Бураков П.В., Косовцева Т.Р. Информатика. Алгоритмы и программирование. Учебное пособие.-СПб НИУ ИТМО, 2013. – 83с.

Учебное пособие предназначено для студентов экономических специальностей специальности 080100 «Экономика» гуманитарного факультета, изучающих дисциплину «Информатика». Пособие представляет собой вводный курс по построению алгоритмов и по программированию на языке Турбо Паскаль. Пособие содержит много примеров. Подробно излагаются теоретические аспекты построения алгоритмов и основ программирования. Во второй части учебного пособия приводтся лабораторный практикум.

Рекомендовано к печати на заседании Ученого совета Гуманитарного факультета, протокол № 8 от 15 октября 2013 г.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

СОДЕРЖАНИЕ

ЭЛЕМЕНТЫ ТЕХНОЛОГИИ РЕШЕНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ НА КОМПЬЮТЕРЕ

Решение задач с применением персонального компьютера включает следующие основные этапы.

1. Постановка задачи.
2. Разработка математической модели.
3. Выбор метода численного решения для расчетных задач.
4. Разработка алгоритма решения и структуры данных.
5. Реализация алгоритма в виде программы.
6. Отладка и тестирование программы.
7. Решение задачи на компьютере, численные эксперименты и анализ результатов.

Постановка задачи

В работе специалиста, проектирующего новое изделие, технологический процесс или решающего определенную практическую задачу для нужд пользователя, сочетаются наука и искусство.

Конечным итогом является нахождение подходящего решения и представление его в виде удобном для практического применения. Эффективность решения во многом определяется четким осознанием потребности и точной словесной формулировкой этой потребности в виде цели (или нескольких целей). После этого в исследуемой предметной области собирается информация о тех факторах, которые влияют на достижение конечной цели, о взаимосвязи этих факторов, об ограничениях, связанных с различными сторонами рассматриваемой задачи. Таким образом, создается возможно более полное словесное описание задачи и условий ее решения.

Пример 1

Требуется определить длину пути, который проходит тело, движущееся с переменной скоростью. Начальная скорость тела равна 0. С определенной степенью степенью достоверности движение можно считать равноускоренным. Использование пакетов стандартных программ не предполагается.

В настоящее время эффективное решение сложных задач базируется на использовании математических моделей. Кроме возможности получения решения целого класса задач применение математических моделей обеспечивает проведение многовариантных расчетов, быструю адаптацию к изменению условий и поиск наилучших решений.

Разработка математической модели

На основе словесной формулировки задачи выбираются переменные, участвующие в процессе решения, определяется их область изменения, математически описываются связи между этими переменными и ограничения. Выделяются и формально описываются те факторы, которые входят в эти взаимосвязи как параметры. В результате инженерная задача приобретает вид формализованной математической модели.

Пример 2

Математической моделью задачи, сформулированной в примере 1 является формула: $S = 0{,}5 \cdot a \cdot t^2$, где в качестве основной переменной выступает время **t**, параметром движения является ускорение ***a***. Формула показывает зависимость между длиной пути и временем в соответствии с законом механики.

Следует отметить, что для математического описания зависимостей можно использовать различную степень детализации и различные формы математического описания. Так, в общем случае при движении с переменной скоростью длина пути определяется интегралом

$$S = \int_0^t V(t)dt ,$$

при этом закон изменения скорости V(t) в зависимости от времени может быть достаточно сложным и определить значение интеграла аналитически не удается. Поэтому в большинстве случаев практически целесообразно использовать приближенные методы решения.

Особое внимание следует уделять адекватности математической модели по отношению к исходному описанию задачи и условиям ее решения с одной стороны. Неоправданная сложность приводит к ненужным затратам сил, средств и времени. С другой стороны, недостаточная точность математического описания (грубость модели) приводит к большим погрешностям в решении, а иногда и к ошибочным выводам.

Выбор метода численного решения

Для расчетной задачи необходимо выбрать метод ее численного решения, сводящий решение задачи к последовательности арифметических и логических операций. Разработкой и изучением таких методов занимается раздел математики, называемый численным анализом. В качестве примера численного метода для вычисления определенных интегралов можно привести метод прямоугольников. В этом методе интеграл заменяется конечной суммой

$\int_a^b f(x)dx \approx \Delta x \sum_{\forall i} f(x_i)$, где Δx - шаг интегрирования (const), Δx=(b-a)/n,

$x_{i+1}=x_i+\Delta x$, $x_1=a$, $x_{n+1}=b$

Вычисление приближенного значения определенного интеграла по этому методу сводится к последовательному расчету значений подынтегральной функции, их суммированию и умножению на величину шага интегрирования.

При выборе метода надо учитывать требования, предъявляемые постановкой задачи, и возможности его реализации на компьютере: точность решения, быстроту получения результата, требуемые затраты оперативной памяти для хранения исходных и промежуточных данных и результатов (для задач большого объема), сложность программной реализации, трудоемкость подготовки программы и решения задачи.

Разработка алгоритма и структуры данных

Если выбранный для решения задачи численный метод реализован в виде стандартной библиотечной подпрограммы, то алгоритм обычно сводится к описанию и вводу исходных данных, вызову стандартной подпрограммы и выводу результатов на экран или на печать. Более характерен случай, когда стандартные подпрограммы решают лишь какую-то часть задачи.

Четкая структуризация задачи, разбиение ее на последовательность подзадач, реализация подзадач отдельными модулями, постепенная детализация логики алгоритма, использование типовых логических конструкций являются хорошими средствами, позволяющими в разумные сроки создавать работоспособные программы для решения сложных задач.

Важной составной частью разработки алгоритма является выбор состава и способов организации (структур) данных: исходных, промежуточных и конечных результатов. Языки программирования, позволяют работать с числовыми и символьными константами, переменными, массивами данных (векторами и матрицами). Так в языке Turbo Pascal допускаются и более сложные структуры данных, удобные для задач обработки нечисловых данных, например текстов, для решения комбинаторных задач и имитационного моделирования. Поэтому выбор состава, типов и структур данных обычно производится с учетом особенностей реализации алгоритма на том или ином входном языке. Если разработчик программы владеет программированием на разных алгоритмических языках, то у него появляется возможность выбрать язык, наиболее соответствующий структуре данных задачи.

Разработка алгоритма завершается либо его представлением в виде графической схемы, либо записью с помощью символов специального языка проектирования программ, называемого псевдокодом. Используются также другие средства представления логики алгоритма: диаграммы, таблицы решений, схемы.

Реализация алгоритма в виде программы

При составлении программы на выбранном языке следует познакомиться с описанием особенностей реализации этого языка. Сначала рекомендуется использовать основные конструкции и типы данных языка, расширяя диапазон применяемых средств по мере приобретения опыта программирования и практического решения задач на компьютере.

При разработке и реализации алгоритмов не следует стремиться к получению программ, наилучших по быстродействию, объему требуемой памяти, удобству работы с данными. Дополнительное время, затраченное на разработку, программирование и отладку слишком сложного алгоритма, может обесценить результаты от его применения, так как они будут получены слишком поздно.

Важно получить работоспособную программу как можно раньше, жертвуя при этом на первых порах изяществом, компактностью, а порой и вычислительной эффективностью алгоритма. Совершенствование программы можно отложить до более позднего времени, когда проведенные эксперименты позволят уточнить постановку задачи, выяснить сферу применимости

программы, требования удобства ввода исходных данных и обработки результатов. С этой точки зрения также полезно использовать принцип модульности при разработке алгоритма, так как программу, составленную из модулей, легче модернизировать. Выделение в программе модулей ввода данных, расчета характеристик, вывода результатов позволит локализовать будущие изменения в программе в пределах каждого модуля, а также использовать некоторые из них в качестве готовых «строительных» блоков для новых задач.

Отладка и тестирование программы

При программировании и вводе данных с клавиатуры могут быть допущены ошибки. Их обнаружение, локализацию и устранение выполняют на этапе отладки и испытания (тестирования) программы.

Одним из критериев профессионального мастерства программистов является их способность обнаруживать и исправлять собственные ошибки: начинающие программисты не умеют этого делать, у опытных программистов особых затруднений это не вызывает. Тем не менее ошибки в программах делают все.

По данным разных авторов, этап отладки и тестирования программы занимает от 50 до 70 % времени, затрачиваемого на все этапы создания программы и получения решения. В связи с важностью и трудоемкостью этапа отладки все современные системы программирования имеют специальные средства, помогающие в обнаружении и устранении ошибок. Уже на этапе разработки алгоритма полезно предусмотреть простейшие средства контроля, встраиваемые в разрабатываемую программу: печать введенных данных непосредственно после их считывания в память машины (эхо-печать), печать промежуточных результатов в узловых точках. Но главное — надо максимально упрощать структуру программы за счет расчленения ее на модули, реализуемые в виде подпрограмм или процедур, и использования конструкций языка, наиболее простых и освоенных программистом.

Решение задачи на компьютере

На этом этапе компьютер выполняет все предусмотренные программой вычисления и выдает результаты на экран или на печать. Чтобы облегчить последующую обработку результатов счета, надо при проектировании программы предусмотреть вывод результатов с пояснениями. Выводимым числам и таблицам результатов должны предшествовать заголовки, одни группы данных следует отделять от других пропусками строк. При табулировании функции полезно указывать и значения аргументов, а если функции зависят от параметров, то перед выдачей значений функции следует указать и значения параметров.

При недостаточном опыте программирования не следует увлекаться формой представления результатов, однако какое-то время все же следует уделять этим вопросам, так как в конечном итоге это сокращает трудоемкость последующей обработки данных. Целесообразно отрабатывать форму представления выводимых данных на пробных прогонах программы, анализируя результаты на экране.

Контрольные вопросы

1. Какие факторы влияют на эффективность решения задач на компьютере?
2. Перечислите этапы решения задач на компьютере.
3. Каковы основные требования к математической модели задачи?
4. Назовите главные характеристики алгоритмов.
5. Какие особенности следует учитывать при разработке программ на компьютере?
6. Чем завершается разработка алгоритма?

ПОСТРОЕНИЕ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ

Описание алгоритма

Под *алгоритмом* понимается конечная последовательность точно сформулированных правил решения некоторого касса задач. Алгоритм обладает следующими свойствами:

Определенность. Все действия, которые необходимо произвести должны быть строго определены.

Понятность. Все действия, которые необходимо произвести, должны быть однозначно поняты и выполнены каждым исполнителем алгоритма. Это свойство может быть интерпретировано как однозначность алгоритма, под которой понимается единственность толкования исполнителем правил выполнения действий и порядка их выполнения.

Конечность. Обязательность завершения каждого из действий, составляющих алгоритм и завершимость выполнения алгоритма в целом.

Результативность. Если алгоритм применим к данной задаче, то после конечного числа шагов должен быть получен результат.

Массовость. Возможность применения данного алгоритма для решения класса задач, отвечающих общей постановке задачи.

Правильность. Способность алгоритма давать правильные результаты решения поставленных задач.

Каждое правило алгоритма записывается в виде повелительного предложения, понимаемого исполнителем алгоритма как команда на выполнение.

Рассмотрим алгоритмы решения нескольких задач.

Задача 1. Составить алгоритм вычисления x по формуле

$x = \frac{a(r-q)^2}{p+12}$, где p ≠ -12.

1. Начало;
2. Вычислить b := p + 12;
3. Вычислить c := r – q;
4. Вычислить d := c · c;
5. Вычислить e := a · d;
6. Вычислить x := e / b;
7. Записать результат: x;
8. Конец.

В записи алгоритма решения **задачи 1** введены буквенные обозначения – переменные. Так, через *b* обозначено *p + 12*, через *c* – разность *r – q*. Запись

b:= p+12 означает, что вначале должна быть найдена сумма *p + 12* при определенных значениях p, а затем это значение присваивается переменной b.

