string[40];       {фамилия, инициалы владельца}
       Adres : string[60];       {адрес владельца}
     end;
Var M,V : Mash;
```

В данном примере запись Mash содержит четыре компонента: номер, название марки машины, фамилию владельца и его адрес. Доступ к полям записи осуществляется через переменную типа *запись*. В нашем случае это переменные M, V типа Mash.

Идентификатор поля должен быть *уникален* только в пределах записи, однако во избежание ошибок лучше делать его уникальным в пределах всей программы.

Объем памяти, необходимый для записи, складывается из длин полей.

Доступ к полям записи

Значения полей записи могут быть использованы в выражениях. Имена отдельных полей не применяются по аналогии с идентификаторами переменных, поскольку может быть несколько записей одинакового типа. *Обращение к значению поля* осуществляется с помощью *идентификатора переменной и идентификатора поля, разделенных точкой*. Такая комбинация называется составным именем. Например, чтобы получить доступ к полям записи Mash, надо записать: M.Nower, M.Marka, M.FIO, M.Adres.

Составное имя можно использовать везде, где допустимо применение типа поля. Для присваивания полям значений используется оператор присваивания.

Пример.

```
M.Nomer:= 1678;
M.Marka:= 'ГАЗ-24';
M.FIO:= 'Орлов А.Г.';
M.Adres:= 'ул.Пушкина 12-31';
```

Составные имена можно использовать, в частности, в операторах ввода-вывода:

```
read(M.Nomr, M.Marka,M.FIO,M.Adres);
write(M.Nomr:4, M.Marka:7,M.FIO:12,M.Adres:25);
```

Допускается применение оператора присваивания и к записям в целом, если они имеют один и тот же тип. Например, `V:=M;` . После выполнения этого оператора значения полей записи `V` станут равны значениям соответствующих полей записи `M`.

В ряде задач удобно пользоваться массивами из записей. Их можно описать, например, следующим образом:

Пример. Для записи типа `Mash` (см. выше):

```
Type Mash = record
       Nomer : integer;          {номер}
       Marka : string[15];       {марка автомобиля}
       FIO   : string[40];       {фамилия, инициалы
владельца}
       Adres : string[60];       {адрес владельца}
     end;
Var List:array[1..50] of Mash;
```

Обращение к полю каждой записи типа `Mash` выполняется следующим образом: `List[i].Nomer`,

где `i` может принимать значения от 1 до 50.

*Пример. С*писок сотрудников:

```
Type Person = record
       FIO:string[20];
       Age:0 . .90;
       Prof : string [20] end;
Var Spisok:array[1..50] of Person;

  Spisok [i]. FIO
  Spisok [i]. Age
```

Очевидно, что *обращение* к полям записи имеет несколько громоздкий вид, что особенно неудобно при использовании мнемонических идентификато-

ров длиной более 5 символов. Для решения этой проблемы в языке ПАСКАЛЬ предназначен оператор with, который имеет следующий формат:

```
with <переменная типа запись> do <оператор>;
```

Один раз указав переменную типа запись в операторе with, можно работать с именами полей как с обычными переменными, т. е. без указания перед идентификатором поля имени переменной, определяющей запись.

Пример.

Присвоить значения с помощью оператора with.

а) полям записи M типа Mash

```
with M do begin
  Nomer := 5543;
  Marks := 'ГАЗ-2Г';
  FIO   := 'Патин U. И';
  Adres := 'ул. Новая, 5' end;
```

в) полям массива записей List типа Mash:

```
  for i:=1 to n do
with list[i] do begin
  Nomer := 5543;
  Marks := 'ГАЗ-2Г';
  FIO   := 'Патин U. И';
  Adres := 'ул. Новая, 5' end;
```

Здесь n – число автомобилистов в списке.

ПАСКАЛЬ допускает вложение записей друг в друга (т.е. поле в записи может быть в свою очередь тоже записью), соответственно оператор with тоже может быть вложенным:

```
with RV1 do with RV2 do
with RVn do что эквивалентно записи
with   W1.RV2,... ,RVn do ...
```

Лекция 8

ПРИНЦИПЫ РАЗРАБОТКИ БИБЛИОТЕК ПОДПРОГРАММ ПОЛЬЗОВАТЕЛЯ В ЯЗЫКЕ ПАСКАЛЬ

План лекции

- Внешние подпрограммы
- Модули

Стандартный язык Pascal не располагает средствами разработки и поддержки библиотек программиста (в отличие, скажем, от языка Fortran и других языков программирования высокого уровня), которые компилируются отдельно и в дальнейшем могут быть использованы как самим разработчиком, так и другими. Если программист имеет достаточно большие наработки, и те или иные подпрограммы могут быть использованы при написании новых приложений, то приходится эти подпрограммы целиком включать в новый текст.

В языке Pascal это ограничение преодолевается за счет, во-первых, введения внешних процедур, во-вторых, разработки и использования модулей.

Внешние подпрограммы

Такой механизм предусматривает, что исходный текст каждой процедуры или функции хранится в отдельном файле и при необходимости с помощью специальной директивы компилятора включается в текст создаваемой программы.

Покажем это на примере задач целочисленной арифметики, где аргументы, результаты и промежуточные величины являются целыми (`Integer, Word, LongInt` и т.д.). Вот несколько таких задач.

1. Дано натуральное число n. Найти сумму первой и последней цифры этого числа.

2. Дано натуральное число n. Переставить местами первую и последнюю цифры этого числа.

3. Дано натуральное число n. Дописать к нему цифру k в конец и в начало (если это возможно, т.е. результат не выйдет за диапазон допустимых значений), или сообщить о невозможности выполнения операции.

4. Найти наибольшую цифру в записи данного натурального числа.

5. Дано натуральное число n. Переставить его цифры так, чтобы образовалось максимальное число, записанное теми же цифрами.

При решении каждой из этих задач может быть использована функция, возвращающая количество цифр в записи натурального числа и функция возведения натурального числа в натуральную степень.

Натуральные числа (естественные числа) **N = {1, 2, 3, . . .}**— числа, возникающие естественным образом при счёте (как в смысле перечисления, так и в смысле исчисления). Существуют два подхода к определению натуральных чисел — числа, используемые при:

- **перечислении (нумеровании) предметов** (*первый, второй, третий…*) — подход, общепринятый в большинстве стран мира (в том числе и в России);
- **обозначении количества предметов** (*нет предметов, один предмет, два предмета…*). Отрицательные и нецелые числа натуральными числами не являются.

В русской литературе обычно ноль исключён из числа натуральных чисел $0 \notin \mathbb{N}$

Целые числа Z = {-2, -1, 0, 1, 2, 3, . . .}определяется как замыкание множества натуральных чисел $\mathbb{N}$относительно арифметических операций сложения (+) и вычитания (-). Таким образом, сумма, разность и произведение двух целых чисел есть снова целые числа. Оно состоит из натуральных чисел (1, 2, 3), чисел вида **-n** ($n \in \mathbb{N}$) и числа нуль.

Возможный вариант функции, возвращающей количество цифр в записи натурального числа:

```
Function Digits(N : LongInt) : Byte;
Var Kol : Byte;
Begin
    Kol := 0;
    While N 0 Do Begin Kol := Kol + 1; N := N Div 10 End;
    Digits := Kol
End;
```

Этот текст необходимо сохранить в файле с расширением .inc (это расширение внешних подпрограмм в Pascal), например, digits.inc.

- Возможный вариант функции возведения натурального числа A в натуральную степень N {файл power.inc}.

```
Function Power(A,N : LongInt):LongInt;
Var I, St : LongInt;
Begin
    St := 1;
    For I := 1 To N Do St := St * A;
    Power := St
End;
```

- Попробуем использовать эти функции при решении задачи номер один.

```
Program Example1;
Var N, S : LongInt;
{$I digits.inc}
```

{подключаем внешнюю функцию digits.inc, возвращающую количество цифр в записи числа}

```
{$I power.inc} {внешняя функция, выполняющая возведение числа A в степень N}
Begin
    Write('Введите натуральное число: ');
    ReadLn(N);
```

{для определения последней цифры числа N берем остаток от деления этого числа на 10, а для определения первой делим N на 10 в степени на единицу меньшую, чем количество цифр в записи числа (нумерация разрядов начинается с 0)}

```
    S := N Mod 10 + N Div Power(10, Digits(N) - 1);
    WriteLn('Искомая сумма: ', S)
End.
```

Внешние процедуры создаются и внедряются в использующие их программы аналогично функциям.

Модули

Модуль — это набор ресурсов (функций, процедур, констант, переменных, типов и т.д.), разрабатываемых и хранимых независимо от использующих их программ. В отличие от внешних подпрограмм модуль может содержать достаточно большой набор процедур и функций, а также других ресурсов для разработки программ. Обычно каждый модуль содержит логически связанные между собой программные ресурсы.

В основе идеи модульности лежат принципы структурного программирования. В любом языке программирования существуют стандартные модули, которые обычно описываются в литературе по данному языку.

Модуль (UNIT) в PASCAL – это особым образом оформленная библиотека подпрограмм. Модуль, в отличие от программы, не может быть запущен на выполнение самостоятельно, он может только участвовать в построении программ и других модулей. Модули позволяют создавать личные библиотеки процедур и функций и строить программы практически любого размера. Модуль в PASCAL представляет собой отдельно хранимую и независимо компилируемую программную единицу.

Все программные элементы модуля можно разбить на две части:

- программные элементы, предназначенные для использования другими программами или модулями, такие элементы называют видимыми вне модуля;
- программные элементы, необходимые только для работы самого модуля, их называют невидимыми или скрытыми.

В соответствии с этим модуль, кроме заголовка, содержит две основные части, называемые интерфейсом и реализацией.

В общем случае модуль имеет следующую структуру:

```
Unit <имя модуля>
interface   <интерфейсная часть>
uses <список используемых модулей>;        {открытые объявления }
Implementation   <исполняемая часть>
 uses <список используемых модулей>;       {собственные объявления}
                                           {процедуры и функции}
 Initialization   <инициализирующая часть>
 Finalization   <завершающая часть>
End.
```

Зарезервированное слово `Unit` – определяет заголовок модуля; <имя> – имя модуля, которое (для удобства дальнейших действий) должно совпадать с именем файла, содержащего данный модуль и составляться в соответствии с правилами формирования идентификатора.

Зарезервированное слово `Interface` – определяет интерфейсную часть модуля. В разделе `Interface` объявляются все ресурсы, которые будут в дальнейшем доступны программисту при подключении модуля. В этом разделе могут указываться другие модули (их список следует за ключевым словом `uses`). При этом все объекты, объявленные в интерфейсах этих модулей, могут быть использованы в любом объявлении в интерфейсе данного модуля. Использование в модулях процедур и функций имеет свои особенности. Для подпрограмм здесь указывается лишь полный заголовок (без служебного слова `forward`). Полный текст процедуры или функции помещают в часть реализации, причем заголовок может не содержать список формальных параметров.

Зарезервированное слово `Implementation` – начинает исполняемую часть, которая заканчивается зарезервированным словом `End`. В разделе `Implementation` реализуются все подпрограммы, которые были ранее объявлены. Кроме того, здесь могут содержаться свои константы, переменные, типы, подпрограммы и т.д., которые носят вспомогательный характер и используются для написания основных подпрограмм. В отличие от ресурсов, объявленных в разделе `Interface`, все, что дополнительно объявляется в `Implementation`, уже не будет доступно при подключении модуля. При написании основных подпрограмм достаточно указать их имя (т.е. не нужно полностью переписывать весь заголовок), а затем записать тело подпрограммы. Если между словами `Implementation` и `End` появляется ключевое слово begin, то получившийся составной оператор begin …`End` становится разделом инициализации модуля.

Раздел инициализации – `Initialization`, который часто отсутствует, содержит операторы, которые должны быть выполнены сразу же после запуска программы, использующей модуль.

Зарезервированное слово `Finalization` начинает завершающую часть модуля.

В частном случае модуль может не содержать части реализации и части инициализации, тогда структура модуля будет такой:

```
unit <имя модуля>;        {заголовок модуля}
interface
{ описание видимых программных элементов модуля }
implementation
end.
```

Исходный текст модуля должен быть откомпилирован с помощью директивы Make подменю Compile и записан на диск. Результатом компиляции модуля является файл с расширением .TPU (Turbo Pascal Unit). Основное имя модуля берется из заголовка модуля. Для подключения модуля к программе необходимо указать его имя в разделе описания модулей, например:

```
uses  CRT, Graph, Modul;
```

Если в программе, использующей модуль, имеются идентификаторы, совпадающие с точностью до символа с идентификаторами модуля, то они «перекрывают» соответствующие ресурсы модуля и обращение будет происходить к переменной, описанной в программе. Тем не менее, даже в такой ситуации для доступа к этим ресурсам модуля необходимо применить составное имя, состоящее из имени модуля и имени переменной, разделенных точкой.

Например, пусть имеется модуль, в котором описана переменная K:

```
unit M;
 interface
  var K: Integer;
  implementation
.................
 end.
```

Пусть программа, использующая этот модуль, также содержит переменную K:

```
Program P;
 uses M;
 var K: Char;
 begin
.............
 end.
```

Для того, чтобы в программе P иметь доступ к переменной K из модуля M, необходимо задать составное имя `M.K`. Использование составных имен

применяется не только к именам переменных, а ко всем именам, описанным в интерфейсной части модуля. Если в модуле имеется раздел инициализации, то операторы из этого раздела будут выполнены перед началом выполнения программы, в которой используется этот модуль.

Пример

Рассмотрим пример использования модулей [12]. Разработаем модуль с именем Ar и поместим в него функцию, предназначенную для вычисления tg(x), $tg(x)$ функцию для вычисления $\log_a b \log_b(a)$ и процедуру для вычисления суммы n членов арифметической прогрессии по известной формуле $S_n = n*(a_0+a_n)/2$. $S_n = n \cdot (a_0 + a_n)/2$

Для создания модуля следует выбрать команду File ⇒ New ⇒ Unit,

ввести текст модуля и сохранить его в файле Ar.Pas (имя файла должно совпадать с именем модуля).

Текст модуля Ar будет иметь следующий вид:

```
unit Ar;    //Объявление модуля
interface    //Раздел интерфейса
Function Tg(x: Real):Real;    //Объявление функции
```
вычисления tg(x)
```
Function Log(a,b:Real):Real; //Объявление функции
```
вычисления $\log_a(x)$
```
Procedure Sumar(a0,an:Real;n:Integer; Var sn:Real); //
```
Объявление процедуры
```
implementation
Function Tg(x: Real): Real;
begin
Tg:=Sin(x)/Cos(x)
end;
Function Log(a,b:Real): Real;
begin
Log:=Ln(a)/Ln(b);
end;
Procedure Sumar(a0,an: Real; n:Integer; Var sn:Real);
  begin
  sn:=n*(a0+an)/2
  end;
end.
```

Для того, чтобы этот модуль можно было использовать, его имя Ar должно быть указано в разделе Uses программы или другого модуля. Например:

```
Program Test;
{$APPTYPE CONSOLE}
Uses SysUtils, Windows, Ar;
Var x,a,b,a0,an,s:Real;
                        n:Integer;
Begin
Setconsoleoutputcp(1251);
Writeln('Введите значение x');
Readln(x);
Writeln('Значение Tg(x) = ', Tg(x):0:2);
Writeln('Введите значение числа a и основания b');
Readln(a,b);
Writeln('Значение Log(a) = ', Log(a,b):0:2);
Writeln('Введите значение a0');
Readln(a0);
Writeln('Введите значение an ');
Readln(an);
Writeln('Введите значение n');
Sumar(a0,an,n,s);
Writeln('Сумма арифметической прогрессии при
a0=',a0:0:2,'  an=',an:0:2,' n=',n,' равна ',s:0:2);
Readln
{TODO-oUser-cConsole Main : Insert code here }
End.
```

Данная программа запрашивает число x и вычисляет для него значения Tg(x), числа a и b и вычисляет для них значения $\mathrm{Log}_b(a)$, а затем требует ввести три числа: первый и последний члены арифметической прогрессии и количество всех членов арифметической прогрессии. На основании введенных значений вычисляется сумма арифметической прогрессии. Все эти вычисления выполняются с помощью модуля Ar. Таким образом, можно создавать собственные модули, расширяя библиотеку стандартных процедур и функций с учетом специфики решаемых задач.

ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Практическое занятие № 1

ОСНОВЫ РАЗРАБОТКИ ПРОГРАММ ЛИНЕЙНОЙ, РАЗВЕТВЛЯЮЩЕЙСЯ И ЦИКЛИЧЕСКОЙ СТРУКТУРЫ В ЯЗЫКЕ ПАСКАЛЬ

Типы данных. Выражения. Арифметические и логические операции в Паскале. Оператор присваивания

Пример 1. Записать на языке Паскаль выражение

$$U = \frac{a\sqrt[3]{e^{x-a} + b^2}}{Sin(b-x) + \lg(a^2 + x^2)}$$

Пример 2. Целой переменной S присвоить сумму цифр трехзначного целого числа k.