Придавая *a, p, q, r* разные значения, будем получать различные задания на вычисление *x*. Поэтому алгоритм обычно позволяет решать не одну, а целый класс задач. Такую особенность алгоритма называют массовостью алгоритма. Возможны алгоритмы, решающие только единственную задачу.

Величины *a, p, q, r, 12* составляют исходные данные для алгоритма, 12 – постоянную составляющую, a, p, q, r – переменную составляющую информации. Величины *b, c, d, e* являются вспомогательными переменными, *x* – результат исполнения алгоритма.

Алгоритм называется *линейным*, если при его исполнении все команды выполняются строго последовательно, без нарушения порядка их следования. Предложенный алгоритм решения **задачи 1** является линейным.

Следует отметить, что сама формула не является алгоритмом, так как не указан однозначный порядок исполнения операций.

Задача 2. По формуле $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ составить алгоритм решения квадратного уравнения $ax^2 + bx + c = 0$ ($a \neq 0$).

1. Начало;
2. p := 2a;
3. D := b^2 – 4ac;
4. Если D ≥ 0, то перейти к п. 5, иначе к п. 10;
5. d := $\sqrt{D}$;
6. $x_1 := \frac{-b-d}{p}$;
7. $x_2 := \frac{-b+d}{p}$;
8. Записать результат: x_1 , x_2;
9. Перейти к п. 11;
10. Записать результат: уравнение действительных корней не имеет;
11. Конец.

При исполнении алгоритма выполняются не все его команды, следовательно, алгоритм решения **задачи 2** не является линейным. Кроме арифметических действий и операций извлечения квадратного корня, вычислительный процесс содержит в правиле 4 операцию проверки условия: в зависимости от выполнения условия (да) или невыполнения условия (нет) выбирается одно из двух возможных продолжений. Такой алгоритм называют разветвляющимся. Операцию, которая может изменить последовательность выполнения правил алгоритма, назовем оператором управления.

Команды, определяющие порядок исполнения операций, подразделяются на два типа: команды *условного перехода* и команды *безусловного перехода*. К командам условного перехода относится, например, правило 4 алгоритма решения **задачи 2**. Правило 9 содержит безусловный переход.

В командах условного перехода обязательно присутствует логическое условие типа: если $\alpha \lozenge \beta$, то …(где $\lozenge$ - один из операторов $>, \geq, <, \leq, =, \neq$).

Схема алгоритма

Схема алгоритма представляет собой графическое изображение алгоритма с помощью определенным образом связанных между собой блоков. Каждый блок изображается определенной фигурой. Блоки в схеме соединяются дугами. Дуги указывают порядок следования блоков при исполнении алгоритма. Типы блоков представлены на рисунке 1.

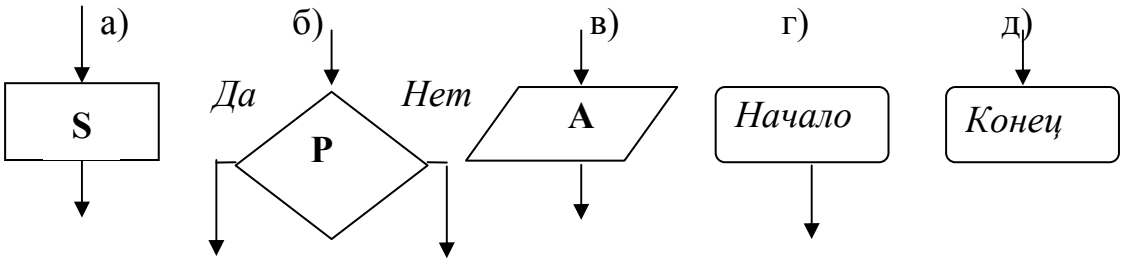


Рис.1. Типы блоков

Прямоугольник вместе с заключенным в нем этапом вычисления *S* называют *функциональным блоком*, или *процессом* (рис.1, а). Ромб с заключенным в нем условием *Р* называют *блоком проверки условия*, или *решением* (рис.1, б). Он используется для управления порядком исполнения блоков в схеме алгоритма. Из функционального блока выходит одна стрелка, а из блока проверки условия – две. Последнее объясняется тем, что в результате исполнения команды проверки условия получаем либо выполнение (да), либо невыполнение (нет) заданного условия Р. *Информационный блок* (рис.1, в) используется для ввода и вывода А. Блоки (рис.1, г) и (рис.1, д) называют соответственно *начальным* и *конечным*. Блок-схема решения задачи начинается в начальном блоке и заканчивается в конечном.

Исполнение алгоритма решения задачи осуществляется по схеме в направлении, заданном дугами от начального блока к конечному. В информационных блоках указываются значения параметров, которые задаются исполнителю, и значения переменных, которые вводятся. В функциональных и логических блоках записаны операторы от переменных.

Исполнение функциональных блоков осуществляется согласно записанным в них действиям. В логических блоках в результате анализа происходит выбор одной из двух возможных дуг «да» или «нет» и передача управления блоку, на который указывает выбранная дуга. Поэтому при исполнении алгоритма существует всего один путь от начального блока к конечному.

При составлении схемы алгоритмов решения задач подразумевается, что значения переменных в операциях, записанных в блоках, уже заданы. Однако для исполнения алгоритмов необходимы специальные указания на то, откуда брать значения этих переменных.

На рисунках 2 и 3 изображены схемы алгоритмов решения **задачи** нахождения суммы *n*- чисел: a_1, a_2, a_3, a_4,.., a_n.

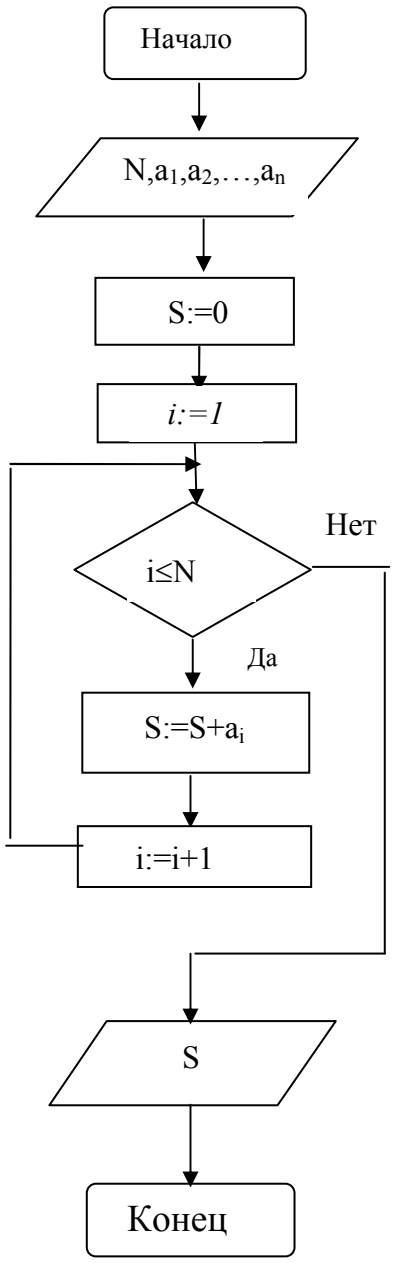


Рис. 2.

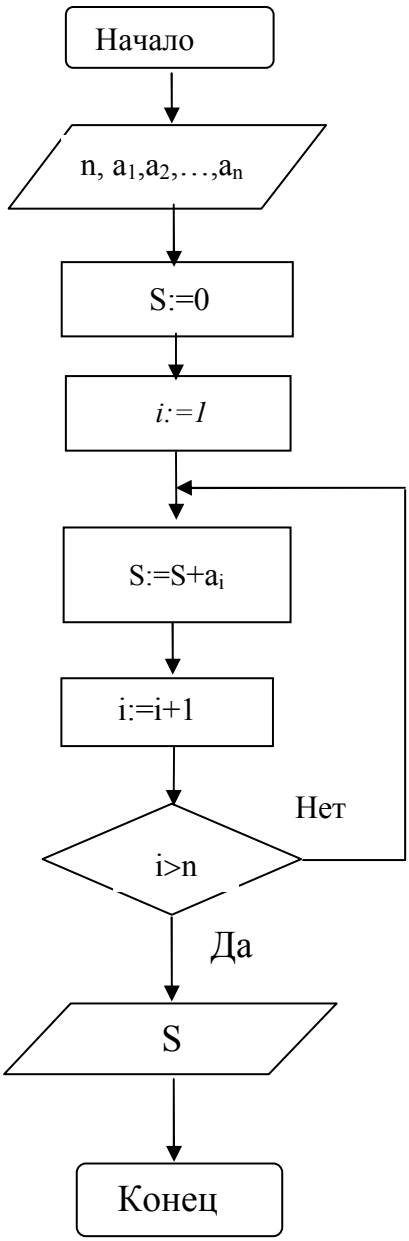


Рис. 3.

Структурированные схемы алгоритмов

В схемах алгоритмов можно усмотреть различные типы следования блоков. Однако для дальнейшего изложения достаточно выделить следующие типы, названные базовыми структурами: *следование, развилка, повторение*.

Структура *следование* (рис.4) означает, что два действия S_1 и S_2 должны быть выполнены последовательно одно за другим, т.е. управление передается от предыдущего функционального блока к следующему.

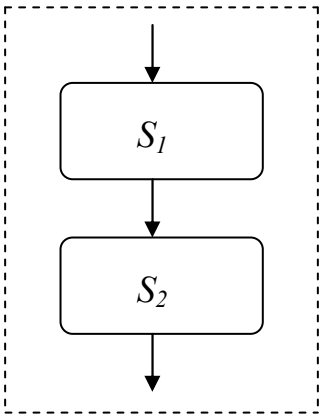


Рис. 4. Структура «**следование**»

Структура *развилка* (рис.5а и 5б) начинается блоком проверки условия и заканчивается пересечением дуг. Структурой развилки осуществляется анализ условия P и выбор одной из двух альтернатив дальнейшего пути выполнения действий.

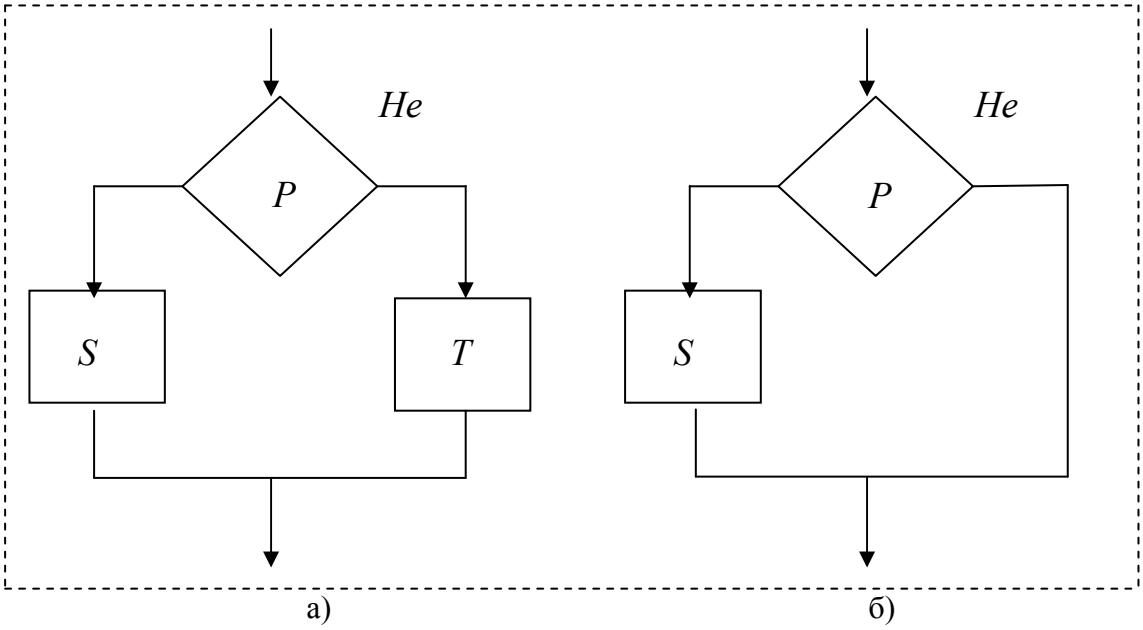


Рис. 5. Структура «**развилка**»

Различают *полную* развилку (рис.5а) и *неполную* развилку (рис.5б). Словесно полная развилка описывается так:

Если *P* истинно, то выполнить *S*, иначе выполнить *T*.

Или кратко: Если *P* то *S* иначе *T*.

Следует отметить, что сначала вычисляется значение истинности условия *P*. В случае истинности исполняется действие *S*, а в случае ложности – действие *T*. Управление передается от блока проверки условия к функциональному блоку, затем осуществляется выход из этой структуры.

Неполная развилка описывается так:

Если *P* истинно, то выполнить *S*.

Или кратко: Если *P* то *S*.

При исполнении этой структуры сначала вычисляется значение истинности условия *P*. В случае истинности управление передается от блока проверки условия функциональному блоку и исполняется действие *S*. В случае ложности управление покидает структуру.

Структура *повторение* (рис.6) описывает циклические процессы.

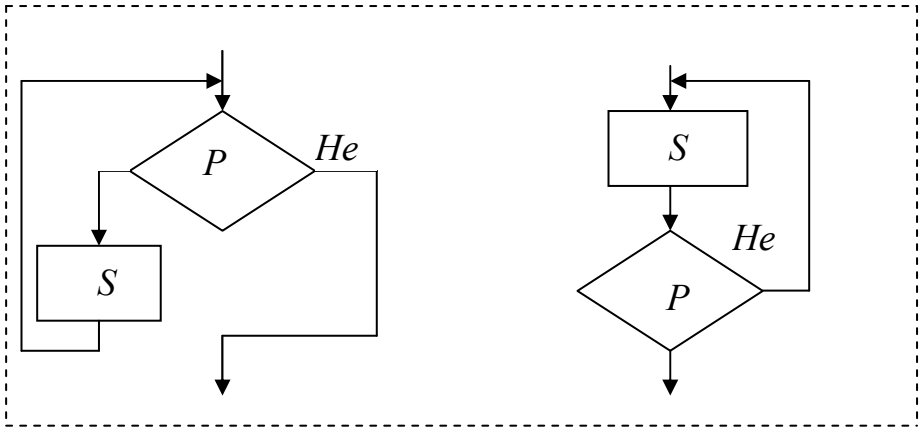


Рис. 6. Структура «**повторение**»

Структуру, изображенную на рис.6а, называют *цикл–пока.* Словесно ее можно выразить так:

Пока *P* истинно, выполнять *S*.

Или кратко: Пока *P*, выполнять *S*.

Истинность условия *P* вычисляется до исполнения действия *S*, а так как анализируемое условие может сразу оказаться ложным, то действие *S* может вообще не исполняться.

Структуру, изображенную на рис.6б, называют *цикл–до*. Словесно она выражается так:

Выполнять *S* до истинности *P*.

Истинность условия *P* в структуре *цикл–до* вычисляется после исполнения действия *S*. В этом случае, независимо от истинности или ложности условия, действие S будет исполнено хотя бы один раз.

Базовые структуры имеют один вход и один выход.

Рассмотренные базовые структуры могут соединяться одна с другой или содержаться одна в другой, как этого требует алгоритм решения задачи. На рис.7 показан пример комбинации структур *следование***,** *цикл-пока* и полная *развилка*.

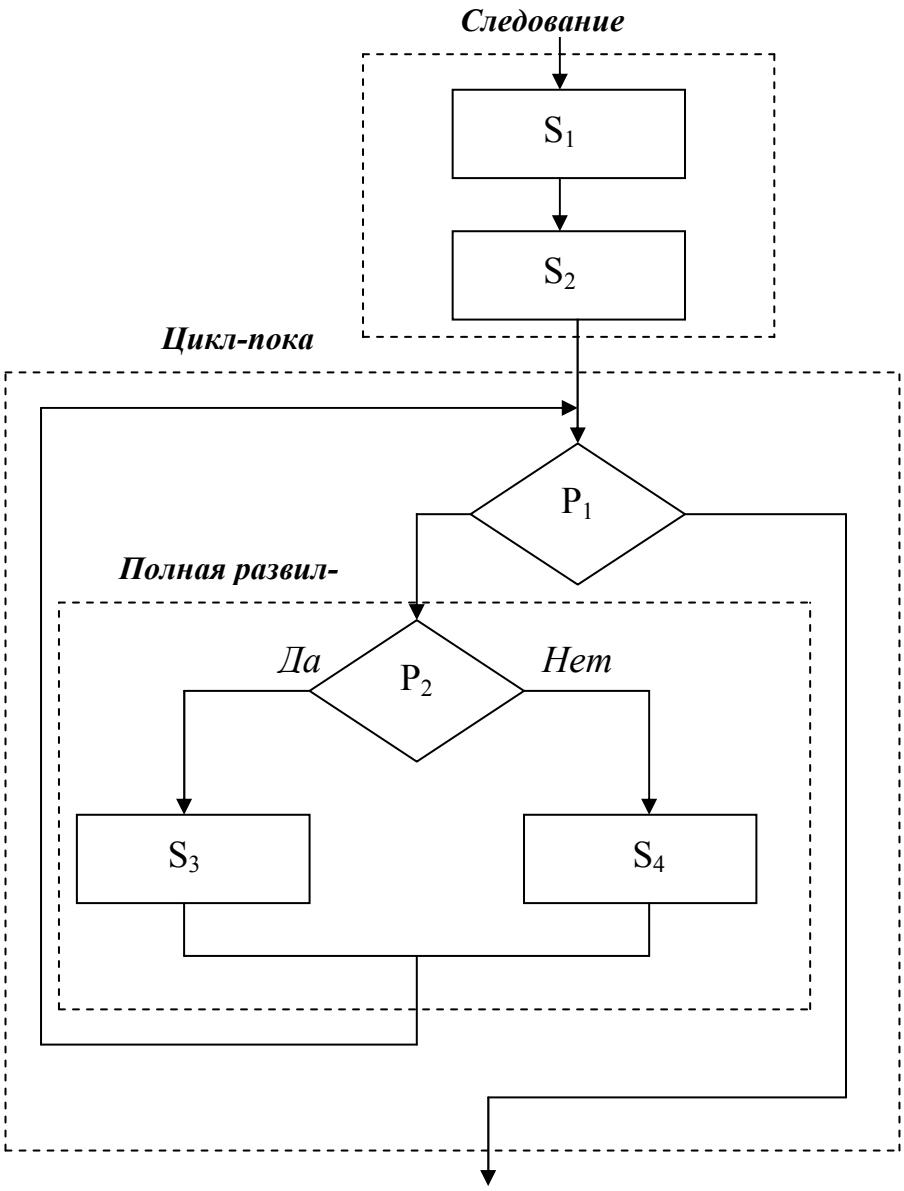


Рис. 7. Комбинация структур

Схема алгоритма, представленная комбинацией базовых структур, называется *структурированной*. Решение любой задачи можно представить в виде структурированной схемы алгоритма.

Контрольные вопросы

1) Что такое алгоритм? Какими свойствами он обладает?

2) *Какие вы знаете способы записи алгоритма?*

СРЕДСТВА РЕАЛИЗАЦИИ АЛГОРИТМА

Записи алгоритма решения задачи в словесной и даже графической форме допускают некоторый произвол при изображении команд. Наличие такого произвола удобно для человека и одновременно позволяет ему понять и исполнить алгоритм правильно. Для компьютера же такой язык будет непонятен. Здесь необходим новый тип языка не допускающий ни малейшего произвола в толковании команд и одновременно понятный компьютеру, т.е.

строго формализованный язык. Языки такого типа получили название *языков программирования*.

Языков программирования разработано большое количество. Однако все они имеют основные компоненты: *алфавит, синтаксис и семантику*.

Алфавит языка программирования представляет собой фиксированный набор основных символов. Использовать символы не входящие в алфавит данного языка запрещено.

Синтаксис языка программирования есть совокупность правил построения фраз, позволяющих определить, правильно или неправильно написана та или иная фраза.

Семантика языка программирования есть система истолкований отдельных языковых конструкций, которая определяет смысловое значение фраз. Именно семантика, в конце концов, определяет, какой алгоритм определен данным текстом на алгоритмическом языке.

Критерии выбора языка программирования

Чаще всего выбор языка программирования диктуется очень простыми «земными» факторами – доступностью того или иного транслятора и умением составлять программы на данном языке. Если, однако, в распоряжении пользователя имеется достаточно большой выбор языков программирования, то следует учитывать следующие обстоятельства:

- назначение разрабатываемой программы – нужна ли она временно или будет использоваться постоянно, будут ли разрабатываться ее новые версии;
- требуемая скорость программы, соотношение ее диалоговых и вычислительных компонентов;
- ожидаемый размер программы – можно ли ее будет создавать как единое целое или придется разбивать на отдельные взаимодействующие модули;
- необходимость сопряжения разрабатываемой программы с другими пакетами или программами, в том числе составленными на других языках программирования;
- характер и уровень использования аппаратных средств.
- проектирование алгоритмов и программ может основываться на различных подходах, среди которых наиболее распространены:
- структурное проектирование и программирование;
- информационное моделирование предметной области и связанных с ней приложений;
- объектно-ориентированное проектирование и программирование.

Технологии **структурного подхода** тесно связаны со структурой и формой представления данных. Данный подход используется при структурном анализе и построении моделей системных процессов, в разработке сложных функциональных структур и программных комплексов. Структурное программирование основано на модульной структуре построения программного продукта и применении управляющих структур алгоритмов модулей. При разработке программ используются ***языки программирования***. Наибо-

лее часто применяемыми языками программирования при структурном подходе являются язык С и TPascal.

Пример 1

Программа чтения символа с клавиатуры на языке TPascal.

```
Begin
     Write ('Введите любой символ:');
     Readln (ch);
     Writeln (ch, '=', ord (ch));
End.
```

Пример 2

Вызов некоторой функции на языке **С**.

```
Void func(int arg1,int arg2,int arg3); /* прототип функции */
Main()
   {
     int arg1=1;
     int arg2=5;
     int arg3=4;
     func (arg1, arg2, arg3); /* вызов функции*/
     . . .
   }
```

Оба эти языка позволяют работать с данными сложной структуры; имеют развитые средства для выделения отельных частей программы в процедуры. Трансляторы с этих языков работают в режиме компиляции, что позволяет создавать эффективные машинные программы. Важным средством для построения больших программных систем является их модульность, т.е. возможность независимой разработки отдельных частей программ и последующего связывания их в отдельную систему. Все эти особенности способствовали тому, что именно на **TURBOPASCALи СИ** разрабатывается большинство крупных программных систем для персональных компьютеров.

Информационное моделирование предметной области имеет решающее значение для разработки алгоритмов и программ, работающих с базами данных (БД). В основе данного подхода лежит положение об определяющей роли и независимости данных при проектировании алгоритмов и программ. Подход сложился в условиях появления концепции БД, его составными составляющими являются:

- информационный анализ предметной области;
- построение взаимосвязанной модели данных;
- системное проектирование функций обработки данных;
- детальное конструирование процедур обработки данных.

Строятся информационные модели различных уровней представления:

- интегрированная информационно-логическая модель предметной области, не зависящая от средств программной реализации хранения и обработки данных;
- даталогическая модель, ориентированная на среду хранения и обработки данных – модель системы управления базами данных (СУБД).

Примером такого подхода является СУБД *Microsoft Access*.

Объектно-ориентированный подход использует следующие базовые понятия: класс, объект, событие, свойства объекта, методы обработки. В отличие от традиционного структурного подхода, объектно-ориентированный подход к проектированию программных продуктов основан на:

- выделении классов;
- установлении характерных свойств классов и методов их обработки;
- создании иерархии классов;
- наследовании свойств классов и методов их обработки.