```
(S:=k div 100 + k mod 100 div 10 + k mod 10)
```

Пример 3. Присвоить целой переменной h третью от конца цифру в записи положительного целого числа k (например, если k=130985, то h=9).

```
(h:= k mod 1000 div 100)
```

Оператор условия. Оператор выбора

Пример 4. Записать указанное действие в виде одного условного оператора:

$$y = \begin{cases} \cos^2 x & при \quad 0 < x < 2 \\ 1 - \sin x^2 & иначе \end{cases}$$

Пример 5. Записать указанное действие в виде условного оператора:

$$Y=\begin{cases}1, & x<0\\ 2+\sin(2x+1), & 0\le x<3\\ -1, & x\ge 5\end{cases}$$

Пример 6. На предприятии установлен повышающий коэффициент за стаж работы в отрасли в соответствии с таблицей. Составить программу для определения оклада сотрудника, отработавшего определенное количество лет.

Коэффициент	1,1	1,2	1,3	1,4	1,7	2
Стаж	0-3	4-6	7-10	11-15	16-20	>20

Оклад = Тарифная заработная плата * Повышающий коэффициент.
Значение тарифной заработной платы задать самостоятельно.

Текст программы на Паскале.

```
Program Staz;
Uses Sysutils, Windows;
Const T_Zp=200000;
Var Oklad,Tk:Real;
    Stag:Byte;
Begin
SetconsoleoutputCP(1251);
   Write('Введите полное количество отработанных лет - ');
     Readln(Stag);
   Case Stag Of
     0..3 : Tk:=1.1;
     4..6 : Tk:=1.2;
     7..10: Tk:=1.3;
     11..15 :Tk:=1.4;
     16..20 :Tk:=1.7;
        else Tk:=2
   end;
   Oklad:=T_Zp*Tk;
   Writeln('Оклад работника, отработавшего ',Stag,' лет,
составляет ',Oklad:5:2,' рублей');
Readln
End.
```

Результат выполнения программы.

Введите полное количество отработанных лет – 5
Оклад работника, отработавшего 5 лет, составляет 240000.00 рублей

Операторы повтора

Оператор повтора FOR

Пример 7. Вычислить сумму элементов конечного ряда:

$$S = \sum_{n=1}^{8} (-1)^n \frac{x^{n+2}}{(n+1)\cdot n!}$$

$$S = \sum_{n=2}^{6} \frac{(x+0.5)^{2n}}{(2n-1)!} \cos nx$$

Оператор повтора WHILE

Пример 8. Индекс производства непродовольственных товаров в зависимости от года выпуска можно описать следующей зависимостью:

$$y = \begin{cases} -0.28*x^3 + 5.39*x^2 - 22.78*x + 131.9 & x < 2000 \\ 1.34*x + 91.3 & x \geq 2000, \end{cases}$$

где x – год выпуска продукции.

Определить индексы производства непродовольственных товаров за период с 1990 по 2005 год.

Текст программы

```
Program Var1;
Var x,a,b,god:integer;
            Y:real;
 Begin
 Writeln('Vvedite granicy vremennogo riada a,b');
 Readln(a,b);
 Writeln('God     Index');
 x:=1;
 For god:=a To b Do
  Begin
   If x<=10 Then Y:=-0.28*exp(3*ln(x))+5.39*sqr(x)-
26.78*x+131.9
            Else Y:=1.34*x+91.3;
   Writeln(god:4,'':3,Y:0:2);
   x:=x+1;
  End;
    End.
```

Пример 9. Вычислить значение функции S на заданном интервале t [0; 5] для t изменяющегося с шагом Δt = 0,5. (Значение a принять равным 1.5).

$$S = \begin{cases} \dfrac{e^{1/2}}{a\sqrt{t^2+4}}, & t \leq 2 \\ a\sqrt{t+1}, & t > 2 \end{cases}$$

Оператор повтора REPEAT

Пример 10. Вычислить сумму элементов бесконечного ряда с заданной точностью ε.

$$S = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{(2n+1)!}$$

Условием окончания вычислений считать $|S_n - S_{n-1}| < \varepsilon$, где ε = 0,00001. Значение x задать самостоятельно.

Вложенные циклы

Пример 11. Составить программу для вычисления суммы элементов ряда $S = \sum_{n=1}^{10} \frac{\ln^n 3}{n!} * x^n$ и функции y = 3^x на заданном интервале $0.1 \le x \le 1$ с заданным шагом h = 0.1.

Задания и упражнения для самоподготовки

Упражнение 1. Можно ли утверждать, что в Паскале значение выражения (1/3)*3-1 = 0 ?

Упражнение 2. Найти значение каждой из следующих переменных:

- `x = 6*(trunc (3.8) + 3.3);`
- `x = trunc(6.8) + round(6.8)`
- `x = trunc(-6.8) + round(-6.8)`
- `x = round(0.5) + round(-0.5)`

Каким типом данных должен быть описан x?

Упражнение 3. Вычислить значение выражений:

- `3*7 div 2 mod 7/3 - trunc (sin(1);`
- `3*7 div 2 mod 7/3-trunc(sin(Pi/2)+0.1);`
- `succ(round(5/2)-pred(3));`

Упражнение 4. Определить тип выражения (целый или вещественный):

- `1 + 0,0`
- `20/4`
- `sqr(4)`
- `sqr(5.0)`
- `sqrt(16)`
- `sin(0)`

- succ(-2)
- trunc(-3.14)

Упражнение 5. Присвоить целой переменной h первую цифру из дробной части положительного вещественного числа x (если x = 32,597 h = 5).

Упражнение 6. Идет k-ая секунда суток. Определить, сколько полных часов (h) и минут (m) прошло к этому моменту. Сколько осталось секунд? (например, если k = 13257, то h = 3, m = 40 → 3*3600 + 40*60 + 57)

- h := k div 3600
- m := k mod 3600 div 60
- s := k mod 3600 mod 60

Упражнение 7. Что будет напечатано программой:

```
Program aba;
Var a,b:integer;
Begin read(a,b,a); writeln(a,b,a); end.
```

Если для ввода заданы числа 1, 2 и 3?

Упражнение 8. Дано натуральное число n. Найти сумму первой и последней цифры этого числа.

Упражнение 9. Написать программу, которая печатает true или false в зависимости от того, больше число e^{π} или π^{e} или нет (числа e и π описать константами).

Упражнение 10. Записать на языке Паскаль следующие формулы:

1	$Y = \frac{a - b\sqrt{x + 2a}}{b \cdot e^{a^2 - x^2}}$	6	$Z = \frac{a \ln\left(b^2 + t^2\right)}{e^{-t} + b \cdot t}$
2	$U = \frac{a\sqrt[3]{e^{x-a} + b^2}}{Sin(b - x) + \lg\left(a^2 + x^2\right)}$	7	$F = \frac{a \cdot tg\left(b - z\right) - b \cdot z}{\sqrt{2^{z+a} + b^2 \cdot z^2}}$
3	$S = \frac{e^{a-t} + b \cdot \ln t}{Ctg\left(at + b\right)}$	8	$Y = \frac{\sqrt{S^2 + a^2}}{a \cdot Cos\left(S - b\right) + b \cdot Sin\left(S + b\right)}$
4	$F = \frac{a \cdot \lg\left(t^2 + b^2\right)}{a \cdot Sin^2 t + b \cdot Sin^2 t}$	9	$S = \frac{\sqrt{\lvert y^2 - a^2 \rvert} + b \cdot Siny}{a \cdot e^{y-b}}$
5	$U = \frac{a \cdot \ln x}{\lg\lvert x - a \rvert + b}$	10	$Z = \frac{a \cdot Sin^2 t - b \cdot Cos 2t}{a \cdot e^{t^2 - b}}$

Упражнение 11. Записать последовательность операторов для решения задачи:

по номеру y (y>0) некоторого года определить c – номер его столетия (учесть, что началом XX столетия был 1901 а не 1900 год).

Упражнение 12. Если n = 3, то какое значение будет иметь переменная f после выполнения следующего составного оператора? (ответ 6)

```
Begin f:=1; i:=2;
1: if i>n then goto 9;
   f:=f*I; i:=i+1; goto 1;
9: end.
```

Упражнение 13.

- Вычислить f = 10! каждым из трех операторов цикла.
- Вычислить:
  - Y = (2n - 1)! =1*2*3*4*5*….*(2n-1), при n>0;
  - Y = 2n! = 2*4*6*…*2n, при n>0;
  - Y = n! = 1*2*3*…*n при n>0.

Упражнение 14. Какие выходные данные будут получены в результате выполнения следующего цикла? Почему?

```
for v:=72 downto 0 do
begin
v:= trunc(v/2);
write(v:2);
end;
```

(Результат: 36, 17, 8, 3, 1, 0)

Составьте программу таким образом, чтобы результатами были значения *36 18 9 4 2 1*

Упражнение 15. Не используя стандартные функции (за исключением abs), вычислить с заданной точностью eps > 0:

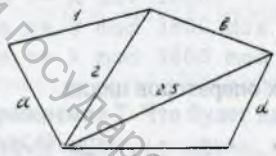
- $y = e^x = x/1! + x^2/2! + x^3/3! + \ldots + x^n/n! + \ldots$
- $y = \cos x = 1 - x^2/2! + x^4/4! - \ldots + (-1)^n x^{2n}/(2n)! + \ldots$
- $y = \ln(x+1) = x - x^2/2 + x^3/3 - \ldots + (-1)^{n-1} x^n/n + \ldots (|x| < 1)$.

Практическое занятие № 2

ОСНОВЫ РАЗРАБОТКИ ПОДПРОГРАММ В ЯЗЫКЕ ПАСКАЛЬ

Пример 1

Вычислить площадь пятиугольника, изображенного на рисунке, где a, b, c, d – длины стороны (действительные числа).



Решение

Для вычисления площади пятиугольника достаточно вычислить площади треугольников со сторонами (a, 1, 2), (2, b, 2.5), (2.5, d, c), а затем их сложить. Для вычисления площади треугольника используется формула Герона $s=\sqrt{p(p-x)(p-y)(p-z)}$, где p – полупериметр, вычисляемый по формуле $p=\frac{x+y+z}{2}$, а x, y, z – длины сторон треугольника. Поэтому логично оформить вычисление площади треугольника через подпрограмму с параметрами, а в основной программе сделать обращения к ней.

Использование подпрограммы-процедуры

```
Program SP;
  Var a,b,c,d,S1,S2,S3,S:real;
(*****************************)
Procedure GERON(x,y,z:real;
                Var SG:real);
Var P:real;
Begin
 P:=(x+y+z)/2;
 SG:=Sqrt(p*(p-x)*(p-y)*(p-
z));
End;
(*****************************)
Begin
  ClrScr;
  Writeln('Ввведите a,b,c,d');
```

Использование подпрограммы-функции

```
Program SF;
 Var a,b,c,d,S:real;
(*****************************)
Function
GERON(x,y,z:real):real;
Var P, SG:real;
Begin
P:=(x+y+z)/2;
SG:=Sqrt(p*(p-x)*(p-y)*(p-z));
 GERON:=SG;
End;
(*****************************)
Begin
ClrScr;
Writeln('введите a,b,c,d');
```

```
    Readln(a,b,c,d);
    GERON(1,a,2,s1);
  GERON(2,b,2.5,S2);
  GERON(c,2.5,d,S3);
  S:=S1+S2+S3;
  Writeln('Площадь S=',S:6:3);
  Readln;
    End.
```

```
Readln(a,b,c,d);
S:=GERON(1,a,2)+GERON(2,b,2.5)
+GERON(c,2.5,d);
Writeln('Площадь  S=',S:6:3);
Readln;
End.
```

Пример 2

Составить программу для вычисления определенного интеграла по методу Симпсона.

$$I = \int_{in}^{ik} \frac{2t}{\sqrt{1-\sin(2t)}}$$

Вычисление подинтегральной функции реализовать с помощью функции, имя которой передается как параметр.

Значение определенного интеграла по формуле Симпсона вычисляется по формуле:

```
ISimps=2*h/3*(0.5*F(A)+2*F(A+h)+F(A+2*h)+2*F(A+3*h)+...+2*F(B
-h)+0.5*F(B))
```

где A и B – нижняя и верхняя границы интервала интегрирования,

N – число разбиений интервала интегрирования,

h = (B-A)/N, причем N должно быть четным.

Решение

```
program Project2;
type
    Func= function(x: Real): Real;
 var I,TN,TK:Real;
    N:Integer;
{$F+}
 Function Q(t: Real): Real;
   begin
     Q:=2*t/Sqrt(1-Sin(2*t));
   end;
{$F-}
 Procedure Simps(F:Func; a,b:Real; N:Integer; var INT:Real);
   var sum, h: Real;
       j:Integer;
   begin
     if Odd(N) then N:=N+1;
     h:=(b-a)/N;
     sum:=0.5*(F(a)+F(b));
     for j:=1 to N-1 do
```

```
      sum:=sum+(j mod 2+1)*F(a+j*h);
      INT:=2*h*sum/3
  end;
begin
  WriteLn(' введите TN,TK,N');
  Readln(TN,TK,N);
  Simps(Q,TN,TK,N,I);
  WriteLn('I=',I:8:3);
  readln
end.
```

Задания и упражнения для самоподготовки

Задание 1. Написать функцию, которая вычисляет сопротивление цепи, состоящей из двух резисторов. Параметрами являются величины сопротивлений и вид соединения (последовательное или параллельное). Функция должна проверять корректность параметров: если неверно указан тип соединения, то функция должна возвращать `-1`.

Задание 2. Написать функцию, которая вычисляет значение a^b (a – вещественное, b – натуральное).

Задание 3. Написать функцию «факториал» и программу, использующую эту функцию для вывода таблицы факториалов.

Задание 4. Написать функцию Dohod, которая вычисляет будущее значение (БЗ) вклада. Исходными данными для функции являются: первоначальная величина вклада, процентная ставка (годовых)[1] и срок вклада.

К-нт наращивания вклада =(1 + процентная ставка)$^{\text{срок вклада}}$.

БЗ вклада = величина вклада * К-нт наращивания.

Задание 5. Вычислить `z = (sign(x) + sign(y))*sign(x+y)`, где

$$sign(a) = \begin{cases} -1 & при \quad a < 0 \\ 0 & при \quad a = 0 \\ 1 & при \quad a > 0 \end{cases}$$

Задание 6. По вещественному x вычислить величину

`th(x)*tg(x+1)-tg`^{2}`(2+th(x-1))`

$$th(a) = \frac{e^a - 1}{e^a + 1}$$

[1] Величину процентной ставки задавать в форме вещественного числа (например ставку 14%, как 0,14).

Задание 7. Написать функцию «Степень» для вычисления x^n (x – вещественное, n – натуральное) и использовать ее для вычисления b = $2.7^k + (a+1)^{-5}$.

Задание 8. Даны отрезки a, b, c, d. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь этого треугольника. Определить процедуру PRINTPL (x,y,z), печатающую площадь треугольника со сторонами x,y,z, если такой треугольник существует.

Задание 9. Вычислить F(m,n) = n!*m!*(n+m)!, где n и m – неотрицательные числа. Описать функцию для вычисления факториала числа.

Задание 10. По вещественному числу a > 0 вычислить величину

$$\frac{\sqrt[3]{a}-\sqrt[6]{a^2+1}}{1+\sqrt[7]{3+a}}$$

Корень $y=\sqrt[k]{x}$ вычислить в подпрограмме.

Практическое занятие №3

ПРАКТИЧЕСКОЕ ИСПОЛЬЗОВАНИЕ СТРУКТУРИРОВАННЫХ ТИПОВ ДАННЫХ В ЯЗЫКЕ ПАСКАЛЬ

Пример 1. (См. лекция 6, пример 2, пример 3)

Пример 2. Обработка многомерного массива

Определить средний тарифный разряд рабочих и работ, если известно число разрядов рабочих N = 5, число разрядов работ M = 3.

Средний тарифный разряд рабочих

$$RSR = \frac{\sum_{i=1}^{n}(R_i \cdot P_i)}{\sum_{i=1}^{n} P_i}$$

Средний тарифный разряд работ

$$SSR = \frac{\sum_{j=1}^{m}(S_j \cdot T_j)}{\sum_{j=1}^{m} T_j}$$

Здесь R – вектор разрядов рабочих, R = {2, 3, 4, 5, 6};

P – вектор количества рабочих каждого разряда, P = {2, 4, 5, 3, 3};

S – вектор разрядов работ, S = {2, 4, 5};

T – вектор трудоемкости работ по разрядам, T={120, 180, 210}.

Сделать вывод, обеспечен ли участок квалифицированными кадрами. Если RSR < SSR, то квалификация рабочих не соответствует квалификации работ.

Текст программы Uchastok на языке Паскаль.