Каждый объект объединяет в себе как данные, так и программу обработки этих данных. *Объект* – это конкретный экземпляр класса. С помощью класса один и тот же программный код можно многократно использовать для различных объектов одного и того же класса.

Для проектирования программных продуктов разработаны объектно-ориентированные технологии, которые включают в себя специализированные языки программирования и инструментальные средства разработки пользовательского интерфейса. В качестве примера можно привести популярные языки программирования **си++, Visual Basic, Delphi**.

Для представления данных в удобном виде используют таблицы. Компьютер позволяет представлять их в электронной форме, а это дает возможность не только отображать, но и обрабатывать данные. Класс программ, используемых для этой цели, называется электронными таблицами. Особенно эффективно использование электронных таблиц при выполнении ряда экономических расчетов и составлении финансовой документации, а также при решении ряда задач. Кроме экономических задач, в Excel можно решать и инженерно – технические задачи, например для:

- проведения однотипных расчетов над большими наборами данных;
- автоматизации итоговых вычислений;
- решения задач путем подбора параметров;
- обработки результатов экспериментов;
- подготовки табличных документов;
- построение диаграмм и графиков по имеющимся данным.

В случае, когда требуется нестандартная обработка данных, Excel предоставляет дополнительные услуги, в частности, помимо встроенных макросов Excel содержит встроенный язык программирования *Visual Basic for Application – VBA.*

Наряду с перечисленными средствами проектирования и решения задач существуют специальные программные продукты, которые предназначены для решения узкого класса задач, а именно задач, связанных с математической обработкой данных, построением графиков функций и т.д.

К такого рода задачам относятся, например, следующие:

- подготовка научно-технических документов, содержащих текст и формулы, записанные в привычной для специалистов форме;
- вычисление результатов математических операций, в которых участвуют числовые константы, переменные и размерные физические величины;

- операции с векторами и матрицами;
- решение уравнений и систем уравнений;
- статистические расчеты и анализ данных;
- построение двухмерных и трехмерных графиков;
- дифференцирование и интегрирование (аналитическое и численное);
- решение дифференциальных уравнений;
- проведение серий расчетов с разными значениями начальных условий и других параметров.

Для решения подобных задач были разработаны специальные программные продукты. В настоящее время известны такие математические пакеты как ***Matlab, Matcad.***

MatCad представляет собой автоматизированную систему, позволяющую динамически обрабатывать данные в числовом и аналитическом (формульном) виде. Эта программа сочетает в себе возможность расчетов и подготовки форматированных научных и технических документов. В программе *MatCad* для этих целей существуют два вида объектов: *формулы и текстовые блоки*. Формулы выполняются с использованием числовых констант, переменных, функций (стандартных и определенных пользователем), а также общепринятых обозначений математических операций. Графики, которые автоматически строятся на основе результатов расчетов, также рассматриваются как формулы.

Система ***MatLab*** предназначена для выполнения инженерных и научных расчетов и высококачественной визуализации получаемых результатов. Эта программа применяется для математических расчетов, моделирования физических систем и управления техническими объектами. Существуют версии системы *MatLab* для операционных систем WINDOWS и UNIX.

Пакет ***Mathematica*** позволяет осуществить широкий спектр символьных преобразований, включающих наряду с другими, операции математического анализа. Одной из сильных сторон рассматриваемого продукта – развитая двух и трехмерная графика, используемая для визуализации математических объектов. По своей сущности *Mathematica* представляет собой язык программирования высокого уровня, позволяющий реализовывать традиционные процедурный и функциональный стили программирования, а также стиль правил преобразований, что превращает эту программу в мощный и удобный инструмент для математических исследований.

В последнее время приобрел широкую популярность пакет – ***Maple*** – мощный инструмент для решения математических задач и подготовки научных публикаций. Система ***Maple*** является интегрированной рабочей средой, объединяя возможности проведения аналитических преобразований, численных вычислений и подготовки текстов в едином рабочем документе. Вычислительные возможности ***Maple*** охватывают все практические разделы современной математики. Всего в системе содержится более чем 2700 функций, операторов, команд, которые появились только в ***Maple*** (особенно это относится к решению дифференциальных уравнений, дифференциальных уравне-

ний в частных производных, трехмерной евклидовой геометрии и дифференциальной алгебре).

СТРУКТУРА ПРОГРАММЫ И ЕЕ ЭЛЕМЕНТЫ

Основные элементы программирования

Большинство программ создаются для решения какой-либо конкретной задачи. В процессе решения задачи на компьютере нужно ввести обрабатываемые данные, указать, как их обрабатывать, задать способ вывода полученных результатов. С этой целью необходимо знать:

- как ввести информацию в память (ввод);
- как хранить информацию в памяти (данные);
- как указать правильные команды для обработки данных (операции);
- как передать обратно данные из программы пользователю (вывод).

Упорядочить команды нужно таким образом, чтобы:

- некоторые из них выполнялись только в том случае, если соблюдается определенное условие или ряд условий (условное выполнение);
- другие выполнялись повторно некоторое число раз (циклы);
- третьи выделялись в отдельные части, которые могут быть неоднократно выполнены в разных местах программы (подпрограммы).

Алфавит и словарь языка TurboPascal (TPascal)

Языком - называется совокупность символов, соглашений и правил, используемых для общения.

При записи алгоритма решения задачи на языке программирования необходимо четко знать правила написания и использования элементарных информационных и языковых единиц. Основой любого языка программирования является *алфавит* – конечный набор знаков, состоящий из букв, десятичных и шестнадцатеричных цифр, специальных символов.

В качестве букв в TPascal используются строчные и прописные буквы латинского алфавита и знак подчеркивания - "_", десятичные цифры от 0 до 9, шестнадцатеричные цифры, включающие в себя комбинацию цифр от 0до 9 и букв от A до F.

При написании программ применяются следующие специальные символы: **"+"; "-"; "*", "/", ">"; "<"; "="; ";"; "#"; "'"; ","; "."; "[ ]"; "{}"; "$"; "( )"; "^";"@".**

Комбинации специальных клавиш могут образовывать составные символы: **":="; "<>"; ">="; "<="** и т.д.

В программе эти символы нельзя разделять пробелами, если они используются как знаки операций отношения или ограничители комментария.

Слова в TPascal

Неделимые последовательности знаков алфавита образуют *слова,* отделенные друг от друга разделителями и несущие определенный смысл в программе. Разделителем может быть пробел, символ конца строки, комментарий. Набор слов, используемый в TPascal, можно разделить на три группы: зарезервированные слова, стандартные идентификаторы и идентификаторы пользователя.

Зарезервированные слова являются составной частью языка, имеют фиксированное начертание и определенный смысл. Они не могут изменяться программистом. В качестве примера ниже приводятся несколько таких слов: **Begin, Go to, Else, If, Var, Const, Set, Case, Array, Until, Function, And, Or** и т.д. – всего 55 слов.

Зарезервированные слова нельзя использовать в качестве имен для обозначения каких-либо величин, введенных в программе.

Группа слов, имеющих некоторый смысл, называется *словосочетанием*. В языке программирования словосочетание, состоящее из слов и символов и задающее правило вычисления некоторого значения, называется *выражением*. Минимальная конструкция языка, представляющая законченную мысль, есть *предложение*. Если предложение языка программирования задает полное описание некоторого действия, которое необходимо выполнить, оно называется *оператором*. Предложение, описывающее структуру и организацию данных - объектов языка, над которыми производятся различные действия, называется *описанием*. Для того чтобы научиться правильно писать программы для компьютера, необходимо изучить *синтаксис языка* программирования (правила записи его конструкций) и его *семантику* (смысл и правила использования этих конструкций).

Идентификаторы

Для того, чтобы программа решения задачи обладала свойством массовости, следует употреблять не конкретные значения величин, а использовать их обозначения для возможности изменения по ходу выполнения программ их значений. Для обозначения программ, а в программе переменных и постоянных величин, различных процедур, функций, объектов используются имена - *идентификаторы* (*identification* - установление соответствия объекта некоторому набору символов).

Для обозначения заранее определенных разработчиками языка типов данных, констант, процедур и функций служат *стандартные идентификаторы,* например: Integer, Sin, Cos, Read, Readlin. В этом примере стандартный идентификатор Sin вызывает функцию, вычисляющую синус заданного угла, Read, Readln вызывают процедуру, организующую ввод данных.

Для обозначения меток, констант, переменных, процедур и функций, определенных самим программистом, применяются *идентификаторы пользователя*. При этом идентификаторы в программе должны быть уникальны, т.е. в данном блоке программы не может использоваться один идентификатор для обозначения более чем одной переменной или постоянной величины.

При записи программ следует соблюдать *общие правила написания идентификаторов*:

Идентификатор начинается только с буквы или знака подчеркивания (исключение составляют метки, которые могут начинаться и цифрой и буквой).

Идентификатор может состоять из букв, цифр и знака подчеркивания (пробелы, точки и другие специальные символы алфавита при написании идентификаторов недопустимы).

Между двумя идентификаторами должен быть по крайней мере один пробел.

Максимальная длина идентификатора 127 символов, но значимы только первые 63 символа.

При написании идентификаторов можно использовать как прописные, так и строчные буквы. Компилятор не делает различий между ними.

Константы и переменные

Константами называют элементы данных, значения которых установлены в описательной части программы и в процессе выполнения программы не изменяются. Константы задаются идентификаторами пользователя. Все константы должны быть описаны в специальном разделе, который начинается зарезервированным словом Const **(**constant - константа).

Формат:

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

В TPascal имеется ряд констант, к значениям которых можно обращаться без предварительного определения. Их называют зарезервированными константами. Наиболее употребительные из них приведены в таблице 1.

Таблица 1

Зарезервированные константы

Идентификатор	Тип	Значение	Описание
True	Boolean	True	Истина
False	Boolean	False	Ложь
Maxint	Integer	32767	Максимальное целое

Переменными называют величины, которые могут менять свои значения в процессе выполнения программы. При объявлении переменной выделяется определенное количество ячеек памяти. Само название "переменная" подразумевает, что содержимое объявленной области памяти будет изменяться в ходе выполнения программы. Переменные описываются в специальном разделе, который начинается зарезервированным словом Var (variable - переменная). Формат:

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

Например, :

```
Var
A, B : Integer;
Summa: Real;
```

Кроме констант и переменных существуют так называемые *типизированные константы,* которые являются как бы промежуточным звеном между переменными и константами. Слово "константа" означает, что данные этого типа описываются в разделе Const, а слово "типизированные" указывает, что для них должен указываться и тип, как у переменных. Формат:

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

Например:

```
Const
```

```
Predmet: String = ' Информатика ' ;
Ocenka: Byte = 4;
```

В прикладном аспекте типизированная константа равнозначна переменной с заранее инициализированным значением, и в программе действия над ней могут производиться так же, как и над переменной.

Структура программы

Программа реализует алгоритм решения задачи. В ней записывается последовательность действий, выполняемых над определенными данными с помощью определенных операций для реализации заданной цели. Основные характеристики программы:

- точность полученного результата;
- время выполнения;
- объем требуемой памяти

Программа на языке TPascal состоит из строк. Набор текста программы осуществляется с помощью встроенного редактора текстов системы программирования TPascal.

Строки могут начинаться с любой колонки. Количество операторов в строке произвольно, но если в строке записан один оператор, такая строка легче читается. Синтаксически программа состоит из необязательного заголовка и блока (рис.8):

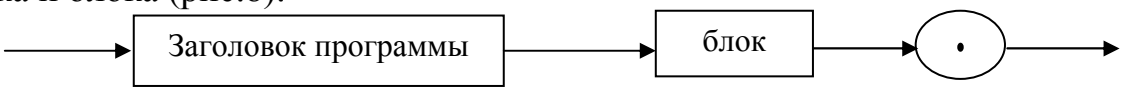


Рис.8. Синтаксическая структура программы

Блок может содержать в себе другие блоки. Блок состоит из двух частей: описательной и исполнительной. Первая часть может отсутствовать, без второй блок не имеет смысла. Блок, который не входит ни в какой другой блок называется *глобальным* - это основная программа - он должен присутствовать в любом случае. Если в глобальном блоке находятся другие блоки, они называются *локальными*. Локальные блоки - это процедуры и функции, их присутствие необязательно. Объекты программы (типы, переменные, константы и др.) соответственно называются локальными и глобальными. Область действия объектов - блок, где они описаны, и все вложенные в него блоки. Блочная структура обеспечивает структуризацию программ на уровне исходных текстов. В идеальном случае программа на языке TPascal состоит из процедур и функций, которые вызываются из раздела операторов основной программы. Синтаксическая диаграмма блока имеет вид, представленный на рис.9.