```
Program Uchastok;
 Uses CRT;
 Type mas=array[1..10] of real;
 Var P,R,S,T: mas;
     RSR,SSR: real;
     n,m: integer;
{Процедура ввода элементов одномерного массива}
Procedure Vvod(dl:integer;var v:mas);
  Var i:integer;
  Begin
  For i:=1 to dl Do
   Begin
    Writeln('Введите',I:2,'элемент массива');
    Readln(v[i]);
   End;
  End;
```

```
{Процедура вывода элементов одномерного массива}
 Procedure Vyvod(dl:integer; v :mas);
  Var i:integer;
  Begin
  For i:=1 to dl Do
   Begin
    Write(v[i]:3:1,'':2);
   End;
  End;
{Функция вычисления суммы элементов одномерного массива}
 Function SUM(dl:integer;v:mas):real;
  Var i:integer; S:real;
  Begin
  S:=0;
  For i:=1 to dl Do
   S:=S+v[i];
  SUM:=S;
  End;
{Функция вычисления суммы произведений элементов двух векторов}
 Function SUM_PR(dl:integer;v,w:mas):real;
  Var i:integer;    S:real;
  Begin
  S:=0;
  For i:=1 to dl Do
   S:=S+v[i]*w[i];
  SUM_PR:=S;
  End;
{Текст основной программы}
 Begin
 ClrScr;
 Writeln('Введите количество разрядов работ - m');
 Readln(m);
 Writeln('Введите количество разрядов рабочих - n');
 Readln(n);
 Writeln('Введите массив разрядов рабочих R');
 Vvod(n,R);
 Writeln('Введите количество рабочих каждого разряда P');
 Vvod(n,P);
 Writeln('Введите массив разрядов работ S');
 Vvod(m,S);
 Writeln('Введите трудоемкость работ по разрядам T');
 Vvod(m,T);
 RSR:=SUM_PR(n,R,P)/SUM(n,P);
 SSR:=SUM_PR(m,S,T)/SUM(m,T);
 ClrScr;
 If RSR>=SSR Then Writeln('Участок обеспечен  рабочими')
          Else  Writeln('Участок не обеспечен рабочими');
 Writeln('Средний разряд рабочих RSR=',RSR:4:1);
```

```
  Writeln('Средний разряд работ SSR=',SSR:4:1);
  Writeln('Массив разрядов рабочих');
  Vyvod(n,R);
  Writeln;
  Writeln('Количество рабочих каждого разряда');
  Vyvod(n,P);
  Writeln;
  Writeln('Массив разрядов работ');
  Vyvod(m,S);
  Writeln;
  Writeln('Трудоемкость работы по каждому разряду');
  Vyvod(m,T);
  Readln;
  End.
```

Результаты

Участок обеспечен квалифицированными рабочими
Средний разряд рабочих RSR=4.1
Средний разряд работ SSR=3.9
Массив разрядов рабочих
2 3 4 5 6
Количество рабочих каждого разряда
2 4 5 3 3
Массив разрядов работ
2 4 5
Трудоемкость работы по каждому разряду
120 180 210

Пояснения к программе.

Для ввода и вывода массивов используются процедуры *Vvod* и *Vyvod*, написанные для формального массива *v* длиной *dl*. Используя эти процедуры, вводятся и выводятся массивы исходных данных (R, P, S, T).

В подпрограмме-функции *SUM* описывается вычисление суммы элементов формального массива *v* длиной *dl*.

В подпрограмме-функции *SUM_PR* описывается вычисление суммы произведений формальных массивов *v* и *w* длиной *dl*.

Далее в программе эти функции используются для вычисления по заданным формулам значений среднего разряда рабочих *RSR* и среднего разряда работ *SSR* для фактических параметров–значений исходных данных.

Задания и упражнения для самоподготовки

Задание 1. Дана последовательность чисел $a_1, \ldots, a_n$ ($N < 20$). Найти среднее арифметическое положительных элементов этой последовательности. Получить последовательность $b_1, \ldots, b_n$ по правилу: элементы с четными

индексами увеличить на эту величину, элементы с нечетными индексами уменьшить на эту величину. Отсортировать полученный массив по возрастанию. Исходный и полученный массивы вывести на экран, используя подпрограмму-процедуру.

Задание 2. Дана последовательность чисел $a_1, \ldots, a_n$ и $b_1, \ldots, b_n$ ($N < 20$). Что больше – число нулей в последовательности А или число единиц в последовательности В? Отсортировать по убыванию массивы А и В, вывести их и результаты вычислений на экран.

Задание 3. Дана последовательность чисел a 1 , …, a n ($N < 20$). Заменить все отрицательные элементы их квадратами. Отсортировать полученный массив по убыванию. Исходный и преобразованный массивы и количество отрицательных элементов вывести на экран.

Задание 4. Дана последовательность целых чисел $a_1, \ldots, a_n$ ($N = 20$). Найти $\min(a_2, a_4, \ldots)$ и $\min(a_1, a_3, \ldots)$. Исходный массив, последовательности $(a_2, a_4, \ldots)$ и $(a_1, a_3, \ldots)$, а также значения минимумов вывести на экран.

Задание 5. Дана последовательность действительных чисел $a_1 \ldots a_n$. Найти минимальный и максимальный элемент последовательности и увеличить каждый элемент последовательности на среднее арифметическое минимального и максимального элемента. Исходный и преобразованный массивы, значения минимального и максимального элементов и их среднее арифметическое вывести на экран.

Задание 6. Дана вещественная квадратная матрица А размерностью M*M. Найти сумму элементов матрицы, расположенных под / над главной диагональю и сравнить эти значения.

Задание 7. Дана вещественная матрица А размерностью M*N. Найти в каждой строке матрицы минимальный (максимальный) элемент. Из этих элементов сформировать дополнительный столбец.

Задание 8. Дана вещественная матрица А размерностью M*N. Найти в каждом столбце матрицы произведение положительных элементов. Из полученных значений сформировать дополнительную строку.

Задание 9. Дана квадратная матрица V(4,4). Найти и вывести строку с минимальной суммой элементов и строку с максимальной суммой элементов, вычислить произведение этих строк.

Задание 10. В матрице А размерности M*N удалить строку, содержащую максимальный элемент. Исходную и полученную матрицы и максимальный элемент вывести на экран.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ[2]

Лабораторная работа № 1

ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Цель работы: приобрести практические навыки программирования линейных вычислительных процессов.

Общие сведения

Объявление переменных

• Каждая переменная программы должна быть объявлена в разделе описаний.

• Формат описания переменной:

```
Var <имя переменной >:<тип переменной>
```

• В имени переменной можно использовать буквы латинского алфавита, цифры и знак подчеркивания (первым символом должна быть буква или знак подчеркивания).

• Основными числовыми типами языка Паскаль являются `integer` (целый) и `real` (вещественный).

Оператор присваивания

• Оператор присваивания «`:=`» предназначен для изменения значений переменных.

• Значение выражения в левой части оператора присваивания зависит от типа операндов и операций, выполняемых над операндами.

Например, если переменная `n`, объявленная как `integer`, имеет значение 32767, то в результате выполнения оператора `n:=n+1` значение переменной `n` будет равно -32768.

Ввод-вывод данных

• Ввод данных обеспечивают операторы

```
read(список ввода) и readln(список ввода).
```

[2] Номера вариантов заданий по лабораторной работе выбираются студентами в соответствии с № ПК. Задания, помеченные знаком *, относятся к заданиям повышенной трудности.

- Вывод данных обеспечивают операторы
  `write(список вывода)` и `writeln(список вывода)`.
- Для вывода значений целого и символьного типа используется формат
  `Write(ch:p)`
- Для вывода значений вещественного типа используется формат
  `Write(ch:p:q)`,

где `p` – всего позиций экрана;
`q` – из них после запятой.

Программы линейной структуры

- Программы с линейной структурой являются простейшими и используются, как правило, для реализации простых вычислений по формулам.
- В программах с линейной структурой операторы выполняются последовательно один за другим.

Задания для самостоятельного выполнения

Задание 1. Отладить программы, реализующие решения заданий 2, 3 и 5 практического занятия № 1.

Задание 2. Составить программу, реализующую вычисление значения функции:

№	Задание	№	Задание
1	$a = \ln\left(y^{-\sqrt{\lvert x\rvert}}\right)\cdot\left(\sin x + e^{x+y}\right)$ $b = \sqrt{c\left(\sqrt{y}+x^2\right)}\cdot\left(\cos x - \lvert c-y\rvert\right)$	6	$a = 0{,}5x^5 + 3\cos(x+y) + e^{-0{,}1yz} - \sqrt{\lvert x\cdot y\rvert}$ $b = \sqrt{\left\lvert\frac{-3\cdot tg(x)\cdot \lg\left(x^4+y\right)}{e^{-x}}+1\right\rvert}$
2	$a = arctg(x) - \frac{3}{5}e^{x+y} + 0{,}5\frac{\lvert x+y\rvert}{(x+y)^2}$ $b = \frac{e^{\lvert x-y\rvert}\cdot tg(z)}{arctg(y)+\sqrt{x}} + \ln x$	7	$a = \sqrt{e^x + tg(x) + 1}\cdot\left(\lg y + \cos(x\cdot y) + \sqrt[3]{x}\right)$ $b = \frac{\lg x - e^{x+y}}{\sqrt{2+y^2+\lvert x^3-\ln y\rvert}}$
3	$a = \frac{(\cos x - \sin y)^3}{\sqrt{tg(z)}} + \ln^2(x\cdot y\cdot z)$ $b = y^x + \sqrt{\lvert x\rvert + e^3} - \frac{z^3\cdot\sin^2 y}{y + \frac{z^2}{y-x}}$	8	$a = \sqrt{12x^4 - 3x^3 + 4x^2 - 5x + 6} - \lg^2 z$ $b = \lg\lvert 1 - 2x + 3x^2 - 4x^3\rvert + \frac{\sqrt{\lvert x\rvert}}{z}$
4	$a = \frac{1+\cos(x+y)}{\left\lvert e^x - \frac{2y}{1+x^2y^2}\right\rvert}\cdot x^3 + arctg(y)$	9	$a = \frac{2\cos\left(x-\frac{1}{6}\right)}{\frac{1}{2}+\sin^2 y} - \frac{1}{\left\lvert\frac{x^2}{y+x^3}\right\rvert}$

	$b = 2 + \frac{x^2}{\sqrt{2}} + \frac{\lvert y\rvert^3}{\sqrt{2}} + \frac{z^4 \cdot (\ln x + 1) \cdot \sqrt{2}}{\sqrt{3}}$		$b = \frac{x \cdot y \cdot z - y \cdot \lvert x + \sqrt{z}\rvert}{10^7 + \sqrt[4]{\lg 4}}$
5	$a = \left((1+y) \cdot \sqrt{\sin^2 z + 3} - \frac{\lvert y - x\rvert}{5} \right)^3$ $b = \ln\left\lvert y - \sqrt{\lvert x\rvert} \right\rvert \cdot \left(x - \frac{y}{z + \frac{x^2}{4}} \right)$	10	$a = \frac{(x+y-z)^3 - (x-y+z)^2 + \sqrt{x+y+z}}{\log_2(tg(z))}$ $b = \frac{\frac{x}{y} \cdot (z+x) \cdot e^{\lvert x-y\rvert} + \ln(1+x)}{\sin^2 y - (\sin x \cdot \sin y)^2}$

Задание 3. Составить программы решения следующих задач:

1. Вычислить, сколько ПК можно купить на имеющуюся сумму (ден. ед.) и какая сумма останется.

2. Ввести число дней, а затем пересчитать эту величину в недели и дни.

3. Преобразовать введенное с клавиатуры дробное число в денежный формат. (Например, число 120,5 должно быть преобразовано к виду 120 руб. 50 коп.

4. Идет `k` -ая секунда суток. Определить, сколько полных часов (`h`) и минут (`m`) прошло к этому моменту. Сколько осталось секунд? (Например, если `k = 13257`, то `h = 3, m = 40` → `3*3600 + 40*60 + 57`).

5. По номеру `y` (`y>0`) некоторого года определить `c` – номер его столетия (учесть, что началом XXI столетия был 2001 а не 2000 год).

6. Составить программу, которая печатает `true` или `false` в зависимости от того, больше число e^{π} или π^{e} или нет (числа `e` и `π` описать константами).

7. Переменной `x` присвоить корень уравнения `arcsin(1+ln x) = a`, если такой существует.

8. Вычислить сопротивление электрической цепи, состоящей из `n` параллельно соединенных сопротивлений.

9. Вычислить, сколько заготовок круглой формы можно изготовить из куска материала длиной $a_м$ и шириной $b_м$, если радиус заготовки $R_{см}$? Центры заготовок должны располагаться на одной линии. Определить площадь материала, идущего в отходы. Напечатать площадь заготовок и площадь отходов.

10. Вычисить доход по вкладу. Исходные данными являются: величина вклада, процентная ставка (годовых)[3] и срок вклада.
`К-нт наращивания вклада =(1 + процентная ставка)`$^{\text{срок вклада}}$.
`Доход = величина вклада * К-нт наращивания - величина вклада.`

11. Дано двузначное число. Найти сумму и произведение его цифр.

[3] Величину процентной ставки задавать в форме вещественного числа (например ставку 14%, как 0,14).

12. Даны целые положительные числа A, B, C. На прямоугольнике размера A x B размещено максимально возможное количество квадратов со стороной C (без наложений). Найти количество квадратов, размещенных на прямоугольнике, а также площадь незанятой части прямоугольника.

13. Дано целое число, лежащее в диапазоне 1-999. Вывести его строку-описание вида «четное двузначное число», «нечетное трехзначное число» и т. д.

14. Составить программу, которая заданный в годах возраст выразит в днях:

a. без учета неполных и високосных лет;

b. с учетом неполных и високосных лет.

15. Составить программу, которая определит, можно ли отпустить со склада X кг гвоздей ящиками по N кг и M кг, не нарушая упаковки?

16. Дано натуральное число n. Найти сумму первой и последней цифры этого числа.

17. Дано натуральное число n. Переставить местами первую и последнюю цифры этого числа.

18. Дано натуральное число n. Дописать к нему цифру k в конец и в начало (если это возможно, т.е. результат не выйдет за диапазон допустимых значений), или сообщить о невозможности выполнения операции.

19. Найти наибольшую (наименьшую) цифру в записи данного натурального числа.

20. Дано натуральное число n. Переставить его цифры так, чтобы образовалось максимальное (минимальное) число, записанное теми же цифрами.

Лабораторная работа № 2

ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Цель работы: приобрести практические навыки использования оператора условного перехода IF … Then … Else и оператор выбора CASE на примере программирования разветвляющихся вычислительных процессов.

Общие сведения

Оператор условного перехода If … Then … Else

- Инструкция **If** используется для выбора одного из двух направлений дальнейшего хода программы.
- Выбор последовательности инструкций осуществляется в зависимости от значения *условия* – выражения логического типа, записанного после **If.**
- Инструкция, записанная после **Else,** выполняется в том случае, если значение выражения *условие* = ЛОЖЬ, во всех остальных случаях выполняется инструкция, следующая за условием.
- Если при соблюдении или несоблюдении *условия* надо выполнить несколько инструкций программы, то эти инструкции заключаются в операторные скобки.
- При помощи вложенных одна в другую нескольких инструкций **If** можно реализовать множественный выбор.

Оператор выбора CASE

- Оператор выбора **case** предназначен для выбора одного из нескольких возможных направлений дальнейшего хода программы.
- Выбор последовательности инструкций осуществляется в зависимости от равенства значений переменной-селектора константе, указанной после слова **case.**
- Если значение выражения-селектора не равно ни одной из констант, записанных после слова **case**, то выполняются инструкции, расположенные после слова **else.**
- В качестве выражения-селектора можно использовать переменную целого (**integer**) или символьного (**char**) типа.

Задания для самостоятельного выполнения

1. Составить программу, которая проверяет, является ли год високосным.

2. Составить программу расчета стоимости покупки с учетом скидки:

- скидка 1 % предоставляется, если стоимость покупки выше 50000 руб.,
- скидка 3 % предоставляется, если стоимость покупки выше 100000 руб,
- скидка 5 % предоставляется, если стоимость покупки выше 200000 руб,
- дополнительно предоставляется скидка 2% на покупки, сделанные в первый и последний час работы магазина.

3. Составить программу проверки знания даты основания Витебска. В случае неправильного ответа пользователя программа должна выводить правильный ответ.

4. Составить программу проверки знания даты основания Витебска. Программа должна вывести вопрос и три варианта ответа. Пользователь должен выбрать правильный ответ и ввести его номер. В случае неправильного ответа пользователя программа должна выводить правильный ответ.

5. Составить программу, которая проверяет, является ли введенное пользователем целое число четным.

6. Составить программу вычисления стоимости разговора по телефону с учетом 20 % скидки, предоставляемой по субботам и воскресеньям. Стоимость 1 минуты по обычному тарифу составляет 30 руб. *(Дни недели: понедельник – 1, ... , воскресенье – 7).*

7. Составить программу, которая вычисляет оптимальный вес для пользователя, сравнивает его с реальным и выдает рекомендацию, на сколько кг необходимо поправиться или похудеть. Оптимальный вес вычисляется следующим образом:

для мужчин Рост (см) – 100;

для женщин Рост (см) – 100 – 10 % * (Рост (см) - 100).

8. Составить программу, которая запрашивает у пользователя номер месяца и затем выводит соответствующее название времени года. В случае ввода недопустимого числа программа должна вывести сообщение «Ошибка ввода данных».

9. Составить программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: «Рабочий день», «Суббота» или «Воскресенье».

10. Составить программу, которая запрашивает Ваше желаемое месячное жалование и рассчитывает годовой доход. (Выплаты можно учесть из годовой суммы). Предусмотрите комментарий, устраивают ли Вас Ваши доходы.

11. Составить программу, которая подсчитывает сумму расчета за электроэнергию, исходя из следующих тарифов:

- первые 240 квт/ч: $0.1143 за 1 квт/ч;

- следующие 300 квт/ч: $0.1329 за 1 квт/ч;
- свыше 540 квт/ч: $0,1402 за 1 квт/ч.

12. Составить программу, которая подсчитывает, сколько потребуется банок краски, чтобы покрасить заданное количество квадратных фунтов поверхности, если известна площадь, для покраски которой достаточно одной банки краски. Учесть, что дробить банки невозможно. Если требуется, например, 1.2 банки краски, ответом должно быть число 2.

13. Составить программу, вычисляющую площадь треугольника по заданным сторонам. Предусмотреть возможность некорректного ввода.

14. Написать программу, которая требует ввода числа часов, отработанных за неделю, а затем выводит на печать заработную плату без вычетов, сумму выплачиваемых налогов и зарплату после вычетов. Принять следующие условия:

- Базовая часовая ставка = $10.00 /час;
- Переработка (при превышении 40 часов) = в полтора раза;
- Налоговая ставка : 15% с первых $300
  20% со следующих $150
  25% с остальной суммы.

15. Составить программу, которая рассчитывает Вашу скорость пробега дистанции в км/час. Исходными данными для программы являются: длина дистанции в метрах, время пробега дистанции в минутах и секундах, Ваш возраст. Сравните свою скорость со средней для вашего возраста и получите рекомендации.

Возраст, лет	Скорость км/час
до 20	12
от 20 до 30	10
от 30 до 40	8
от 40 до 50	6
после 50	5

16. Составить программу вычисления стоимости междугороднего разговора по телефону (цена 1 минуты определяется расстоянием до города, в котором находится абонент). Исходными данными для программы являются код города и длительность разговора. Учтите 10 % скидку на разговоры в ночное время с 21.00 час. до 07.00 час.

Город	Код	Цена минуты (руб)
Минск	0172	20
Москва	095	800
Киев	135	350
Симферополь	065	650
Владивосток	423	1500

17. Составить программу для определения стоимости 1 секунды работы со 2 по 6 разряд, если стоимость 1 секунды работы 1 разряда равна 3 рубля. Тарифные коэффициенты для каждого разряда представлены в таблице.

Разряд	2	3	4	5	6
Тарифный коэффициент	1,56	2,03	2,16	2,27	2,5

Стоимость 1 секунды работы i-го разряда = Стоимость 1 с. 1 разряда * Тарифный коэффициент.

18. На предприятии установлен повышающий коэффициент за стаж работы в отрасли в соответствии с таблицей. Составить программу для определения оклада сотрудника, отработавшего определенное количество лет.

Стаж	0-3	4-6	7-10	11-15	16-20	>20
Коэффициент	1,1	1,2	1,3	1,4	1,7	2

Оклад = Тарифная заработная плата * Повышающий коэффициент.

Значение тарифной заработной платы задать самостоятельно.

19. Составить программу для определения размера начислений сотрудникам отдела продаж. Размер начислений зависит от количества продаж и вычисляется в соответствии с таблицей. Значение минимального размера начислений задать самостоятельно.

Продажи	<15	15-25	26-30	31-50	51-75	76-100	>100
Коэффициент начислений	1	1,2	1,3	1,4	1,5	1,7	2

Начислено = Коэффициент начислений * минимальный размер начислений.

20. Составить программу для определения экологического налога на выбросы загрязняющих веществ.

Сумма налога = Фактически сожжено топлива * Ставка налога * Понижающий коэффициент.

Данные для расчета экологического налога

Вид топлива	Фактически сожжено топлива за отчетный месяц, $м^3$	Ставка налога за выбросы, руб.	Сумма налога к уплате, руб.
1		1155	
2		1128	
3		735	

Понижающий коэффициент

Сожжено топлива за отчетный месяц, $м^3$	Понижающий коэффициент за передвижные источники выбросов
До 200	0,9
От 201 до 500	0,85
Более 500	0,7

Лабораторная работа № 3

ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Цель работы: приобрести практические навыки использования операторов повтора For, While и Repeat на примере программирования циклических вычислительных процессов.

Общие сведения

Оператор повтора For

- Оператор повтора **For** используется для организации циклов с фиксированным, известным во время разработки программы, числом повторений.
- Количество повторений цикла определяется начальным значением переменной-счетчика и условием завершения цикла.
- Переменная счетчик может быть любого скалярного типа, кроме вещественного.

Оператор повтора While

- Число повторений инструкций цикла **While** определяется ходом выполнения программы.
- Инструкции цикла **While** выполняются до тех пор, пока значение выражения, стоящего после слова **While,** остается истинным.
- Для завершения цикла **While** в теле цикла должен быть хотя бы один оператор, выполнение которого влияет на условие завершения цикла.
- Цикл **While** – это цикл с предусловием, то есть возможна ситуация, при которой инструкции тела цикла не будут выполнены ни разу.
- В случае составного оператора в теле цикла цикл **While** требует операторные скобки.
- Цикл **While,** как правило, используется для организации приближенны вычислений, в задачах поиска и обработки данных, вводимых с клавиатуры или файла.

Оператор повтора Repeat

- Число повторений инструкций цикла **Repeat** определяется ходом выполнения программы.

- Инструкции цикла **Repeat** выполняются до тех пор, пока значение выражения, стоящего после слова **Until,** не станет истинным.
- Для завершения цикла **Repeat** в теле цикла должен быть хотя бы один оператор, выполнение которого влияет на условие завершения цикла.
- Цикл **Repeat** – это цикл с постусловием, то есть инструкции цикла будут выполняться хотя бы один раз.
- В случае составного оператора в теле цикла цикл **Repeat** не требует операторных скобок.
- Цикл **Repeat,** как правило, используется для организации приближенных вычислений, в задачах поиска и обработки данных, вводимых с клавиатуры или файла.

Задания для самостоятельного выполнения

1. Составьте программу, которая подсчитывает, какую сумму денег вы получите за n дней, если в первый день Вы получаете $1, во второй – $4, в третий – $9 и т.д.

2. Какие выходные данные будут получены в результате выполнения следующего цикла? Почему?

```
for v: = 72 downto 0 do
begin
v:= trunc(v/2);
write(v:2);
end;
```

Результат: 36, 17, 8, 3, 1, 0

Составьте программу таким образом, чтобы результатами были значения *36 18 9 4 2 1*

3. Не используя стандартные функции (за исключением abs), вычислить значение y для n=50:

$y = e^x = x/1! + x^2/2! + x^3/3! + \ldots + x^n/n! + \ldots$

4. Не используя стандартные функции (за исключением abs), вычислить вычислить значение y для n=50:

$y = \cos(x) = 1 - x^2/2! + x^4/4! - \ldots + (-1)^n x^{2n}/(2n)! + \ldots$

5. Не используя стандартные функции (за исключением abs), вычислить вычислить значение y для n=50:

$y = \ln(x+1) = x - x^2/2 + x^3/3 - \ldots + (-1)^{n-1} x^n/n + \ldots (|x| < 1)$.

6. Составьте программу, которая вычисляет среднее арифметическое последовательности дробных чисел, вводимых с клавиатуры. После ввода последнего числа программа должна вывести минимальное и максимальное число последовательности. Количество чисел последовательности должно задаваться во время работы программы.

7. Вычислить число сочетаний из m по n по формуле:

$$c = \frac{n!}{m!(n-m)!}$$

где n!= 1*2*3*...*(n-1)*n, целые числа n и m ввести с клавиатуры.

8. Составить программу, реализующую возведение натурального числа в любую степень.

9. Вычислить сумму элементов конечного ряда:

Вариант	Задание
1	$S = \sum_{n=1}^{8} (-1)^n \frac{x^{n+2}}{(n+1)\cdot n!}$
2	$S = \sum_{n=1}^{9} (-1)^{n+1} \frac{x^{2n+1}\cdot n!}{2n+3}$
3	$S = \sum_{n=1}^{8} (-1)^n \frac{x^{2n-1}\cdot \sin nx}{(2n+1)!}$
4	$S = \sum_{n=2}^{6} \frac{x^{2n}\cdot \cos nx}{(n+2)!}$
5	$S = \sum_{n=1}^{10} (-1)^{n+1} \frac{x^{n+2}}{(n+3)!}$

Вариант	Задание
6	$S = \sum_{n=2}^{6} \frac{(x+0.5)^{2n}}{(2n-1)!} \cos nx$
7	$S = \sum_{n=1}^{8} \frac{x^{2n+1}}{(2n+1)!}$
8	$S = \sum_{n=1}^{8} (-1)^n \frac{x^{2n-1}}{(n+1)!}$
9	$S = \sum_{n=1}^{10} \frac{(x+2)^{n+1}}{(n+1)!} x^3$
10	$S = \sum_{n=1}^{8} (-1)^{n+1} \frac{x^{2n+2}}{(2n+2)!}$

10. Индекс производства непродовольственных товаров в зависимости от года выпуска можно описать следующей зависимостью:

$$y = \begin{cases} -0.28*x^3 + 5.39*x^2 - 22.78*x + 131.9 & x < 2000 \\ 1.34*x + 91.3 & x \geq 2000, \end{cases}$$

где x – год выпуска продукции.

Определить индексы производства непродовольственных товаров за период с 1990 по 2005 год[4].

11. Индекс общего объема продукции легкой промышленности в зависимости от года выпуска можно описать следующей зависимостью:

$$y = \begin{cases} 18.59*x + 60{,}45 & x \leq 1998 \\ 0.084*x^3 - 1.4*x^2 + 5.5*x + 103.2 & x > 1998, \end{cases}$$

где x – год выпуска продукции.

Определить индексы общего объема продукции легкой промышленности за период с 1995 по 2005 год.

12. Индекс общего объема продукции машиностроительной отрасли в зависимости от года выпуска можно описать следующей зависимостью:

$$y = \begin{cases} 8.69*x + 77.67 & x \leq 1999 \\ 0.0091*x^3 + 0.156*x^2 + 0.956*x + 110.57 & x > 1999, \end{cases}$$

где x – год выпуска продукции.

[4] См. приложение к лабораторной работе 3, стр. 129 или [8].

Определить индексы общего объема продукции машиностроительной отрасли за период с 1995 по 2005 год.

13. Зависимость рентабельности реализованной промышленной продукции от года реализации продукции можно описать следующей зависимостью:

$$y = \begin{cases} 7.01 * 1.108^{x} & x \le 1999 \\ 0.0032 * x^{4} - 0.086 * x^{3} + 0.75 * x^{2} - 1.97 * x + 11.06 & x > 1999, \end{cases}$$

где x – год реализации продукции.

Найти рентабельность реализации промышленной продукции за период с 1992 по 2005 год.

14. Среднегодовую численность производственно-промышленного персонала по Республике Беларусь можно описать следующей зависимостью:

$$y = \begin{cases} 5.95 * x^{3} - 110.61 * x + 1649 & x \le 1999 \\ -18.9 * x + 1343.3 & x > 1999, \end{cases}$$

где x – год.

Найти среднегодовую численность производственно-промышленного персонала за период с 1990 по 2005 год.

15. Объем выпуска продукции в сопоставимых ценах в процентах к предыдущему году можно описать следующей зависимостью:

$$y = \begin{cases} 74.1 * 1.06^{x} & x \le 1999 \\ 0.0053 * x^{2} + 1.24 * x + 93.68 & x > 1999, \end{cases}$$

где x – год выпуска продукции.

Определить объемы выпуска продукции в сопоставимых ценах за период с 1992 по 2005 год.

16. Объемы производства химических волокон и нитей по Республике Беларусь в зависимости от года выпуска можно описать следующей зависимостью:

$$y = \begin{cases} -3.65 * x^{3} + 35.6 * x^{2} - 103.75 * x + 281.01 & x \le 1999 \\ -4.91 * x + 249 & x > 1999, \end{cases}$$

где x – год выпуска продукции.

Определить, сколько тонн химических волокон и нитей выпущено предприятиями Республики Беларусь за период с 1995 по 2005 год.

17. Объем производства тканей, м^2, предприятиями Республики Беларусь в зависимости от года выпуска можно описать следующей зависимостью:

$$y = \begin{cases} -21.8 * x + 203 & x \le 1998 \\ 0.186 * x^{3} - 3.139 * x^{2} + 18.67 * x + 225.73 & x > 1998, \end{cases}$$

где x – год выпуска продукции.

Определить объем производства тканей предприятиями Республики Беларусь за период с 1995 по 2005 год.

18. Объем производства трикотажных изделий, млн. шт., предприятиями Республики Беларусь в зависимости от года выпуска можно описать следующей зависимостью:

$$y = \begin{cases} 1.29*x^2 - 0.714*x + 36.6 & x \le 1999 \\ 0.075*x^2 - 4.29*x + 74.85 & x > 1999, \end{cases}$$

где x – год выпуска продукции.

Определить объем производства трикотажных изделий предприятиями Республики Беларусь за период с 1995 по 2005 год.

19. Объем выпуска ковровых изделий, млн. м^2, предприятиями Республики Беларусь в зависимости от года выпуска можно описать следующей зависимостью:

$$y = \begin{cases} 1.25*x + 3.05 & x \le 1999 \\ 0.011*x^3 - 0.19*x^2 + 0.69*x + 8.19 & x > 1999, \end{cases}$$

где x – год выпуска продукции.

Определить объем производства ковровых изделий предприятиями Республики Беларусь за период с 1995 по 2005 год.

20. Вычислить значение функции на заданном интервале.

Вариант	Функция	Исходные данные
1	$S = \begin{cases} \dfrac{e^{\frac{1}{2}}}{a\sqrt{t^2+4}}, & t \le 2 \\ a\sqrt{t+1}, & t > 2 \end{cases}$	[0; 5]; Δt = 0,5; a = 1,5
2	$Y = \begin{cases} a \cdot \cos(2x), & x < 1 \\ a\sqrt{4 - \lg(x)} & x \ge 1 \end{cases}$	[-1;3]; Δx = 0,25; a = 2,5
3	$Y = \begin{cases} \cos(x), & x < 1 \\ a\sqrt{2 + \ln(x)}, & x \ge 1 \end{cases}$	[-1;3]; Δx = 0,4; a = 2
4	$Y = \begin{cases} a \cdot \sin^2(x), & x \le 0 \\ \dfrac{arctg\ (x)}{\ln\left(1 + x^2\right)}, & x > 0 \end{cases}$	[-1;1]; Δx = 0,2; a = 3
5	$S = \begin{cases} \dfrac{a \cdot ctg\ (t)}{e^t}, & t < 4 \\ e^{a-t}\cos(t), & t \ge 4 \end{cases}$	[2;6]; Δt = 0,4; a = 2,4
6	$Y = \begin{cases} \sqrt{1 + \lg\left(x^2 + a\right)}, & x < 3 \\ \sqrt{a^2 + \lg(x)}, & x \ge 3 \end{cases}$	[1;5]; Δx = 0,5; a = 3,5
7	$S = \begin{cases} a \cdot e^{\sin(t)}, & t \le 3 \\ a \cdot \lg(t), & t > 3 \end{cases}$	[2;4]; Δt = 0,2; a = 5,2
8	$Y = \begin{cases} a\sqrt{4 + \ln\lvert x\rvert}, & x \le 2,5 \\ \sqrt{a^2 + \cos^2(x)}, & x > 2,5 \end{cases}$	[1;4]; Δx = 0,25; a = 4

9	$S = \begin{cases} \sqrt{a^2 + \lg^2 \lvert t+3 \rvert}, & t < -1 \\ e^{\cos(a-t)}, & t \geq -1 \end{cases}$	[-2;2]; Δt = 0,25; a = 2,5
10	$Y = \begin{cases} a \cdot (\cos^2 x \cdot \sin^2 x), & x < -1 \\ \sqrt{a^2 + tg^2 x}, & x \geq -1 \end{cases}$	[-2;5]; Δx = 0,5; a = 3,2

21. Вычислить сумму элементов бесконечного ряда с заданной точностью ε. Условием окончания вычислений считать $|S_n - S_{n-1}| < \varepsilon$. ε = 0,00001.Значение x задать самостоятельно.

Вариант	Задание
1	$S = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{(2n+1)!}$
2	$S = \sum_{n=1}^{\infty} \frac{x^{n+1}}{(n+1)!}$
3	$S = \sum_{n=1}^{\infty} \frac{x^{2n+1} \cdot \sin nx}{2n!}$
4	$S = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^{2n+1}}{n!}$
5	$S = \sum_{n=2}^{\infty} (-1)^n \frac{x^n \cdot \sin^{n+1}(x+2)}{(2n+2)!}$

Вариант	Задание
6	$S = \sum_{n=2}^{\infty} (-1)^n \frac{x^{n+1}}{(2n-1)!}$
7	$S = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{(x+3)^{n+1} \cdot \sin^n x}{2n!}$
8	$S = \sum_{n=1}^{\infty} (-1)^n \frac{x^n}{(n+1)!}$
9	$S = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{\sin^{n+1}(x+2)}{(n+2)!}$
10	$S = \sum_{n=1}^{\infty} (-1)^n \frac{x^{2n}}{(2n+1)!}$

Вложенные циклы

22. Составить программу для вычисления суммы элементов ряда (столбец 2) и функции (столбец 3) на заданном интервале (столбец 4) с заданным шагом (столбец 5).

Вар.	Сумма	Функция	Интервал	Шаг
1	$S = \sum_{n=1}^{10} \frac{\ln^n 3}{n!} * x^n$	y= 3^x	$0,1 \leq x \leq 1$	h=0.1.
2	$S = \sum_{n=1}^{10} \frac{x^{2n+1}}{(2n+1)!}$	y=Sin(x)	$\frac{\pi}{5} \leq x \leq \frac{9\pi}{5}$	$h = \frac{\pi}{10}$
3	$S = \sum_{n=1}^{25} \frac{Cos(n\frac{\pi}{4})}{n!} x^n$	$y = e^{xCos(\frac{\pi}{4})} \times Cos(x)$	$0,1 \leq x \leq 1$	h=0.1
4	$S = \sum_{n=1}^{20} x^n Sin(n\frac{\pi}{4})$	$y = \frac{xSin\frac{\pi}{4}}{1 - 2xCos\frac{\pi}{4} + x^2}$	$0,1 \leq x \leq 0.8$	h=0.05

5	$S=\sum_{n=1}^{10}\frac{1}{2n+1}\bullet\left(\frac{x-1}{x+1}\right)^{2n+1}$	$y=\frac{1}{2}\ln(x)$	$0,2\le x\le 1$	h=0.2.
6	$S=\sum_{n=1}^{10}(-1)^{n+1}\bullet\frac{x^{2n+1}}{4n^2-1}$	$y=\frac{1+x^2}{2}\bullet arctg(x)-\frac{x}{2}$	$0,1\le x\le 1$	h=0.1.
7	$S=\sum_{n=1}^{10}\frac{x^{2n}}{(2n)!}$	$y=\frac{e^x-e^{-x}}{2}$	$0,1\le x\le 1$	h=0.1
8	$S=\sum_{n=1}^{10}(-1)^n\frac{n^2+1}{n!}\times\left(\frac{x}{2}\right)^n$	$y=\left(\frac{x^2}{4}+\frac{x}{2}+1\right)\times e^{x/2}$	$0,1\le x\le 1$	h=0.1
9	$S=\sum_{n=1}^{15}\frac{Cos(2n-1)x}{(2n-1)^2}\times n!$	$y=\frac{\pi^2}{8}-\frac{\pi}{4}\times\lvert x\rvert$	$\frac{\pi}{5}\le x\le\pi$	h=$\frac{\pi}{10}$.
10	$S=\sum_{n=1}^{20}x^n Sin(n\frac{\pi}{4})$	$y=\frac{1}{2}\ln(x)$	$0,2\le x\le 1$	h=0.2

Приложение к лабораторной работе №3

Пример табулирования функции (Вариант 1)