В начале программы находится заголовок, состоящий в общем случае 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.

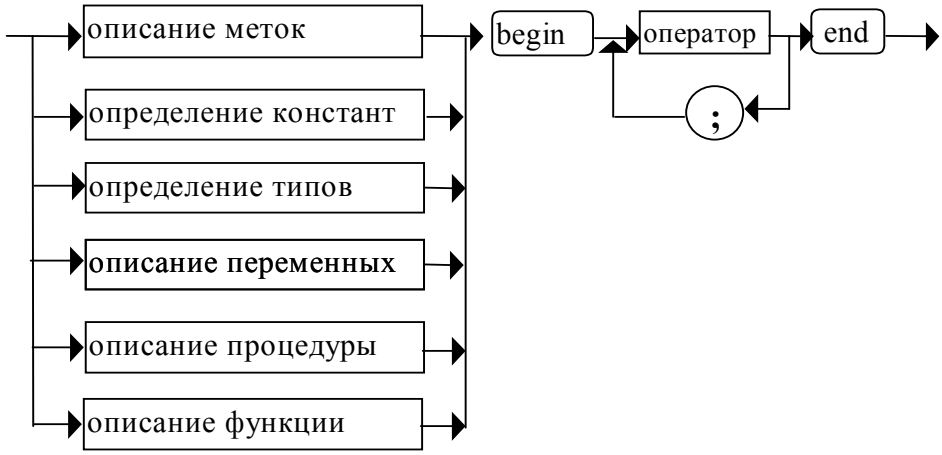


Рис.9. Синтаксическая диаграмма программного блока

После заголовка следует программный блок, состоящий в общем случае из семи разделов:

— список имен подключаемых библиотечных модулей (он определяется зарезервированным словом *uses*);
— описание меток;
— описание констант;
— определение типов данных;
— описание переменных;
— описания процедур и функций;
— операторов.

Разделы описаний (кроме *uses,* который всегда расположен после заголовка программы) могут встречаться в разделе объявлений и соглашений любое количество раз и следовать в произвольном порядке.

Раздел Uses

Этот раздел состоит из зарезервированного слова uses и списка подключаемых стандартных и пользовательских библиотечных модулей.

Формат:

```
Uses <имя1 >, <имя2>, ...;
```

Например:

```
Uses Crt, Dos, MyLib
```

Раздел описания меток

Перед любым оператором языка TPascal можно поставить метку, что позволяет выполнить прямой переход на этот оператор с помощью оператора перехода go to из любого места программы.

Раздел описания констант

В разделе описания констант производится присваивание идентификаторам констант постоянных значений. Начинается раздел зарезервированным словом const за которым следует ряд выражений присваивания, которые отделяются друг от друга точкой с запятой.

Формат:

```
Const
```

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

Например:

```
Name = 'Петя';        {Строковая константа}
Code=$124;            {Константа - шестнадцатеричное значение}
MaxInd:word =100      {Типизированная константа}
```

Раздел описания типов данных

Тип данных может быть описан непосредственно в разделе описания переменных, либо определяется идентификатором типа. Стандартные типы не требуют описания в отличие от типов, описанных пользователем.

Раздел описания типов данных начинается зарезервированным словом **Type**, за которым следует одно или несколько определений типов, разделенных точкой с запятой.

Формат:

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

Например:

```
Type
 Days = 1..31;
 Matr = array[1..10] of integer;
```

Раздел описания переменных

Каждая переменная, встречающаяся в программе, должна быть описана в разделе описания переменных, который начинается зарезервированным словом **Var** (variable - переменная), затем через запятую перечисляются имена переменных одинакового типа, а через двоеточие следуют их тип и точка с запятой.

Формат

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

Например:

```
A, B, Proizved:Integer;
```

Раздел описания процедур и функций

В этом разделе размещаются тела подпрограмм. **Подпрограммой** называется программная единица, имеющая имя, по которому она может быть вызвана из других частей программы. В языке TPascal роль подпрограмм выполняют процедуры и функции. Для их описания используются зарезервированные слова **procedure** и **function**, которые записываются в начале подпрограммы.

Формат процедуры:

```
Procedure <имя процедуры> {<параметры>};
     <разделы описаний >
     <раздел операторов>
End;
```

Формат функции:

```
Function <имя функции> {<параметры>}: <тип результата>;
     <разделы описаний>
     <раздел операторов>
End;
```

Процедуры и функции подразделяются на стандартные и определенные пользователем. Стандартные процедуры и функции являются частью языка и могут вызываться без предварительного описания. Описание пользовательских процедур и функций обязательно.

Раздел операторов

Раздел операторов является основным, так как именно в нем с предварительно описанными переменными, константами, значениями функций выполняются действия, позволяющие получить конечный результат. Начинается раздел зарезервированным словом **Begin** (начало), далее следуют операторы языка, отделенные друг от друга точкой с запятой. Завершает раздел зарезервированное слово **End** (конец). Эти зарезервированные слова являются аналогом открывающей и закрывающей скобки в обычных арифметических выражениях.

Операторы выполняются строго последовательно в том порядке, в котором они записаны в теле программы в соответствии с синтаксисом и правилами пунктуации.

Комментарии

Для лучшего понимания программы в ней записывается пояснительный текст - **комментарий.** Комментарий можно записывать в любом месте программы, где разрешен пробел. Текст комментария ограничен символами {} или (* *) и может содержать любые комбинации русских и латинских букв, цифр и символов. Ограничений на длину комментария нет.

Комментарий игнорируется компилятором и поэтому никакого влияния на программу не оказывает. По месту положения в программе комментарии можно подразделить на четыре класса: объясняющие положение программы, поясняющие смысл идентификаторов переменных и констант, описывающие логически обособленные части программы, объясняющие трудно понимаемые элементы алгоритма.

ТИПЫ ДАННЫХ

При решении задач выполняется обработки информации различного характера. Это могут быть целые и дробные величины, строки. Для описания множества допустимых значений величины и совокупности операций, в которых может участвовать данная величина, используется указание ее типа данных. **Тип данных** (data type) - множество величин, объединенных определенной совокупностью допустимых операций. Каждый тип имеет свой диапазон значений и специальное зарезервированное слово для описания. В **TURBOPASCAL**для описания типа в общем случае используется зарезервированное слово type.

Формат:

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

Все типы данных можно разделить на **скалярные** и **структурированные** (составные). Скалярные типы, в свою очередь, делятся на стандартные и

пользовательские. К стандартным типам относятся целочисленные, вещественные, литерные, булевские типы данных и указатели.

Пользовательские типы разрабатываются пользователями системы Турбо Паскаль.

Скалярные типы данных

К скалярным (scalar - простые) типам данных относят типы данных таких величин, значения которых не содержат составных частей.

Целочисленные типы данных

Целочисленные типы данных представляют собой значения, которые могут использоваться в арифметических выражениях и занимать в памяти от 1 до 4 байт (табл. 2)

Таблица 2

Целочисленные типы данных

Тип	Диапазон	Требуемая память (байт)
byte	0..255	1
shorint	- 128..127	1
integer	-32768..32767	2
word	0..65535	2
longint	2147483648..2147483647	4

Значения целых типов могут изображаться в программе двумя способами: в десятичном виде (последовательность цифр) и в шестнадцатеричном виде (в этом случае вначале числа ставится знак $, а цифры старше 9 обозначаются латинскими буквами от A до F).

Над данными целого типа определены следующие операции

- арифметические, возвращающие результат выполнения над целыми операндами в виде целого числа - + ; - ; *; /; div; mod;

- отношения - <; >; >=; <=; =; <>, вырабатывающие результат логического типа.

Над целыми числами определен ряд стандартных функций, например:

Chr(x) - возвращает символ, ASCII-код которого равен x;

Sqr(x) - возвращает квадрат числа x;

Sqrt(x) - возвращает значение корня квадратного из x;

Exp(x) - возвращает **e** в степени **x** (экспоненту), результат вещественного типа;

и ряд других функций.

Для целых чисел определены следующие стандартные процедуры:

- **dec (x,i)** - уменьшает значение x на i, если i не задано, то на 1;

- **inc (x,i)** - увеличивает значение ч на i, если i не задано, то на 1.

Вещественные типы данных

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 и с плавающей (т.е. парой чисел вида <мантисса>E<порядок>), и с фиксированной точкой (например,7.32) (табл.3).

Таблица 3

Вещественные типы данных

Тип	*Диапазон*	*Мантисса*	*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

Над вещественными данными определены такие же арифметические и логические операции, как и над данными целого типа. Отличие заключается в том, что результат выполнения арифметических операций получается также вещественного типа.

Для вещественных чисел также определены стандартные математические функции, но есть и специализированные функции, такие как:

Trunc(x) -преобразует вещественный аргумент x в целое число путем отбрасывания дробной части;

Round(x) - преобразует вещественный аргумент x в целое число путем округления до ближайшего целого.

Литерный (символьный) тип

Литерный (символьный) тип **Char** определяется множеством значений кодовой таблицы ПК. Каждому символу приписывается целое число в диапазоне от 0 до 255. Для кодировки используется код ASCII. Для размещения в памяти переменной литерного типа требуется один байт.

В программе значения переменных и констант типа *char* должны быть заключены в апострофы. Например, 'A' обозначает букву A, ' ' - пробел, ';' - точку с запятой.

Над данными символьного типа определены следующие операции отношения **=, <>, <, >, <=, >=,** вырабатывающие результат логического типа.

Для данных символьного типа определены следующие стандартные функции:

Chr(x) - преобразует выражение x типа byte в символ и возвращает значение символа;

Ord(ch) - преобразует символ ch в его код типа byte и возвращает значение кода;

Pred(ch) - возвращает предыдущий символ;

Succ(ch) - возвращает следующий символ.

Булевский тип

Булевским типом называют тип данных, представляемый двумя значениями: True (истина) False (ложь). Он широко применяется в логических выражениях и выражениях отношения. При описании этого типа указывают

слово boolean. Для размещения в памяти переменной булевского типа требуется 1 байт.

Например:

```
Flag, Result: Boolean;
```

Пользовательские типы

Кроме стандартных типов данных **ПАСКАЛЬ** поддерживает скалярные типы, определенные самим пользователем. К ним относятся перечисляемый и интервальный типы.

Данные этих типов занимают в памяти 1 байт, поэтому скалярные пользовательские типы не могут содержать более 256 элементов. Их применение обеспечивает семантический контроль вводимых данных, значительно улучшает наглядность программ, делает более легким поиск ошибок и экономит память.

Перечисляемый тип

Перечисляемый тип (enumerated type) - тип данных, заданных списком принадлежащих ему значений.

Объявление перечисляемого типа описывает множество идентификаторов, которые являются возможными значениями перечисляемого типа. Идентификаторы в описании типа представляют собой константы. Отдельные значения указываются через запятую, а весь список заключается в круглые скобки. Первая константа имеет порядковый номер нуль, вторая 1 и т.д.

Формат:

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

Например:

```
Type
     Gaz = (Ge, C, O, N);
Var
     g1,g2,g3 : Gaz;
```

Интервальный тип (диапазон)

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 (тип real здесь недопустим). Значение первой константы обязательно должно быть меньше значения второй.