```
Program Var1;
Var x,a,b,god:integer;
              Y:real;
 Begin
 Writeln('Vvedite granicy vremennogo riada a,b');
 Readln(a,b);
 Writeln('God    Index');
 x:=1;
 For god:=a To b Do
  Begin
   If x<=10 Then Y:=-0.28*exp(3*ln(x))+5.39*sqr(x)-
26.78*x+131.9
            Else Y:=1.34*x+91.3;
   Writeln(god:4,'':3,Y:0:2);
   x:=x+1;
  End;
End.
```

Лабораторная работа № 4

ПРАКТИЧЕСКОЕ ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ ПОЛЬЗОВАТЕЛЯ

Цель работы: приобрести практические навыки разработки модульных программ с использованием подпрограмм-процедур и подпрограмм-функций пользователя.

Общие сведения

Процедуры и функции пользователя организуются самим программистом в соответствии с синтаксисом языка и представляют собой локальный блок. Предварительное описание процедур и функций пользователя обязательно.

Все параметры, используемые в подпрограммах пользователя, подчиняются определенным правилам.

1. Каждый идентификатор должен быть описан перед тем, как он будет использован.
2. Областью действия идентификатора является блок, в котором он описан.
3. Идентификаторы в блоке должны быть уникальными.
4. Один и тот же идентификатор может быть по-разному определен в каждом отдельном блоке.

Если процедура или функция пользователя совпадает по имени со стандартной процедурой или функцией, то будет выполняться подпрограмма пользователя, а стандартная игнорироваться.

Особенности, присущие функциям:

1. Обязательно задается тип возвращаемого результата.
2. В результате работы функции получается только один результат.
3. В разделе операторов блока должен присутствовать хотя бы один оператор, влияющий на значение функции.

Обращение к функции должно происходить только из какого-либо выражения, *например*, `Y:=One(X);` а не `One(X)`.

Для передачи имен процедур и функций в качестве параметров в языке Паскаль введен новый тип данных - процедурный или функциональный, в зависимости от того, что описывается. Описание процедурных и функциональных типов производится в разделе описания типов:

```
type
      FuncType = Function(z: Real): Real;
      ProcType = Procedure (a,b: Real; var x,y: Real);
```

Функциональный и процедурный тип определяется как заголовок процедуры и функции со списком формальных параметров, но без имени. Можно определить функциональный или процедурный тип без параметров, например:

```
type
     Proc = Procedure;
```

После объявления процедурного или функционального типа его можно использовать для описания формальных параметров - имен процедур и функций. Кроме того, необходимо написать те реальные процедуры или функции, имена которых будут передаваться как фактические параметры. Эти процедуры и функции должны компилироваться в режиме дальней адресации с ключом {$F+} (см. Приложение к лабораторной работе № 4. Пример).

Задания для самостоятельного выполнения

1. Написать функцию, которая вычисляет сопротивление цепи, состоящей из двух резисторов. Параметрами являются величины сопротивлений и вид соединения (последовательное или параллельное). Функция должна проверять корректность параметров: если неверно указан тип соединения, то функция должна возвращать -1.

2. Написать функцию Dohod, которая вычисляет доход по вкладу. Исходными данными для функции являются: величина вклада, процентная ставка (годовых) и срок вклада.

К-нт наращивания вклада = (1 + процентная ставка)$^{\text{срок вклада}}$.
БЗ вклада = величина вклада * К-нт наращивания.

3. Вычислить z = (sign(x) + sign(y))*sign(x+y), где

$$sign(a) = \begin{cases} -1 & при \quad a < 0 \\ 0 & при \quad a = 0 \\ 1 & при \quad a > 0 \end{cases}$$

4. По вещественному x вычислить величину

th(x)*tg(x+1)-tg^2(2+th(x-1)), где $th(a) = \dfrac{e^a - 1}{e^a + 1}$

5. Написать функцию «Степень» для вычисления x^n (x – вещественное, n – натуральное) и использовать ее для вычисления b = 2.7^k + (a+1)$^{-5}$.

6. Написать функцию «факториал» и программу, использующую эту функцию для вывода таблицы факториалов.

7. Вычислить `F(m,n)=n!*m!*(n+m)!`,
где `n` и `m` – неотрицательные числа.
Описать функцию для вычисления факториала числа.

8. По вещественному числу `a>0` вычислить величину

$$\frac{\sqrt[3]{a}-\sqrt[6]{a^2+1}}{1+\sqrt[7]{3+a}}$$

Корень $y=\sqrt[k]{x}$ вычислить в подпрограмме.

9. Вычислить

$$s=\frac{\min(\max(x,y),\max(y,z))}{\max(y,z)}$$

Вычисление `max(a,b)` и `min(a,b)` оформить подпрограммами.

10. Вычислить

$$s=\frac{\max(\min(x,y),\min(y,z))}{\max(y,z)}$$

Вычисление `max(a,b)` и `min(a,b)` оформить подпрограммами.

11. Составить программу для вычисления определенного интеграла по методу трапеций.

Вариант	Условие	Вариант	Условие
1	$\int_1^4 \frac{\ln^2 x}{x}dx$	11	$\int_0^1 \frac{x\cdot arctgx}{\sqrt{1+x^2}}dx$
2	$\int_2^3 \frac{1}{x\lg x}dx$	12	$\int_0^1 \frac{1}{(3\sin x+2\cos x)^2}dx$
3	$\int_1^2 \frac{x^3}{3+x}dx$	13	$\int_{\pi/6}^{\pi/3} tg^2x+ctg^2xdx$
4	$\int_0^1 xe^x\sin xdx$	14	$\int_0^{3/4} \frac{1}{(x+1)\sqrt{x^2+1}}dx$
5	$\int_1^{2.5} \frac{1}{x^2}\sin\frac{1}{x}dx$	15	$\int_0^{1.5} \frac{e^x(1+\sin x)}{1+\cos x}dx$

6	$\int_0^2 \frac{1}{\sqrt{9+x^2}}dx$	16	$\int_0^1 \frac{x^2-1}{(x^2+1)\sqrt{x^4+1}}dx$
7	$\int_1^2 \frac{\sqrt{x^2-0.16}}{x}dx$	17	$\int_1^3 x^x(1+\ln x)dx$
8	$\int_1^{1.5} \sin x \ln(tg(x))dx$	18	$\int_0^{1.9999} x^2\sqrt{4-x^2}\,dx$
9	$\int_1^3 x^x(1+\ln x)dx$	19	$\int_0^3 \arcsin\sqrt{\frac{x}{1+x}}dx$
10	$\int_0^{\ln 2} \sqrt{e^x-1}dx$	20	$\int_1^2 \left(\frac{\ln x}{x}\right)^3 dx$

Значение определенного интеграла по методу трапеций вычисляется по формуле:

$$\int_a^b f(x)dx \approx \frac{b-a}{2n}\left(y_0 + y_n + \sum_{i=1}^{n-1} 2y_i\right),$$

где a и b - нижняя и верхняя границы интервала интегрирования,

n - число разбиений интервала интегрирования,

h=(b-a)/n, причем n должно быть четным.

Вычисление подынтегральной функции реализовать двумя способами:

- с помощью подпрограммы-функции (см. Приложение к лабораторной работе № 4. Задание 11а, стр. 134);
- с помощью подпрограммы-функции, имя которой передается как параметр (см. Приложение к лабораторной работе № 4. Задание 11б, стр. 134).

12. По заданным вещественным числам a, b, c и d (a<b, c<d) вычислить

$$\int_c^d arctg^2(x)dx + \int_a^b \sin(e^{10x})dx$$

Интегралы вычислять приближенно по формуле трапеций. Вычисление подынтегральной функции реализовать с помощью функции, имя которой передается как параметр (см. Приложение. Задание 12).

Приложение к лабораторной работе №4

Решение типового примера

Составить программу для вычисления определенного интеграла

$$I = \int_{m}^{tk} \frac{2t}{\sqrt{1-\sin(2t)}}$$

по методу Симпсона. Вычисление подынтегральной функции реализовать с помощью функции, имя которой передается как параметр.

Формула Симпсона для вычисления значения определенного интеграла имеет вид:

```
Isimps = 2*h/3*(0.5*F(A)+2*F(A+h)+F(A+2*h)+2*F(A+3*h)+...
                +2*F(B-h)+0.5*F(B))
```

где A и B нижняя и верхняя границы интервала интегрирования,

N - число разбиений интервала интегрирования,

h=(B-A)/N, причем N должно быть четным.

```
program Project2;
type Func= function(x: Real): Real;
 var I,TN,TK:Real;
     N:Integer;
{$F+}
 Function Q(t: Real): Real;
   begin
     Q:=2*t/Sqrt(1-Sin(2*t));
   end;
{$F-}
 Procedure Simps(F:Func;a,b:Real;N:Integer;var INT:Real);
   var sum, h: Real;
       j:Integer;
   begin
     if Odd(N) then N:=N+1;
     h:=(b-a)/N;
     sum:=0.5*(F(a)+F(b));
     for j:=1 to N-1 do
       sum:=sum+(j mod 2+1)*F(a+j*h);
       INT:=2*h*sum/3
   end;
 begin
   WriteLn(' введите TN,TK,N');
   Readln(TN,TK,N);
   Simps(Q,TN,TK,N,I);
```

```
  WriteLn('I=',I:8:3);
  readln
end.
```

Решение варианта 1 задания 11

Вычислить $\int_1^4 \frac{\ln^2 x}{x} dx$

а) использование подпрограммы-функции

```
Program int_tr;
Uses crt;
Var a,b,int,x:real; n:integer;
(*********************************)
Function Fun(z:real):real;
 Begin
  Fun:=sqr(ln(z))/z;
 End;
(*********************************)
Function Integral(a1,b1:real;n1:integer):real;
 Var i:integer; h1,x,s:real;
 Begin
  h1:=(b1-a1)/n1; s:=0;
  For i:=1 to n1-1 do
   Begin
    x:=a1+i*h1;
    s:=s+2*fun(x);
   End;
  Integral:=h1/2*(fun(a1)+fun(b1)+s);
 End;
(*********************************)
Begin
  ClrScr;
  Writeln('Введите a,b,n'); Readln(a,b,n);
  Int:=Integral(a,b,n);
  Writeln('Int=',int:7:4);
  Readln;
 End.
```

б) использование подпрограммы-функции, имя которой передается как параметр

```
Program int_tr;
Uses crt;
Type Func=function(x:real):real;
Var a,b,int:real; n:integer;
```

```
{$F+}
 (*********************************)
 Function Fun(x:real):real;
  Begin
   Fun:=sqr(ln(x))/x;
  End;
 (*********************************)
{$F-}
 Function Integral(f:Func;a1,b1:real;n1:integer):real;
  Var i:integer; h1,x,s:real;
  Begin
   h1:=(b1-a1)/n1;
   s:=0;
   For i:=1 to n1-1 do
    Begin
     x:=a1+i*h1;
     s:=s+2*f(x);
    End;
   Integral:=h1/2*(f(a1)+f(b1)+s);
  End;
 (*********************************)
Begin
  CLrScr;
  Writeln('Введите a,b,n'); Readln(a,b,n);
  Int:=Integral(Fun,a,b,n);
  Writeln('Int=',int:7:4);
  Readln;
  End.
```

Решение задания 12 (a=0, b=1, c=0, d= Pi)

```
Program int_tr;
Uses crt;
Type Func=function(x:real):real;
Var a,b,c,d,int:real; n1,n2:integer;
{$F+}
(*********************************)
Function F1(x:real):real;
 Begin
  F1:=sqr(arctan(x));
 End;
 (*********************************)
{$F-}
{$F+}
(*********************************)
```

```
Function F2(x:real):real;
 Begin
  F2:=sin(exp(10*x));
 End;
(*********************************)
{$F-}
Function Integral(f:Func;a1,b1:real;m:integer):real;
 Var i:integer; h1,x,s:real;
 Begin
  h1:=(b1-a1)/m;
  s:=0;
  For i:=1 to m-1 do
   Begin
    x:=a1+i*h1;
    s:=s+2*f(x);
   End;
  Integral:=h1/2*(f(a1)+f(b1)+s);
 End;
(*********************************)
 Begin
 CLrScr;
  Writeln('Введите a,b,c,d,n1,n2');
  Readln(a,b,c,d,n1,n2);
  Int:=Integral(F1,a,b,n1)+Integral(F2,c,d,n2);
  Writeln('Int=',int:7:4);
  Readln;
 End.
```

Лабораторная работа № 5

ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ

Цель работы: приобрести практические навыки разработки программ обработки массивов данных, выполнения поиска и сортировки в одномерных массивах.

Общие сведения (см. материалы лекции 6)

Массив – это структурированный (регулярный) тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип [1].

Для описания массива в языке Pascal предназначено словосочетание ARRAY OF (массив из...). Описание массива возможно двумя способами.

Способ 1. Используя раздел описания типов.

```
Type <имя типа> = array [тип индекса] of <тип компонент>;
     Var <идентификатор>: <имя типа>;
```

Например,
```
Type Mas = array[1..100] of real;
     Var  M1,M2:Mas;
```

Способ 2. Используя раздел описания переменных (Var) без представления типа в разделе описания типов данных:

```
Var
<идентификатор>:  array  [тип  индекса]  of  <тип компонент>;
```

Например,
```
Var M1, M2: array[1..100] of real;
    V1: array[1..50] of integer;
```

Задания для самостоятельного выполнения

Составить блок-схемы и программы решения следующих задач:

- для сортировки массивов использовать процедуру с параметрами;
- основные действия над массивами описывать подпрограммами.

1. Подсчитать количество положительных и количество отрицательных элементов массива X(20).

2. Вычислить значения функции z=sqrt((xi+ai)/2), если xi и ai - элементы массивов, состоящих из десяти элементов каждый. Отсортировать полученный массив.

3. Записать в массив Y(20) квадраты чисел, указанных в массиве N(20). Отсортировать массив Y.

4. Вывести на печать отдельно положительные элементы массива X(16) и отдельно отрицательные элементы. Отсортировать полученные массивы.

5. Вывести на печать первый отрицательный элемент массива A(20) и его номер, полагая, что в массиве есть хотя бы один отрицательный элемент.

6. Найти сумму положительных и сумму отрицательных элементов массива X(11).

7. Задан массив X, состоящий из пяти элементов. Вычислить значения функции y=0.5*ln(x) при значениях аргумента, заданных в массиве X, и поместить их в массив Y. Напечатать массивы X и Y в виде двух отсортированных столбцов.

8. Вычислить сумму и разность массивов A(20) и B(20). Отсортировать полученные массивы. Результат распечатать в виде двух параллельных столбцов.

9. Задан вектор x = {x_1, . . . , x_{12}}. Найти максимальное и минимальное значение этого вектора и поменять их местами.

10. Задан вектор x = {x_1, . . . , x_{12}}. Найти сумму тех из компонент вектора x, для которых выполнено условие $a \le x_i \le b$, где a и b – заданные числа.

11. Вывести на печать номера элементов массива Y(16), удовлетворяющих условию 0 < yi < 1.

12. Составить программу для вычисления вектора z = {z_1, . . . , z_{10}}, равного сумме двух векторов

x={x_1,...,x_{10}} и y={y_1,...,y_{10}} по формуле $z_i=x_i+y_i$, i=1,...,10.

Вектор x_i={x_1,...,x_{10}} задан. Значения компонент вектора y вычислить по формуле: y_i=0.7*Sin(0.5i)+1.3*Cos(i+1), i=1,...,10. Отсортировать массивы x_i , y_i и z_i и вывести их в виде трех параллельных строк.

13. Составить программу для вычисления суммы

$h = \sum_{i=1}^{n} (a_i - b_i)^2$, где a_i и b_i вычисляются по формулам (n=5):

$$a_i = \begin{cases} i, \text{если } i - \text{нечетное} \\ i^2, \text{если } i - \text{четное} \end{cases} \qquad b_i = \begin{cases} i^2, \text{если } i - \text{нечетное} \\ i^3, \text{если } i - \text{четное} \end{cases}$$

Вывести массивы a_i и b_i в виде двух параллельных строк.

14. Составить программу для вычисления вектора

x={$x_1, \ldots, x_{10}$} . Каждая компонента x_j вектора определяется по формуле:

$$x_j = \begin{cases} arctg\dfrac{\sqrt{j+2}}{n+3}, если\ tg(j) \le 3 \\ e^{j+Cos(x)}, если\ tg(j) > 3 \end{cases} \qquad j = 1, \ldots, 10.$$

15. Задан массив y из n целых чисел. Сформировать массив z таким образом, чтобы в начале массива шли отрицательные элементы массива y, затем положительные и, наконец, нулевые.

16. Записать элементы массива x в обратном порядке. Упорядочить массив x по убыванию. Вывести эти три массива.

17. Задан массив из n элементов. Сформировать массивы номеров положительных и отрицательных элементов. Вывести исходный массив и массивы номеров положительных и отрицательных элементов.

18. Удалить из массива X, состоящего из n элементов, первые четыре нулевых элемента. Вывести исходный и полученный массивы.

19. Записать элементы массива X в массив Y, сдвинув элементы массива X вправо на три позиции. При этом три элемента с конца массива X переместить в начало. Определить номера максимального и минимального элементов массивов X и Y.

20. Составить программу для вычисления функции d = t^2 * Sin(t) при t, изменяющемся от 1 до 1,5 с шагом 0,1. Вычисленные значения организовать в векторы и вывести на печать. Далее найти средний модуль разности между компонентами вектора d и заданного вектора y = {y_1, . . . , y_6}:

$$L = \frac{\sum_{i=1}^{6} |d_i - y_i|}{6}.$$

Указание: для формирования вектора t_i использовать соотношение: t_i=+0.1*(i-1).

21. Решить уравнение a*x = b для пяти пар значений a и b, заданных в виде двух массивов. Результат поместить в массив X.

22. В массиве X определить количество элементов меньше среднего арифметического значения. Не упорядочивая массив, удалить из него элементы, расположенные между минимальным и максимальным

23. Даны два массива X(10) и Y(8). Сформировать массив Z(18) из положительных элементов массивов X и Y. Если положительных элементов меньше 18, то оставшимся элементам массива Z присвоить значение +1.

24. Записать -1 вместо минимального элемента массива X(11) и +1 - вместо максимального элемента.

25. Найти наибольший среди отрицательных элементов массива C(15).

Лабораторная работа № 6

ОБРАБОТКА МНОГОМЕРНЫХ МАССИВОВ

Цель работы: приобрести практические навыки разработки программ, содержащих процедуры и функции обработки многомерных массивов.

Общие сведения

Процедуры и функции обработки многомерных массивов

Основные операции по обработке данных, организованных в многомерные массивы, рассмотрим на уровне подпрограмм-процедур и подпрограмм-функций, описанных для формальных параметров. В качестве примера использования в составленных ниже программах подпрограмм опишем следующие массивы: *матрицы* A, B, C размера M x N и *одномерные массивы* D и F длиной N ($N \le 10$).

```
Type Matr = Array[1..10,1..10] Of Real;
     Mas = Array[1..10] Of Real;
Var A, B, C: Matr; F,D:  Mas;
```

Ввод матрицы по строкам

```
Procedure Vvod(M,N:Integer; Var
V:Matr);
Var i,j:Integer;
Begin
For i:=1 To M Do
Begin
Writeln('Введите ', i ,' строку
матрицы');
For j:=1 To N Do
 Readln(V[I,J]);
End;
End;
```

Обращение из программы:
Vvod(4,5,A); или *Vvod(3,5,B);*

Вывод матрицы (в виде матрицы)

```
Procedure Vyvod(M, N:Integer;
V:Matr);
Var i,j:Integer;
Begin
For i:=1 To M Do
 Begin
 For j:=1 To N Do
   Write(V[I,J]:4:1,'':2);
 Writeln;
 End;
End;
```

Обращение из программы:
Vyvod(4,5,A); или *Vyvod(3,5,B);*

Вычисление суммы элементов матрицы

Суммирование диагональных элементов матрицы

```
Function
Sum(M,N:Integer;V:Matr):Real;
Var i,j:Integer; S:Real;

Begin
S:=0;
For I:=1 To M Do
For J:=1 To N Do
S:=S+V[I,J];
Sum:=S;
End;
```

Обращение из программы:
Sa:=Sum(4,5,A); или *Sb:=Sum(3,5,B);*

Суммирование матриц

```
Procedure
Sum_Matr(M,N:Integer;V,W:Matr;
Var U:Matr);
Var i,j:Integer;
Begin
For i:=1 To M Do
 For j:=1 To N Do
 U[i,j]:=V[i,j]+W[i,j];
End;
```

Обращение из программы:
Sum_Matr(3,5,A,B,D);

Умножение матрицы на вектор

$Fi = \sum_{j=1}^{M} A_{ij} * D_j$. Длина исходного вектора равна количеству столбцов матрицы. Длина результирующего вектора равна количеству строк матрицы.

```
Procedure
Vector(M,N:Integer;V:Matr;T:Mas;
Var S:Mas);
 Var i,j:Integer;
 Begin
 For i:=1 To M Do
```

```
Function
Sled(M:Integer;V:Matr):Real;
Var i:Integer; S:Real;
Begin
S:=0;
For i:=1 To M Do
S:=S+V[i,i];
Sled:=S;
End;
```

Обращение из программы:
Sla:=Sled(4,A); или *Slb:=Sled(3,B);*

Суммирование матрицы по строкам

```
Procedure
Sum_Str(M,N:Integer;V:Matr;
Var S:Mas);
Var i,j:Integer;
Begin
For i:=1 To M Do
 Begin
  S[i]:=0;
  For j:=1 To N Do
  S[i]:=S[i]+V[i,j]
 End
End;
```

Обращение из программы:
Sum_Str(4,5,A,S_A);
Sum_Str(3,5,B,S_B);

Удаление строки матрицы

Требуется удалить строку с заданным номером К. Все строки, начиная с (К+1)-ой, нужно переместить вверх. Число строк уменьшится на 1.

```
Procedure
Udstr(M,N,Kk:Integer;V:Matr;
Var W:Matr);
 Var i,j:Integer;
 Begin
  W:=V;
```

```
Begin
S[i]:=0;
 For j:=1 To N Do
 S[i]:=S[i]+V[i,j]*T[j];
End;
End;
```

Обращение из программы:
Vector (4,5,A,D,F);

```
 M:= M-1;
 For i:=Kk To M Do
  For j:=1 To N Do
 W[i,j] := W[i+1,j];
End;
```

Обращение из программы:
Udstr (4,5,K,A,D);

Перестановка строк матрицы

Пусть требуется поменять местами К-ую и (К+2)-ую строки.

Способ 1: с использованием вспомогательной переменной Р.

```
Procedure
Perest(N,K:Integer;V:Matr;
Var W:Matr);
 Var i,j:Integer; P:Real;
 Begin
 W:=V;
 For j:=1 To N Do
 Begin
  P:=W[K,J];
  W[k,j] := W[i+2,j];
  W[k+2,j]:=P;
 End;
 End;
```

Обращение из программы:
Perest (5,K,A,D);

Способ 2: с использованием вспомогательного массива P_j: одна из строк пересылается целиком в массив P_j для временного хранения:

```
Procedure
Perest1(N,K:Integer;
V:Matr; Var W:Matr);
 Var i,j:Integer; P:Mas;
  Begin
  W:=V;
  For j:=1 To N Do
   P[j]:=W[k,j];
   For j:=1 To N Do
    W[k,j]:= W[k+2,j];
   For j:=1 To N Do
    W[k+2,j]:=P[j];
   End;
```

Обращение из программы:
Perest1 (5,2 ,A,D);

Поиск минимального (максимального) элемента матрицы

```
Procedure Min_El(M,N:Integer;
V:Matr; Var Min:Real;
Var L,K:Integer);
 Var i,j:Integer;
 Begin
 Min:=V[1,1]; K:=1; L:=1;
 For i:=1 To M Do
  For j:=1 To N Do
  If Min>V[i,j] Then
  Begin
```

Поиск минимального элемента строки (столбца) матрицы

```
Procedure
Min_Str(M,N:Integer;V:Matr;
Var Min:Mas);
Var i,j:Integer;
Begin
For i:=1 To M Do
 Begin
  Min[i]:=V[I,1];
  For j:=2 To N Do
  If Min[i]>V[i,j]
```

```
   Min:=V[i,j];K:=i;L:=j;
  End;
 End;
```

Обращение к процедуре:
Min_El(4,5,A,Min_A,K_A,L_A);

```
      Then Min[i]:=V[i,j];
  End;
 End;
```

Обращение к процедуре:
Min_El(4,5,A,Min_A);

Пример обработки двумерного массива

Определить средний тарифный разряд рабочих и работ, если известно число разрядов рабочих N=5, число разрядов работ M=3. Здесь R – вектор разрядов рабочих, R={2, 3, 4, 5, 6}; P – вектор количества рабочих каждого разряда, P={2, 4, 5, 3, 3}; S – вектор разрядов работ, S={2, 4, 5}; T – вектор трудоемкости работ по разрядам, T={120, 180, 210}.

Сделать вывод, обеспечен ли участок квалифицированными кадрами.

Средний тарифный разряд рабочих

$$RSR = \frac{\sum_{i=1}^{n}(R_i \cdot P_i)}{\sum_{i=1}^{n} P_i}$$

Средний тарифный разряд работ

$$SSR = \frac{\sum_{j=1}^{m}(S_j \cdot T_j)}{\sum_{j=1}^{m} T_j}$$

Если RSR < SSR, то квалификация рабочих не соответствует квалификации работ.

Текст программы Uchastok на языке Паскаль.