Формат

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

```
     <идентификатор, ...>:<имя типа>;
```

Например:

```
Type
     Days= 1..31;
Var
     RabDay, BolDay: Days;
```

Структурированные типы данных

Структурированные типы данных определяют упорядоченную совокупность скалярных переменных и характеризуются типом своих компонентов. В языке Паскаль допускаются следующие структурированные типы данных: строки, массивы, множества, записи, файлы, указатели, процедурные типы и объекты.

Массивы

Массив - это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип. Название **регулярный тип** (или ряды) массивы получили за то, что в них объединены однотипные (логически однородные) элементы, упорядоченные (урегулированные) по индексам, определяющих положение каждого элемента в массиве. Элементы, образующие массив, упорядочены таким образом, что каждому элементу соответствует совокупность номеров (индексов), определяющих его местоположение в общей последовательности. Доступ к каждому отдельному элементу осуществляется путем индексирования элементов массива. Индексы представляют собой выражения любого скалярного типа, кроме вещественного. Тип индекса определяет границы изменения индекса. Для описания массива предназначено словосочетание array of (массив из).

Формат:

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

Например:

```
Type
  znak= array[1..255] of char;
Var
  M1:znak;              {Тип znak предварительно описан в разделе типов}
  M2:array[1..60] of integer; {Прямое описание массива M2}
```

Тип элементов массива называется **базовым**. Если в качестве базового взят другой массив, образуется структура, которую принято называть многомерным массивом.

Пример.

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

Ту же структуру можно получить, используя другую форму записи:

```
Var
```

```
Matr: Array[1..4, 1..4] of Integer;
```

Если в такой форме описания массива задан один индекс, массив называется **одномерным**, если два индекса - **двумерным**, если n индексов - **n-мерным**.

Элементы массива располагаются в памяти последовательно. Элементы с меньшими значениями индекса хранятся в более низких адресах памяти. Многомерные массивы располагаются таким образом, что самый правый индекс возрастает самым первым.

ВВОД-ВЫВОД ДАННЫХ

Общие сведения

Решение самой простой задачи на компьютере не обходится без операции ввода-вывода информации. Ввод данных - это передача информации от внешнего носителя в оперативную память для обработки. Вывод - обратный процесс, когда данные передаются после обработки из оперативной памяти на внешний носитель. Внешним носителем может служить терминал ввода-вывода, принтер, жесткий (винчестер) магнитный диск и другие устройства.

В языке TPascal стандартным средством общения пользователя и компьютера являются преопределенные файлы (Input и Output), которые по умолчанию являются параметрами программы. Программа получает входные данные из файла Input и помещает результат обработки в файл Output. Стандартно файлу Input назначена клавиатура, а файлу Output – экран терминала.

Процедуры ввода-вывода

Для выполнения операций ввода-вывода служат четыре процедуры: **Read, Readln, Write, Writeln.**

Процедура чтения **Read** обеспечивает ввод числовых данных, символов, строк и т.д., для их последующей обработки программой. Формат:

```
Read(X1,X2,…,Xn);
```

или

```
Read(FV,X1,X2,…,Xn);
```

где X1,X2,…,Xn – переменные допустимых типов данных; FV – переменная, связанная с файлом, откуда будет выполнятся чтение. Значения X1,X2,…,Xn набираются минимум через один пробел на клавиатуре и высвечиваются на экране. После набора данных для одной процедуры **Read** нажимается клавиша ввода Enter.

Значения переменных должны вводиться в строгом соответствии с синтаксисом языка TPascal. Если соответствие нарушено (например, X1 имеет тип `integer`, а при вводе набирается значение типа `char`), то возникают ошибки ввода-вывода. Сообщение об ошибке имеет вид: **I/O error XX**, где XX - код ошибки.

Пример

```
Var
     I:real;
     J:integer;
     K:char;
Begin
```

```
Read(I,J,K);
.   .   .
```

Набираем на клавиатуре: `212.45 38. П`

Если в программе имеется несколько процедур **Read**, данные для них вводятся потоком, т.е. после считывания значений переменных для одной процедуры **Read** данные для следующей процедуры **Read** набираются в той же строке, что и для предыдущей, до окончания строки, затем происходит переход на следующую строку.

Пример

```
Var
     A,B,Sum1:integer;
     C,D,Sum2:real;
     .   .   .
Begin
     Read(A,B);
     Sum1:=A+B;
     Read(C,D);
     Sum2:=C+D;
     .   .   .
End.
```

Набираем на клавиатуре: `18758 34 Enter 2.62E-02 1.54E+01 Enter.`

Процедура чтения **Readln** аналогична процедуре **Read**, единственное отличие заключается в том, что после считывания последнего в списке значения для одной процедуры **Readln** данные для следующей процедуры ввода будут считываться с начала новой строки. Если в предыдущем примере заменить **Read** на **Readln**:

```
.   .   .
     Readln(A,B);
     Sum1:=A+B;
     Readln(C,D);
     Sum2:=C+D;
     .   .   .
```

то после набора на клавиатуре значений для A и B курсор автоматически перейдет на новую строку, где будут набираться данные C и D, т.е.

```
18758 34 Enter
2.62E-02 1.54E+01 Enter
```

Процедура записи **Write** производит вывод числовых данных, символов, строк и булевских значений. Формат:

```
Write(Y1,Y2,…,Yn);
```

или

```
Write(FV,Y1,Y2,…,Yn);
```

где Y1,Y2,...,Yn – выражения типа `integer, byte, real, char, boolean` и т.д.; FV – имя файла, куда производится вывод.

Пример

```
uses Printer;
Var
```

```
      .  .  .
Begin
   Write(234);          {Выражение представлено значением}
   Write(A+B-2);        {Выводится результат выражения}
   Write('Результат вычисления=',Result1);
   {Выводится текст Результат вычисления= и значение Result1}
End.
```

Формат вывода

В процедурах вывода **Write** и **Writeln** имеется возможность записи выражения, определяющего ширину поля вывода.

Для задания формата вывода данных таких типов, как `integer, byte, word, char, string` используется следующий синтаксис:

`ВыводимыеДанные:N`

где выводимые данные – имя переменной или константы, выражение, имя функции, текст в апострофах; N – количество позиций, отводимых под выводимые данные.

Пример

Значение	Выражение	Результат
15	`Write(I:4);`	_ _ 1 5
'X'	`Write(C:3);`	_ _ X
'День'	`Write(S:4);`	Д е н ь
True	`Write(B:5);`	_ T r u e

Для задания формата вывода данных вещественного типа используется следующий синтаксис:

`ВыводимыеДанные:N:M`

где `ВыводимыеДанные` – имя переменной или константы, выражение, имя функции, текст в апострофах; N – количество позиций, отводимых под число, т.е. под целую часть, десятичную точку и дробную часть; M – количество позиций, отводимых под дробную часть числа.

Пример

Значение	Выражение	Результат
6.15	`Write(R:6:3);`	_ 6 . 1 5 _

ОПЕРАТОРЫ

Общие сведения

Оператором называется предложение языка программирования, задающее полное описание некоторого действия, которое необходимо выполнить. Основная часть программы на языке TPascal представляет собой последовательность операторов. Разделителем операторов служит точка с запятой.

Все операторы языка TPascal можно разделить на две группы: простые и структурные.

Простые операторы

Операторы, не содержащие никаких других операторов, называются *простыми*.

Оператор присваивания

Оператор присваивания (:=) предписывает выполнить выражение, заданное в его правой части, и присвоить результат переменной, идентификатор которой расположен в левой части. Переменная и выражение должны быть совместимы по типу. На рис.10 представлен общий вид оператора.

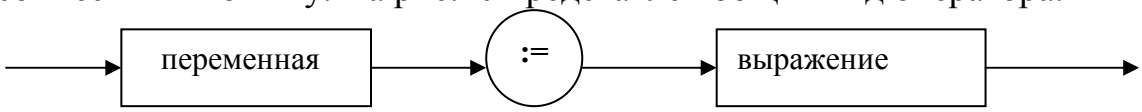


Рис.10. Общий вид оператора присваивания

Оператор присваивания выполняется следующим образом: сначала вычисляется выражение в правой части присваивания, а затем его значение присваивается переменной, указанной в левой части оператора.

Пример. Необходимо определить значение *y*: $y = \dfrac{a = b}{c}$

```
Program Primer;
Var
     A,B,C,Y:real;
Begin
     Write('A=');
     Readln(A);
     Write('B=');
     Readln(B);
     Write('C=');
     Readln(C);
     Y:=(A+B)/C;
     Writeln('Y=',Y:8:3);
End.
```

Структурные операторы

Структурные операторы представляют собой конструкции, построенные из других операторов по строго определенным правилам. Все структурные операторы можно разделить на три группы: составные, условные, повтора.

Составной оператор

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

Синтаксическая диаграмма составного оператора записывается так:

Составной оператор воспринимается как единое целое и может находиться в любом месте программы, где синтаксис языка допускает наличие оператора (рис.11).

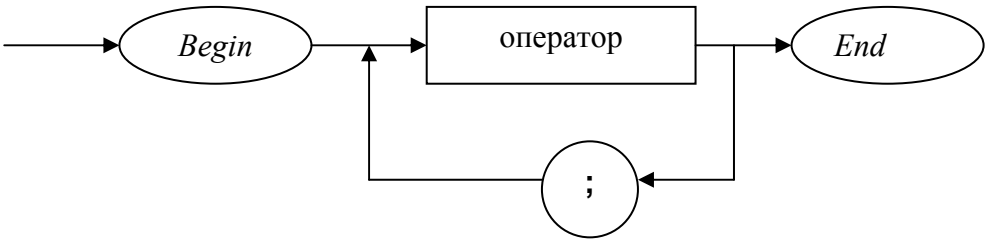


Рис.11. Структура составного оператора

Условные операторы

Условные операторы предназначены для выбора к исполнению одного из возможных действий (операторов) в зависимости от некоторого условия (при этом одно из действий может быть пустым, т. е. отсутствовать). В качестве условия выбора используется значение логического выражения. В TPascal имеются два условных оператора: **If** и **Case.**

Оператор условия **If.**

Синтаксическая диаграмма оператора условия **If** представлена на рис.12.

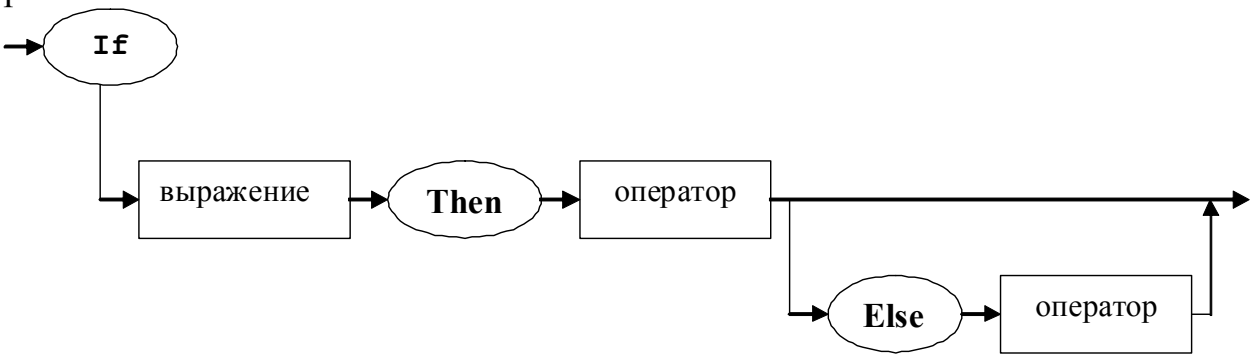


Рис.12. Синтаксическая диаграмма оператора условия

Как видно из диаграммы он может принимать одну из следующих форм:

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

2)  If <условие> Then <оператор>;
```

Оператор условия **If** выполняется следующим образом. Сначала вычисляется выражение, записанное в условии. В результате его вычисления получается значение булевского типа. В первом случаи, если значение выражения есть *True*(истина), выполняется <оператор1>, указанный после **Then** . Если результат вычисления выражения в условии есть *False* (ложь), то выполняется <оператор2>. Во втором – если результат выражения *True*, выполняется <оператор>, если *False* – оператор, следующий сразу за оператором **If.** Операторы **If** могут быть вложенными.

Пример фрагмента программы с оператором условия **If:**

```
Read(Ch);
If Ch='N' Then Parol:=True
Else Parol:=False;
Read(X);
If Parol=True
  Then    If X=100 Then
                    Writeln('Пароль и код правильные')
                  Else
                          Begin
                          Writeln('Ошибка в коде');
                          Halt(1)
                          End;
```

В данном примере с клавиатуры считывается значение переменной символьного типа *Ch*. Затем проверяется условие Ch='N'. Если оно выполняется, то переменной *Parol* присваивается значение *True*, если условие не выполняется - *False*. Затем с клавиатуры считывается значение кода *X*. Далее оператор *If* проверяет условие *Parol=True*. Если оно имеет значение *True*, то выполняется проверка введенного пароля оператором **If** *X*=100. Если условие *X*=100 имеет значение *True*, то выводится сообщение "Пароль и код правильные", и управление в программе передается на оператор, следующий за словом *End*. Иначе, если оно имеет значение *False*, выполняется составной оператор, стоящий после слова *Else*, который выводит на экран сообщение "Ошибка в коде", и вызывает стандартную процедуру **Halt**(1) для остановки программы.

Оператор выбора CASE.

Синтаксическая диаграмма оператора выбора **CASE** представлена на рис.13.

Следуя диаграмме (рис.13), получим следующий формат записи оператора **CASE:**

```
Case
<выражение> Of
     <список1>:<оператор1;>
     <список2>:<оператор2;>
     .   .   .
     <списокN>:<операторN;>
   Else <оператор>
End;
```

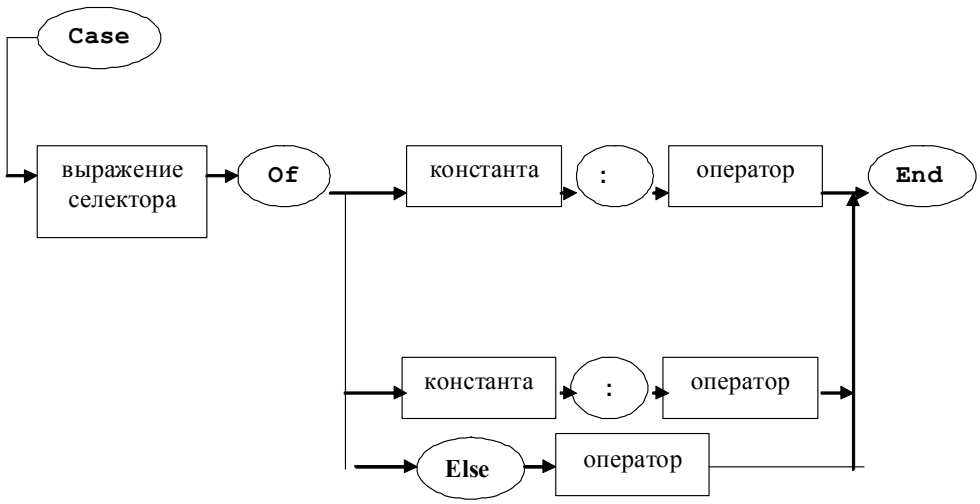


Рис.13. Синтаксическая диаграмма оператора выбора

Оператор **CASE** работает следующим образом. Сначала вычисляется значение *выражения-селектора*, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна текущему значению селектора, выполняется оператор, находящийся за словом *Else*. Если слово *Else* отсутствует, активи-

зируется оператор, находящийся за словом *End*, т.е. оператор за границей **CASE**.

Селектор должен относиться к одному из целочисленных типов (находящихся в диапазоне –32768..32767): булевскому, литерному или пользовательскому. Список констант выбора состоит из произвольного количества значений, или диапазонов, отделенных друг от друга запятыми. Границы диапазона записываются двумя константами через разграничитель “..”. Тип константы в любом случаи должен совпадать с типом селектора. В синтаксическом описании, приведенном выше, предполагается использование одного оператора для каждой альтернативы, но при необходимости можно задать несколько операторов, сгруппировав их в составной оператор. В тоже время ветвь *Else* допускает использование последовательность операторов, разделенных символом “;”.

При использовании оператора выбора **CASE** должны выполняться следующие правила:

Значения выражения селектора, записанного после служебного слова **CASE**, должны принадлежать дискретному типу (лат. Discretus – прерывистый, дробный, состоящий из отдельных частей); для **целого типа** они должны лежать в диапазоне *Integer*.

Все константы, предшествующие операторам альтернатив, должны иметь тип, совместимый с типом выражения.

Все константы в альтернативах должны быть уникальны в пределах оператора варианта (т.е. повторения констант в альтернативах не допускаются). Диапазоны не должны пересекаться и не должны содержать констант, указанных в этой или других альтернативах.

Пример программы с использованием оператора **CASE**, которая по введенному номеру дня недели выводит на экран монитора его название на русском языке.

```
Program Day_Week;
Var
     Day:byte;
Begin
     Write('Введите номер дня недели:);
     Readln(Day);
     Case Day Of
          1: Writeln('Понедельник');
          2: Writeln('Вторник');
          3: Writeln('Среда');
          4: Writeln('Четверг');
          5: Writeln('Пятница');
          6: Writeln('Суббота');
     Else
          Writeln('Воскресенье');
     End;
End.
```

Контрольные вопросы

1. Назначение, формы записи и порядок выполнения оператора условия **If**.

2. Особенности использования вложенных условных операторов.
3. Каковы отличия оператора выбора **CASE** от оператора условия **If**?
4. Какие правила должны выполняться при использовании оператора выбора **CASE**?

Операторы цикла

Если в программе возникает необходимость неоднократно выполнить некоторые операторы, то используются операторы цикла. В языке TPascal различают три оператора цикла: **While, Repeat, For**. Они используются для организации циклов различных типов. Выражение, управляющее повторениями, должно иметь булевский тип.

Если число повторений оператора (составного оператора) заранее неизвестно, а задано лишь условие его повторения (или окончания), используются операторы While, Repeat. Оператор For используется, если число повторений заранее известно.

Оператор While

Оператор **While** (пока) часто называют оператором цикла с предусловием за то, что условия выполнения тела цикла производится в самом начале оператора.

Синтаксическая диаграмма, для данного оператора представлена на рис.14.

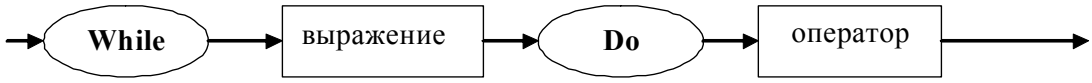


Рис.14. Синтаксическая диаграмма оператора While

Формат записи:

```
While <условие продолжения повторений> Do
                <тело цикла>;
```

Условие – булевское выражение, тело цикла – простой или составной оператор. 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*, происходят выход из цикла и переход к первому после *While* оператору.

Примером работы *While* может служить программа *DemoWhile,* которая производит суммирование 10 произвольно введенных целых чисел.

```
Program DemoWhile;
Const
     Limit=10;              {Ограничение на количество вводимых чисел}
Var
     Const,Item,Sum:Integer;
Begin
     Count:=0; {Счетчик чисел}
     Sum:=0;    {Сумма чисел}
     While (count<Limit) Do          {Условие выполнения цикла}
      Begin
```

```
        Count:=Count+1;              {Изменение счетчика чисел}
        Write('Введите ',Count:2,'-е целое число:');
        Readln(Item);                {Ввод нового числа}
        Sum:=Sum+Item;           {Добавление нового числа к сумме}
    End;
  Writeln('Сумма введенных чисел равна ',Sum:7:3)
End.
```

Оператор цикла Repeat

Оператор **Repeat** отличается от оператора **While** следующим: во-первых, тем, что условие проверяется после очередного выполнения операторов тела цикла (очередной итерации) и таким образом гарантируется хотя бы однократное выполнение цикла, во-вторых, тем, что критерием прекращения цикла является равенство выражения константе *True*. За это цикл **Repeat** часто называют циклом с постусловием, или циклом "ДО", так как он прекращает выполняться, как только значение выражения условия, записанного после слова **Until,** равно *True* (истина).

Оператор повтора **Repeat** состоит из заголовка **Repeat,** тела и условия окончания **Until**.

На рис.15 представлена синтаксическая диаграмма оператора.

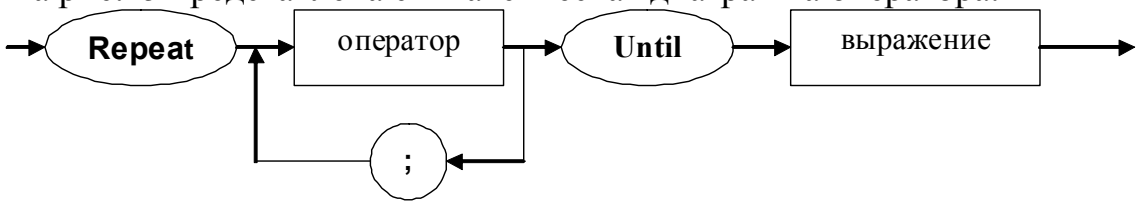


Рис.15. Синтаксическая диаграмма оператора Repeat

Формат записи:

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

Операторы, заключенные между словами **Repeat** и **Until** составляют тело цикла. Вначале выполняется тело цикла, затем проверяется условие выхода из цикла. Именно поэтому цикл, организованный с помощью оператора **Repeat,** в любом случае выполниться хотя бы один раз. Если же результат булевского выражения равен *False*, то тело цикла активизируется еще раз и т. д. Наконец, когда результат равен *True*, происходит выход из цикла.

При программировании операторов тела цикла следует обеспечить влияние по крайней мере одного из операторов тела цикла на значение условия, иначе цикл будет выполняться бесконечно.

Примером действия оператора **Repeat** может служить программа *Demorepeat*, которая вводит и суммирует любое количество целочисленных значений. Если введено значение 999, то на экран выводится результат суммирования.

```
Program DemoRepeat;
```

```
Var
     X:Integer;
     Sum:Real;
Begin
     Sum:=0;
     Repeat                                  {Повторять}
          Write('Значение X=');              {Начало тела цикла}
          Readln(X);
          If X<>999 Then                     {Ввод очередного значения X}
          Sum:=Sum+X;
     Until X=999;                    {Условие окончания цикла}
Writeln('Сумма введенных чисел=',Sum:8:3);
End.
```

Оператор цикла For

В случаях, когда число повторений может быть заранее известно, для организации обработки информации применяется оператор повтора **For**. Часто этот оператор повтора называют оператором цикла с параметром, так как число повторений задается переменной, называемой параметром цикла, или управляющей переменной. Оператор повтора **For** состоит из заголовка и тела цикла.

Синтаксическая диаграмма оператора приводится на рис.16.

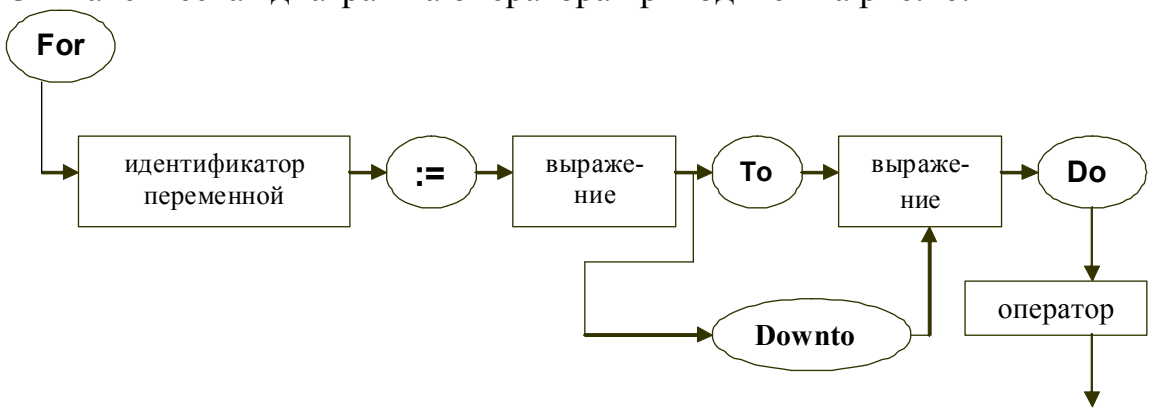


Рис.16. Синтаксическая диаграмма оператора For

Как видно из диаграммы, оператор *For* может быть представлен в двух форматах:

1) For <параметр цикла> :=<S1> **To** <S2> **Do** <оператор>;

2) For <параметр цикла> :=<S1> **Downto** <S2> **Do** <оператор>;

где S1 и S2 – выражение, определяющие соответственно начальное и конечное значение параметра цикла;

For … Do – заголовок цикла;

<оператор> - тело цикла.

Тело цикла может быть простым или составным оператором. Оператор *For* обеспечивает выполнение тела цикла до тех пор, пока не будет перебраны все значения параметра цикла от начального до конечного. Заголовок оператора повтора *For* определяет:

- диапазон изменения значений параметра цикла и одновременно число повторений оператора, содержащегося в теле цикла;

- направление изменения значения параметра цикла (возрастание – **To** или убывание – **Downto**).

Пример