```
Program Uchastok;
 Uses CRT;
 Type mas=array[1..10] of real;
 Var P,R,S,T: mas;
     RSR,SSR: real;
     n,m: integer;
{Процедура ввода элементов одномерного массива}
Procedure Vvod(dl:integer;var v:mas);
  Var i:integer;
  Begin
  For i:=1 to dl Do
   Begin
    Writeln('Введите',I:2,'элемент массива');
    Readln(v[i]);
   End;
  End;
{Процедура вывода элементов одномерного массива}
 Procedure Vyvod(dl:integer; v :mas);
  Var i:integer;
  Begin
```

```
      For i:=1 to dl Do
       Begin
        Write(v[i]:3:1,'':2);
       End;
      End;
    {Функция вычисления суммы элементов одномерного массива}
      Function SUM(dl:integer;v:mas):real;
      Var i:integer;
        S:real;
      Begin
      S:=0;
      For i:=1 to dl Do
       S:=S+v[i];
       SUM:=S;
      End;
{Функция вычисления суммы произведений элементов двух векторов}
      Function SUM_PR(dl:integer;v,w:mas):real;
      Var i:integer;
        S:real;
      Begin
      S:=0;
      For i:=1 to dl Do
       S:=S+v[i]*w[i];
       SUM_PR:=S;
      End;
    {Текст основной программы}
      Begin
      ClrScr;
      Writeln('Введите количество разрядов работ - m');
      Readln(m);
      Writeln('Введите количество разрядов рабочих - n');
      Readln(n);
      Writeln('Введите массив разрядов рабочих R');
      Vvod(n,R);
      Writeln('Введите количество рабочих каждого разряда P');
      Vvod(n,P);
      Writeln('Введите массив разрядов работ S');
      Vvod(m,S);
      Writeln('Введите трудоемкость работ по разрядам T');
      Vvod(m,T);
      RSR:=SUM_PR(n,R,P)/SUM(n,P);
      SSR:=SUM_PR(m,S,T)/SUM(m,T);
      ClrScr;
      If RSR>=SSR Then
      Writeln('Участок обеспечен квалифицированными рабочими')
        Else
      Writeln('Участок не обеспечен квалифицированными
рабочими');
```

```
Writeln('Средний разряд рабочих RSR=',RSR:4:1);
Writeln('Средний разряд работ SSR=',SSR:4:1);
Writeln('Массив разрядов рабочих');
Vyvod(n,R);
Writeln;
Writeln('Количество рабочих каждого разряда');
Vyvod(n,P);
Writeln;
Writeln('Массив разрядов работ');
Vyvod(m,S);
Writeln;
Writeln('Трудоемкость работы по каждому разряду');
Vyvod(m,T);
Readln;
End.
```

Результаты

Участок обеспечен квалифицированными рабочими
Средний разряд рабочих RSR=4.1
Средний разряд работ SSR=3.9
Массив разрядов рабочих
2 3 4 5 6
Количество рабочих каждого разряда
2 4 5 3 3
Массив разрядов работ
2 4 5
Трудоемкость работы по каждому разряду
120 180 210

Задания для самостоятельного выполнения

1. Дана вещественная матрица A, размерностью MxN. Найти наибольший элемент матрицы и номер строки и столбца, в которых он находится.

2. Дана матрица B(10,10). Определить сумму элементов, кратных 3, и количество отрицательных элементов.

3. Дана действительная квадратная матрица C(7,7). Найти максимальный элемент в главной диагонали и напечатать строку, в которой он находится.

4. Вычислить общее количество осадков за каждый год, среднее количество осадков за каждый год и среднее месячное количество осадков на основании данных об осадках за несколько лет.

Год / Месяц	январь	февраль	март	апрель	май	июнь	июль	август	сентябрь	октябрь	ноябрь	декабрь
1998	10,2	8,1	6,8	4,2	2,1	1,8	0,2	0,3	1,1	2,3	6,1	7,4
1999	9,2	9,8	4,4	3,3	2,2	0,8	0,4	0,0	0,6	1,7	4,3	5,2
2000	6,6	5,5	3,8	2,8	1,6	0,2	0,0	0,0	0,0	1,3	2,6	4.2
2001	4,3	4,3	4,3	3,0	2,0	1,0	0,2	0,2	0,4	2,4	3,5	6,6

5. Задана квадратная матрица `Y(5,5)`. Определить, где больше четных элементов: выше или ниже главной диагонали?

6. Дана вещественная матрица `A` размерностью `NxN`. Найти след матрицы. Преобразовать исходную матрицу следующим образом: элементы, стоящие над главной диагональю, увеличить на величину следа, под главной диагональю – уменьшить на величину следа.

7. В матрице `A(MxN)` найти строку с максимальной суммой элементов и строку с минимальной суммой элементов. Далее сформировать вектор `B` длиной `2M`, в котором бы чередовались элементы из этих строк.

8. Дана квадратная матрица `A(4,4)`. За один просмотр элементов матрицы `A(4,4)` сформировать вектор `C(4)`, каждый `j`-й элемент которого равен произведению элементов `j`-го столбца исходной матрицы, и вектор `D(4)`, каждый `j`-й элемент которого равен сумме элементов соответствующей строки матрицы A.

9. Задана квадратная матрица `A(5,5)`, матрица `X(5,5)`. Получить $X_1X_5+X_2X_4+X_3X_3+X_4X_2+X_5X_1$, где X_i - наибольшее значение элементов `i`-го столбца исходной матрицы.

10. Дана действительная квадратная матрица `X(5,5)`. Требуется преобразовать матрицу: поэлементно вычесть последнюю строку из всех строк, кроме последней.

11. В данной целочисленной квадратной матрице `A(5,5)` указать индексы всех элементов, имеющих наибольшее значение.

12. Задана квадратная матрица `A (5,5)`. Найти местоположение столбца (номер `L`) строки и с минимальной суммой элементов, а также элемент с минимальным значением в матрице `A`. Далее сформировать вектор `P(5)`, каждый элемент которого равен разности соответствующих элементов `K`-го столбца и `L`-строки, деленной на минимальный элемент матрицы `A`.

13. Задана прямоугольная матрица `A (4,5)`. Найти местоположение (номер `K`) строки с максимальной суммой элементов. Далее сформировать матрицу `B(4,5)`, каждый элемент строки которой равнялся бы элементу соответствующей строки матрицы `A`, деленному на соответствующий элемент `K`-й строки.

14. Заполнить квадратную матрицу `B(10,10)` единицами в шахматном порядке, начиная с верхнего левого угла. Соседние клетки заполнить нулями.

15. Дана система линейных алгебраических уравнений. Найти корни этой системы, используя матричный метод.

16. Даны две прямоугольные матрицы A(3,4) и B(3,4). Найти матрицу C(3,4), элементы которой равны сумме соответствующих элементов матриц A и B, после чего произвести транспонирование полученной матрицы C.

17. Задан двумерный массив A (4,4). Сформировать массив B, состоящий из сумм элементов соответствующих строк массива A. Расположить элементы полученного массива B в порядке возрастания. Вывести на экран исходную матрицу A и отсортированный массив B.

18. Дана вещественная квадратная матрица A(4,4). Найти количество отрицательных элементов под главной диагональю и количество нечётных элементов над ней. Вывести на экран полученные значения и исходную матрицу.

19. В заданной матрице A(4x4) найти число отрицательных элементов на главной диагонали. Элементы чётных строк матрицы увеличить на полученное число, а элементы нечётных строк уменьшить на это число. Количество отрицательных элементов, исходную и преобразованную матрицы вывести на экран.

20. Дана матрица A(4,5). Найти минимальный элемент каждого столбца матрицы A. Полученные элементы расположить в порядке убывания. Вывести на экран исходную матрицу и вектор минимальных значений элементов каждого столбца.

Лабораторная работа № 7

ОБРАБОТКА СТРОК И ЗАПИСЕЙ

Цель работы: приобрести практические навыки разработки программ обработки строк и записей.

Общие сведения

Процедуры и функции обработки строк

`Delete(St,Poz,N)` – удаление N символов строки St, начиная с позиции `Poz`.

`Insert(St1,St2,Poz)` – вставка строки `St1` в строку `St2` начиная с позиции `Poz`.

`Str(IBR, Str)` – преобразование числового значения величины `IBR` и помещение результата в строку `Str`.

`Val(St,IBR,Cod)` – преобразует значение `St` в величину целочисленного или вещественного типа и помещает результат в `IBR`. Значение `IBR` не должно содержать незначащих пробелов. `Cod` – целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, значение `Cod = 0`. Если ошибка обнаружена, `Cod` будет содержать номер позиции первого ошибочного символа, а значение IBR не определено.

`Length(St)` – вычисляет длину в символах строки `St`. Результат имеет целочисленный тип.

`Copy(St,Poz,N)` – выделяет из `St` подстроку длиной `N` символов, начиная с позиции `Poz`.

`Concat(St1,St2,... ,StN)` – выполняет сцепление строк `St1, St2... ,StN` в том порядке, в каком они указаны в списке параметров.

`Pos(St1,St2)` – обнаруживает первое появление в строке `St2` подстроки `St1`. Результат имеет целочисленный тип и равен номеру той позиции, где находится первый символ подстроки `St1`. Если в `St2` подстроки `St1` не найдено, результат равен `0`.

UpCase(Ch) – преобразует строчную букву в прописную. Параметр и результат имеют литерный тип. Обрабатывает буквы только латинского алфавита.

Задания для самостоятельного выполнения

1. Составить программу, производящую подсчет символов, отличных от одинарных и двойных кавычек.

2. Составить программу, которая считывает вводимые символы до тех пор, пока не встретится символ #, и сообщает, сколько раз во входной последовательности встречается сочетание ei.

3. Задан список фамилий студентов. Найти студентов, чьи фамилии заканчиваются на *«ова»* и подсчитать их количество.

4. Задан список названий автомобилей. Подсчитать количество моделей заданной марки.

5. Задан список слов одинаковой длины. Напечатать слово, составленное из последних букв этих слов.

6. Задана дата в формате ДЕНЬ.МЕС.ГОД. Вывести ее в виде текста. (Например 04.11.2010 – «четвертое ноября 2010 года»).

7. Дан текст из 60-ти литер. Напечатать только строчные буквы, входящие в этот текст.

8. Список фамилий представлен в виде: Фамилия Имя Отчество. Преобразовать его к виду: И. О. Фамилия

9. Определить, сколько раз (k) в стоку S входят буквы, перечисленные в константе V.

10. В строке S все вхождения abc заменить на def.

11. Список автомобилистов включает следующие данные: номер машины, название марки машины, год выпуска, фамилию владельца и его адрес. Подсчитать количество владельцев автомобилей, выпущенных после 2005 года, и вывести их список – ФИО, адрес, год.

12. Список автомобилистов включает следующие данные: номер машины, название марки машины, год выпуска, фамилию владельца и его адрес. Подсчитать количество и вывести список владельцев автомобилей марки Volvo и Craisler, выпущенных после 2007 года.

13. Ассортиментный перечень автосалона включает следующие данные: марка машины, модель, год выпуска, цвет, цена базовая, вычисляемые поля - наценка, цена продажи.

Наценка рассчитывается следующим образом: если год выпуска до 2005, наценка составляет 15% от базовой цены, если год выпуска после 2005, наценка составляет 20% от базовой цены.

Цена продажи = цена базовая + наценка.

Рассчитать цену продажи автомобилей и вывести список:

- имеющихся в продаже автомобилей ;
- автомобилей всех моделей заданной марки (например Mazda);
- автомобилей определенного года (или периода) выпуска.

14. Ассортиментный перечень автосалона включает следующие данные: марка машины, модель, год выпуска, цвет, цена базовая, вычисляемые поля – наценка, цена подажи.

Наценка рассчитывается следующим образом:
если год выпуска до 2005, наценка составляет 15% от базовой цены,
если год выпуска после 2005, наценка составляет 20% от базовой цены.
Цена подажи = цена базовая + наценка.

Рассчитать цену продажи автомобилей и вывести список имеющихся в продаже автомобилей. Осуществить поиск автомобиля по заданному критерию (критерий задать самостоятельно).

15. Список сотрудников фирмы включает следующие данные: ФИО, должность, оклад, количество детей, вычисляемое поле – премия. У сотрудников, не имеющих детей, премия составляет 20% от оклада, у сотрудников, имеющих детей, – 30% от оклада.

Рассчитать премию каждого сотрудника фирмы и вывести:
- Список всех сотрудников.
- Список сотрудников, имеющих детей.

Определить общий фонд премии фирмы.

16. Список сотрудников фирмы включает следующие данные: ФИО, должность, оклад, стаж, количество детей, вычисляемые поля – премия, доплата.

У сотрудников, проработавших менее пяти лет,премия составляет 20% от оклада, у сотрудников, проработавших более пяти лет,– 30% от оклада. Кроме того,на каждого ребенка начисляется пособие (доплата), равное 15% от оклада.

Рассчитать премию и доплату каждого сотрудника фирмы и вывести:
- Список всех сотрудников.
- Список сотрудников, не имеющих детей.

Определить фонд премии и фонд доплаты на детей фирмы.

17. Список студентов включает следующие данные: ФИО, год рождения, пол. Найти, сколько в этом списке несовершеннолетних девушек, фамилии которых *оканчиваются на «ич»*, и вывести их список.

18. Список студентов включает следующие данные: ФИО, год рождения, адрес. Найти, сколько в этом списке совершеннолетних юношей, проживающих не в Витебске, и вывести их список.

19. Список студентов группы включает следующие данные: ФИО, год рождения, рост. Вывести список девушек, чей рост ниже среднего по группе и список юношей, чей рост выше среднего по группе.

20. Описать процедуру, которая выводит фамилии двух людей, проживающих в разных городах по одному адресу.

Лабораторная работа № 8

МОДУЛИ. РАЗРАБОТКА БИБЛИОТЕК ПОЛЬЗОВАТЕЛЯ

Цель работы: приобрести практические навыки разработки модульных программ с использованием библиотек пользователя.

Общие сведения

Модуль — *это набор ресурсов (функций, процедур, констант, переменных, типов и т.д.), разрабатываемых и хранимых независимо от использующих их программ.*

Обычно каждый модуль содержит логически связанные между собой программные ресурсы.

- В общем случае модуль имеет следующую структуру:

```
Unit <имя модуля>
interface   <интерфейсная часть>
 uses <список используемых модулей>;
{открытые объявления }
Implementation   <исполняемая часть>
 uses <список используемых модулей>;
{собственные объявления}
{процедуры и функции}
Initialization   <инициализирующая часть>
Finalization   <завершающая часть>
End.
```

- В частном случае модуль может не содержать части реализации и части инициализации, тогда структура модуля будет такой:

```
unit <имя модуля>;        {заголовок модуля}
 interface
 { описание видимых программных элементов модуля }
 implementation
 end.
```

- Для создания модуля следует выбрать команду `File => New => Unit`.
- В случае работы в среде TURBO PASCAL исходный текст модуля должен быть откомпилирован с помощью директивы Make подменю Compile и записан на диск. Результатом компиляции модуля будет файл с расширением .TPU (Turbo Pascal Unit).

- Основное имя модуля берется из заголовка модуля. Для подключения модуля к программе необходимо указать его имя в разделе описания модулей, например: `uses CRT, Graph, Modul;`.

Задания для самостоятельного выполнения

1. Создать модуль `Ar` и отладить программу `Test` (см. пример лекции 8).

2. Создать модуль `TRG`, содержащий функции вычисления тригонометрических и обратных тригонометрических функций:

tg(x) =sin(x)/cos(x),

ctg(x)=cos(x)/sin(x),

$$Arc\sin(x) = Arctg(\sqrt{x/(1-x^2)})$$

$$Arc\cos(x) = \pi/2 - Arc\sin(x)$$

$$Arcctg(x) = \pi/2 - Arctg(x)$$

3. Создать модуль `LOGR`, содержащий функции вычисления логарифма числа по основанию $\log_a(x)$ и десятичного логарифма `lg(x)`.

4. Составить программу `FUNC`, содержащую модули `TRG` и `LOGR`, для вычисления тригонометрических и обратных тригонометрических функций, натурального, десятичного логарифма и логарифма числа по заданному основанию для некоторого набора чисел.

5. Создать модуль `MAS_1`, содержащий функции, реализующие следующие операции с одномерными массивами:

- ввод элементов одномерного массива;
- вывод элементов одномерного массива.

6. Создать модуль `MAS_2`, содержащий функции, реализующие следующие операции с одномерными массивами:

- вычисление суммы (произведения) всех элементов массива;
- вычисление суммы (произведения) квадратов элементов массива;
- вычисление количества положительных (отрицательных) элементов массива.

7. Создать модуль `MAS_3`, содержащий функции, реализующие основные операции с одномерными массивами:

- поиск максимального элемента массива;
- поиск минимального элемента массива;
- вычисление значения среднего арифметического элементов массива;
- вычисление значения среднего геометрического элементов массива;

8. Составить программу `MASSIV` для обработки одномерного массива с использованием алгоритмов, реализованных в модуле `MAS_1`, `MAS_2` и `MAS_3`.

9. Создать модуль MATR_1, содержащий функции, реализующие следующие операции с матрицами:

- ввод элементов матрицы;
- вывод элементов матрицы;
- вычисление суммы (произведения) элементов матрицы.

10. Создать модуль MATR_2, содержащий функции, реализующие следующие операции с матрицами:

- поиск максимального элемента матрицы;
- поиск минимального элемента матрицы;
- вычисление следа матрицы;
- вычисление суммы элементов, стоящих над главной диагональю матрицы;
- вычисление суммы элементов, стоящих под главной диагональю матрицы.

11. Создать модуль MATR_3, содержащий функции, реализующие следующие операции с матрицами:

- сложение матриц;
- вычитание матриц;
- умножение матриц;
- умножение матрицы на вектор.

12. Составить программу MATRICA для обработки матриц с использованием алгоритмов, реализованных в модуле MATR_1, MATR _2 и MATR _3.

13. Создать модуль ST_F, содержащий функции:

- вычисления факториала числа x;
- возведения целого положительного числа в любую степень;
- возведения натурального числа в натуральную степень.

14. Составить программу CHISLA для вычисления:

- факториала некоторого целого числа X;
- вычисления n-ой степени некоторого натурального числа y.

Использовать модуль ST_F.

15. Составить программу, использующую модуль INT, который содержит процедуру вычисления значения интеграла с помощью одного из известных методов (трапеции, Симпсона, см. Приложение к Лабораторной работе № 4, задание 11).

ВОПРОСЫ К ЭКЗАМЕНУ

1. Классификация персональных компьютеров (ПК).
2. Состав и структура персонального компьютера (ПК).
3. Принцип программного управления ПК.
4. Определение алгоритма. Свойства алгоритмов.
5. Способы описания алгоритмов.
6. Основные алгоритмические структуры.
7. Основные этапы технологического процесса разработки программ.
8. Характеристика и назначение структурного программирования.
9. Методы преобразования неструктурированных программ в структурированные.
10. Этапы подготовки и решения задачи с использованием ПК.
11. Этапы создания и обработки программного файла в системе консольного приложения Delphi.
12. Язык программирования Паскаль. Алфавит языка. Идентификаторы.
13. Лексемы языка Паскаль.
14. Константы и переменные языка Паскаль.
15. Основные типы данных языка Паскаль. Стандартные скалярные типы.
16. Перечисляемый и интервальный типы данных языка Паскаль.
17. Операции и выражения в языке Паскаль.
18. Стандартные функции и правила записи арифметических выражений в языке Паскаль.
19. Функции выделения целой и дробной части аргумента. Определение четности аргумента.
20. Классификация операторов языка. Простые операторы. Оператор присваивания.
21. Операторы ввода-вывода в языке Паскаль. Форматный вывод данных.
22. Оператор безусловного перехода в языке Паскаль.
23. Структура программы на языке Паскаль. Разделы описания меток, констант и переменных.
24. Структура программы на языке Паскаль. Раздел описания типов пользователя.
25. Структура программы на языке Паскаль. Раздел операторов.
26. Структурированные операторы языка Паскаль. Оператор условного перехода.
27. Оператор выбора языка Паскаль.
28. Операторы цикла. Оператор цикла с параметром.
29. Операторы цикла. Оператор цикла с предусловием.
30. Операторы цикла. Оператор цикла с постусловием.
31. Классификация подпрограмм языка Паскаль.
32. Локализация имен в подпрограммах пользователя. Формальные и фактические параметры.

33. Подпрограммы-процедуры пользователя.
34. Подпрограммы-функции пользователя.
35. Рекурсивные подпрограммы.
36. Структурированные типы данных. Понятие массива. Способы индексации элементов массива.
37. Способы описания одномерных и многомерных массивов в языке Паскаль.
38. Операции над одномерными массивами. Ввод-вывод элементов одномерного массива. Вычисление суммы (произведения) элементов одномерного массива.
39. Поиск максимального (минимального) элемента в одномерном массиве. Перестановка элементов одномерного массива.
40. Ввод-вывод элементов двумерного массива.
41. Вычисление суммы (произведения) элементов двумерного массива.
42. Вычисление суммы (произведения) элементов строки (столбца) двумерного массива.
43. Поиск максимального (минимального) элемента в двумерном массиве.
44. Поиск максимального (минимального) элемента строки (столбца) в двумерном массиве.
45. Сортировка массивов.
46. Строковый тип данных.
47. Функции обработки строк.
48. Понятие записи. Записи без вариантов.
49. Понятие записи. Записи с вариантами.
50. Операции над записями.

ОБРАЗЕЦ ВАРИАНТА ЗАДАНИЯ

ДЛЯ ТЕСТОВОГО КОНТРОЛЯ ЗНАНИЙ

по курсу

«ОСНОВЫ АЛГОРИТМИЗАЦИИ и ПРОГРАММИРОВАНИЯ»

(количество вопросов в тесте 30 -35)

1. Алгоритм может быть описан следующим образом...
в виде текста
в виде музыки
в виде блок-схемы
в виде последовательности математических формул

2. Какой из нижеперечисленных терминов обозначает свойство алгоритмов, согласно которому алгоритм служит не для решения какой-то одной конкретной задачи, а для решения целого класса однотипных задач?
массовость
определенность
понятность
конечность

3. Диапазон данных, описываемый типом *byte* языка Паскаль, составляет…
0..65535
-128..127
0..255
-32768..32767

4. К встроенным процедурам преобразования типов языка Паскаль относятся…
Pred(S)
Round(R)
Trunc(R)
Succ(S)

5. Из перечисленных идентификаторов но правилам языка Паскаль записаны...
Byte

Gr_Mn18
2_Mn18
Var_A

6. Для выделения целой части числа в языке Паскаль используют функции...
Int(x)
Frac(x)
Trunc(x)
Round(x)

7. Какие скобки не используются в языке программирования Паскаль для обозначения комментариев?
{ }
/* */
(* *)
[ ]

8. Какой из перечисленных типов языка программирования Паскаль не является порядковым?
Word
Real
Byte
Char

9. Функция $a^{\frac{1}{3}}$ на языке программирования Паскаль записывается в виде…
exp(1/3*ln(x))
a^1/3
exp(a *ln(1/3))
exp(1/3*ln(a))

10. Функция Round(24,31) возвратит результат...
24,3
25
20
24

11. Из нижепредставленных утверждений верными являются...

Int(85.7)=85
Int(85.7)=86
Round(85.7)=86
Round(85.7)=85

12. Из нижепредставленных утверждений верными являются ...
10 div 3 =2
5 mod 2 =1
15 div 2 =7
11 mod 3 =3

13. Какой из операторов языка Паскаль имеет наименьший приоритет выполнения?
Not
and
or
=

14. Для вывода значений Y вещественного типа следует использовать оператор вывода языка программирования Паскаль в формате...
Writeln(Y:p)
Writeln(Y:p:q)
Write(Y:p:q)
Write(Y:p)

15. Чему равно значение y, если x=9? Program test; Var x,y:integer; Begin Read(x); If not ODD(x) then y:=x Else y:=x2; Writeln(y); End.
X
x^2
9
81

16. Рекуррентное соотношение для вычисления элементов ряда

$$S = \sum_{n=1}^{6} \frac{(x+1)^{2n}}{(n+2)!}$$

имеет вид:

$U_0=0,5;\ U_n = U_{n-1}\cdot(x+1)^2/(n+2)$

$U_0=1;\ \ U_n = U_{n-1}\cdot(x+1)/n+2$

$U_0=0,5;\ U_n = U_{n-1}\cdot(x+1)/(n+2)$

$U_0=x;\ \ U_n = U_{n-1}\cdot(x+1)^2/(n+2)$

17. Каким будет результат работы фрагмента программы?

```
var
 a,b,c,d:integer;
begin
 a:=1;
 b:=2;
 c:=3;

 d:=abs(a-c*b)*c;

 Writeln('d=',d);
 Readln;
end.
```

d=10

d=16

d=15

d=18

18. Каким будет результат работы фрагмента программы?

```
var
 a,b,c,d:integer;
begin
 a:=1;
 b:=2;
 c:=3;
 if ((a=a)or(a<>c)) then
                 begin
                  b:=a+c;
                  a:=2*a+b;
                 end;
  d:=a+b+c;
 Writeln('d=',d);
 Readln;
```

```
    end.
```

d=5

d=13

d=7

d=9

19. Выражение $y = \sqrt{e^x} \cdot \ln(x^2)$ на языке программирования Паскаль записывается в виде…

y:=sqrt(exp^x)* ln(x)

y:=sqrt(exp(x))*ln(sqr(x))

y:=sqrt(exp(x)* sqr(ln(x)))

y:=sqr(exp(x))*ln(x)

20. Каким будет результат работы фрагмента программы?

```
var
  k,i:integer;
begin
  k:=0;

  for i:=1  to 10 do;
    k:=k+1;

  Writeln('k=',k);
  Readln;
end.
```

k=0

k=1

k=11

k=10

21. В подпрограмме-процедуре Sum вычисляется сумма значений a и b. Заголовок такой процедуры, составленной на языке программирования Паскаль, может иметь вид:

```
Procedure Sum(a,b:integer; Var s:integer);
Procedure Sum(a,b:real; Var s:integer);
Procedure Sum(a:real;b:integer; Var s:integer);
Procedure Sum(a:real;b:integer; Var s:real);
```

22. В каком месте Паскаль-программы не может располагаться описание подпрограммы пользователя (Procedure ..., Function...)?

После раздела операторов (Begin ... End)

Перед разделом описания типов (Type ...)

После раздела описания типов (Type ...)
Перед разделом описания переменных (Var ...)

23. В подпрограмме-процедуре Sum вычисляется сумма значений a и b. Заголовок такой процедуры, составленной на языке программирования Паскаль, может иметь вид:

```
Procedure Sum(a,b:integer; Var s:integer);
Procedure Sum(a,b:real; Var s:integer);
Procedure Sum(a:real;b:integer; Var s:integer);
Procedure Sum(a:real;b:integer; Var s:real);
```

24. Подпрограмма-функция SR, вычисляющая среднее арифметическое трех целых чисел x,y,z (и обращение к ней из программы), на языке программирования Паскаль должна быть описаны следующим образом :

```
Function SR(x,y,z:real): real;
Begin S:=(x+y+z)/3; End;
SR(a,b,c,SRED);
```

```
Function SR(x,y,z:integer): real;
Begin SR:=(x+y+z)/3; End;
Sred:=SR(a,b,c);
```

```
Function SR(x,y,z:real):real;
Var S:real; Begin S:=(x+y+z)/3;SR:=S;End;
SR(a,b,c,S);
```

```
Function SR(x,y,z:integer):integer;
Begin SR:=(x+y+z)/3;End;
Sred:=SR(a,b,c);
```

25. Формат записи заголовка подпрограммы-процедуры с параметрами в языке программирования Паскаль имеет вид:

```
Procedure <имя процедуры>{(формальные параметры)};
Procedure <имя процедуры>{(фактические параметры)};
Procedure <имя процедуры>{(локальные параметры)};
Procedure <имя процедуры>{(формальные параметры)}:<тип>;
```

26. Формат описания одномерного массива на языке программирования Паскаль имеет вид...

```
Type <имя типа>:array[тип индекса] of <тип компонент>;
    Var <имя типа> : <тип компонент>;
```

```
Type <имя типа>=array[тип индекса] of <тип компонент>
    Var <идентификатор>:<имя типа>;
```

```
Var <идентификатор>: array of <тип компонент>;
```

```
Var <идентификатор>: array[тип индекса] of <тип компонент>;
```

27. Матрица A (3 X 4) из элементов вещественного типа на языке программирования Паскаль может быть описана следующим образом...

Var A : array [1..4,1..3] of real;

Var A : array [1..3,1..4] of real;

```
Type mas = array[1..3,1..4]  of real;
  Var a:mas;
```

```
Type mas = array [1..4,1..3]  of real;
  Var mas:A;
```

28. Вывод элементов одномерного массива A (i =1..n) в строку на языке программирования Паскаль может быть выполнен следующим образом...

For i:=1 to n do writeln(a[i])

For i =1 to n do write(a[i])

For i:=1 to n do write(a[i])

For i:=1 to n do read(a[i])

29. Каким будет результат работы фрагмента программы?

```
k:=1;
for i:=2 to 10 do
if A[i]<A[k] then k:=i;
Writeln(k);
```

индекс минимального элемента массива

значение максимального элемента массива

значение минимального элемента массива

индекс максимального элемента массива

1

30. Каким будет результат работы фрагмента программы?

```
var
 a:array[1..10] of integer;  i,s:integer;
begin
for i := 1 to 10 do
 a[i]:=i;  s:=0;
for i := 1 to 4 do
  s:=s+a[i];
Writeln('s=',s);
Readln;
end.
```

s=5

s=4

s=10

s=9

ЛИТЕРАТУРА

1. *Абрамов, В.* Введение в язык Паскаль / В. Г. Абрамов, Н. П. Трифонов, Г. Н. Трифонова. – Москва : Наука, 1988. – 320 с.
2. *Алексеев, А. П.* Информатика 2007 : учебное пособие для студентов вузов, обучающихся по направлению подготовки дипломированных специалистов 210400 (654400) –Телекоммуникации / А. П. Алексеев. – Москва : Солон-Пресс, 2007. – 608 с.
3. *Бобровский, С. И.* Delphi 7. Учебный курс / С. И. Бобровский. – Санкт-Петербург : Питер, 2008. – 736 с.
4. *Бородич, Ю. С.* Паскаль для персональных компьютеров / Ю. С. Бородич [и др.]. – Минск : Высшая школа, 1991. – 336 с.
5. *Вальвачев, А. Н.* Программирование на языке Паскаль для персональных ЭВМ ЕС /А. Н. Вальвачев, В. С. Крисевич. – Минск : Вышейшая школа, 1991. – 224 с.
6. *Вардомацкая, Е. Ю.* Основы информатики и вычислительной техники : курс лекций / Е. Ю. Вардомацкая. – Витебск, 2005. – 130 с.
7. *Вардомацкая, Е. Ю.* Информатика в прикладных задачах легкой промышленности : пособие / Е. Ю. Вардомацкая, Т. Н. Окишева. – Витебск, 2007. – 187 с.
8. *Вардомацкая, Е. Ю.* Информатика : учебное пособие. В двух частях. Часть I. / Е. Ю. Вардомацкая, Т. Н. Окишева. – Витебск, 2007. – 237 с.
9. *Велихов, А. В.* Основы информатики и компьютерной техники : учебное пособие для студентов ссузов и вузов по дисциплине "Основы информатики" / А. В. Велихов. – Москва : СОЛОН-ПРЕСС, 2007. – 544 с.
10. *Епанешников, А.* Программирование в среде Turbo Pascal 7.0 / А. Епанешников, В. Епанешников. – Москва : Диалог – МИФИ, 2000. – 256 с.
11. *Морозевич, А. Н.* Прикладная информатика : учебное пособие / под общ. ред. А. Н. Морозевича. – Мн.: Выш. школа, 2003. – 335 с.: ил.
12. *Офицеров, Д. В.* Программирование на персональных ЭВМ : практикум : учебное пособие / Д. В. Офицеров, А. Б. Долгий, В. А. Старых ; под общ. ред. Д. В. Офицерова. – Минск : Высшая школа, 1993. – 256 с.
13. *Попов, В. Б.* Паскаль и Дельфи : учебный курс / В. Б. Попов. – Санкт-Петербург : Питер, 2005. – 576 с.
14. *Прищепов, М. А.* Программирование на языках Basic, Pascal и Object Pascal в среде Delphi : учебное пособие / М. А. Прищепов, Е. В. Севернева, А. И. Шакирин ; под общ. ред. М. А. Прищепова. – Минск : ТетраСистемс, 2006. – 320 с.
15. *Фаронов, В. В.* Delphi. Программирование на языке высокого уровня : учебник для вузов / В. В. Фаронов. – Санкт-Петербург : Питер, 2008. – 640 с.

Учебное издание

Вардомацкая Елена Юрьевна

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Учебно-методический комплекс

Редактор *В. П. Терентьев*
Технический редактор *И.В. Соколов*
Корректор *Е. М. Богачева*
Компьютерная верстка *Е. Ю. Вардомацкая*

Подписано к печати 5.11.10. Формат 60x84 1/16. Бумага офсетная №1.
Гарнитура «Таймс». Усл. печ. листов 6.06. Уч.-изд. листов 5.9.
Тираж *48* экз. Заказ № 432

Учреждение образования «Витебский государственный технологический университет» 210035, Витебск, Московский пр-т, 72

Отпечатано на ризографе учреждения образования «Витебский государственный технологический университет».
Лицензия №02330/0494384 от 16.03.2009 г.