```
For I:=1 To 100 Do
     Readln(M[I]);
For I:=100 Downto 1 Do
     Write(M[i]);
```

При первом обращении к оператору *For* вначале, вычисляются выражения S1 и S2, и осуществляется присваивание <параметр цикла>:=S1. После этого циклически повторяются следующие действия.

Проверяется условие <параметр цикла><=S2. Если условие выполнено, то оператор *For* продолжает работу (выполняется оператор в теле цикла), если условие <параметр цикла><=S2 не выполнено, то оператор *For* завершает работу, и управление в программе передается оператору, следующему за циклом.

Значение параметра цикла изменяется на +1 (**To**) или –1 (**Downto**) и далее с пункта 1. Обратите внимание, что шаг изменения параметра цикла – единица.

На использование параметра цикла в цикле *For* налагаются следующие ограничения.

— В качестве параметра цикла должна использоваться простая переменная, описанная в данном блоке.
— Параметр цикла должен иметь дискретный тип.
— Начальные и конечные значения диапазона должны иметь тип, совместимый с типом параметра цикла. При этом допускается любой скалярный тип, кроме вещественного.
— В теле цикла запрещается явное изменение значения параметра цикла (например, оператором присваивания).

Пример. Программа DemoFor выводит на экран таблицу перевода из градусов по шкале Цельсия(C) в градусы по Фаренгейту(F) для значений от 15^0C до 30^0C с шагом 1 градус.

Перевод осуществляется по формуле: F=C·1.8+32.

```
Program DemoFor;
Var
   I:Integer;
   F:Real;
Begin
     Writeln('    Температура');
     For I:=15 To 30 Do            {Заголовок цикла с параметром}
        Begin                      {Начало тела цикла}
             F:=I*1.8+32;
     Writeln('По Цельсию= ',I:2,' по Фаренгейту = ',F:5:2);
        End;                       {Конец тела цикла}
End.
```

Вложенные операторы цикла

Если телом цикла является циклическая структура, то такие циклы называют вложенными. Цикл, содержащий в себе другой цикл, называют

внешним, а цикл, содержащийся в теле другого цикла, называют внутренним. Внешний и внутренний циклы могут быть трех видов: циклами с предусловием *While*, циклами с постусловиями *Repeat* и циклами с параметрами *For*.

Правила организации, внешнего и внутреннего циклов, такие же, как и для простого цикла каждого из видов. Но при программировании вложенных циклов необходимо соблюдать следующее дополнительное условие: все операторы внутреннего цикла должны полностью располагаться в теле внешнего цикла.

Пример

Программа выводит на экран таблицу умножения от 1 до 10.

```
Program Tab_Umn1;
Var
     I,J:Byte;
Begin
 For I:=1 To 10 Do                          {Внешний цикл}
   For J:=1 To 10 Do                        {Внутренний цикл}
      Writeln(I:2,'*',J:2,'=',I*J:3);  {Тело внутреннего цикла}
End.
```

Контрольные вопросы.

1. Какое назначение операторов повтора(цикла)?
2. Какие требования предъявляются к выражениям, управляющим повторениями?
3. В чем отличия операторов повтора **While** и **Repeat**?
4. В каких случаях предпочтительнее использовать для организации циклов оператор повтора **For**? Что записывается в заголовке этого оператора?
5. Каким образом в операторе цикла **For** описывается направление изменения значения параметра цикла?
6. Какие ограничения налагаются на использование параметра цикла в цикле **For**?
7. Что такое вложенные циклы? Какие дополнительные условия необходимо соблюдать при организации вложенных циклов?

МАССИВЫ

Структурированные типы задают множества сложных значений, каждое из которых образует совокупность нескольких значений другого типа. В структурных типах выделяют регулярный тип (массивы).

Особенно часто с понятием “массив” приходится сталкиваться при решении научно-технических и экономических задач обработки совокупностей большого количества значений. В общем случае массив – это структурированный тип данных, состоящий из фиксированного числа элементов, имеющих один и тот же тип.

Действия над массивами

Для работы с массивами как единым целым используется идентификатор массива без указания индекса в квадратных скобках. Массив может участвовать только в операциях отношения “равно”, ”не равно” и операторе присваивания. Массивы, участвующие в этих действиях, должны быть иден-

тичны по структуре, т.е. иметь одинаковые типы индексов и одинаковые типы компонентов. Например, если массивы A и B описаны как

```
Var
A,B:Array[1..10] Of Real;
```

то применение к ним допустимых операций даст следующий результат (табл.4):

Таблица 4

Результат допустимых операций к массивам

Выражение	*Результат*
A=B	True, если значение каждого элемента массива A равно соответствующему значению элемента массива B
A<>B	True, если хотя бы одно значение элемента массива A не равно значению соответствующего элемента массива B
A:=B	Все значения элементов массива B присваиваются соответствующим элементам массива A. Значения элементов массива B остаются неизменны

Действия над элементами массива

После объявления массива каждый его элемент можно обработать, указав идентификатор (имя) массива и индекс элемента в квадратных скобках. Например, M[2], Vector[3], Mm[2,3] .

При обращении к элементам, возможно использование неявное обозначение индекса, а именно через другую переменную, имеющую тип индекса. Например, M[I], N[J], Mm[I,J].

Пример типичных ситуаций, возникающих при работе с данными типа `Array` (табл.5).

Предварительно сделаем следующее описание:

```
Const
     N=2;
     M=3;
Type
     Indn=1..N;
     Indm=1..M;
     Vec=Array[Indn] Of Integer;
     Mas=Array[Indn,Indm] Of Real;
Var
     A,B:Vec;
     Matr,M:Mas;
     K:byte;
     I,Imax,Imin:Indn;
     J,Jmax,Jmin:Indm;
     Sr:Real;
     Z:Integer;
```

Таблица 5

Типичные ситуации при работе с массивами

Одномерный массив	*Двумерный массив*
1. Инициализация (присваивание начальных значений элементам массива) заключается в присваивании каждому элементу массива одного и того же значения, соответствующего базовому типу.	

Обнуление
`For I:=1 To N Do`
`A[I]:=0;`
	`For I:=1 To N Do`
	`For J:=1 To M Do`
	`Matr[I,J]:=0;`

2. Ввод значений с клавиатуры	

`For I:=1 To N Do`
`Begin`
`Write('A[',I:1,']=');`
`Readln(A[I]);`
`End;`
	`For I:=1 To N Do`
	`For J:=1 To M Do`
	`Begin`
	`Write('Matr[',I:1,',',j:1,']=');`
	`Readln(Matr[I,J]);`
	`End;`

3. Формирование массива результатом случайной функции **Random**(I)	

`Randomize;`
`For I:=1 To N Do`
`A[I]:=Random(I);`
	`Randomize;`
	`For I:=1 To N Do`
	`For J:=1 To M Do`
	`Matr[I,J]:=Random(I);`

4. Вывод значений элементов массива	

в виде вектора
`For I:=1 To N Do`
`Write(A[I]:6);`
`Writeln;`
	в виде матрицы
	`For I:=1 To N Do`
	`Begin`
	`For J:=1 To M Do`
	`Write(Matr[I,J]:8:3);`
	`Writeln;`
	`End;`

5. Копирование значений одного массива соответствующим элементам другого массива	

`For I:=1 To N Do`
`B[I]:=A[I];`
	`For I:=1 To N Do`
	`For J:=1 To M Do`
	`M[I,J]:=Matr[I,J];`

6. Формирование одного массива из элементов другого массива, удовлетворяющих заданному условию	

`K:=0;`
`For I:=1 To N Do`
`If A[i]>o Then`
`Begin`
`K:=K+1;`
`B[K]:=A[I];`
`End;`
	`K:=0;`
	`For I:=1 To N Do`
	`For J:=1 To M Do`
	`If Matr[I,J]<>0 Then`
	`Begin`
	`K:=K+1;`
	`M[I,K]:=Matr[I,J];`
	`End;`

7. Поиск Поиск выполняется в циклическом сравнении значений всех элементов массива с искомым значением. *Количество нулевых элементов -?*	
`K:=0;`	`K:=0;`

For I:=1 **To** N **Do**
If A[i]=0 **Then**
K:=K+1;
	For I:=1 **To** N **Do**
	For J:=1 **To** M **Do**
	If Matr[I,J]= 0 **Then**
	K:=K+1;

Максимальный элемент -?	

Imax:=1;
For I:=2 **To** N **Do**
If A[i]>A[Imax] **Then**
Imax:=I;
	Imax:=1; Jmax:=1;
	For I:=1 To N Do
	For J:=1 To M Do
	If Matr[I,J]>Matr[Imax,Jmax]
	Then
	Begin
	Imax:=I;
	Jmax:=J;
	End;

Минимальное значение - ?	

Imin:=1;
For I:=2 **To** N **Do**
If A[i]<A[Imin] **Then**
Imin:=I;
	Imin:=1; Jmin:=1;
	For I:=1 **To** N **Do**
	For J:=1 **To** M **Do**
	If Matr[I,J]< Matr[Imin,Jmin]
	Then
	Begin
	Imin:=I;
	Jmin:=J;
	End;

Вычисление среднего арифметического значения -?	

K:=0; Sr:=0;
For I:=1 **To** N **Do**
If A[i]>0 **Then**
Begin
K:=K+1;
Sr:=Sr+A[I];
End;
Sr:=Sr/K;
	K:=0; Sr:=0;
	For I:=1 **To** N **Do**
	For J:=1 **To** M **Do**
	If Matr[I,J]> 0 **Then**
	Begin
	K:=K+1;
	Sr:=Sr+Matr[I,J];
	End;
	Sr:=Sr/K;

8. Перестановка значений элементов массива	
Z:=A[1]; A[1]:=A[2]; A[2]:=Z;	Sr:=Matr[1,1]; Matr[1,1]:=Matr[2,2]; Matr[2,2]:=Sr;

9. Сортировка массивов

Сортировка – один из наиболее распространенных процессов современной обработки данных. *Сортировкой* называется распределение элементов множества по группам в соответствии с определенными правилами.

9.1. Линейная сортировка (сортировка отбором)

Идея линейной сортировки по невозрастанию заключается в том, чтобы последовательно просматривая весь массив, отыскать наибольшее число и поместить его на первую позицию, обменяв его с элементом, который ранее занимал первую позицию. Затем просматриваются все остальные элементы массива и выполняется аналогичная операция по отбору из рассматриваемой

части максимального элемента и обмену местами этого элемента и первого в рассматриваемой части и т.д. Структурная схема алгоритма приведена на рис. 17.

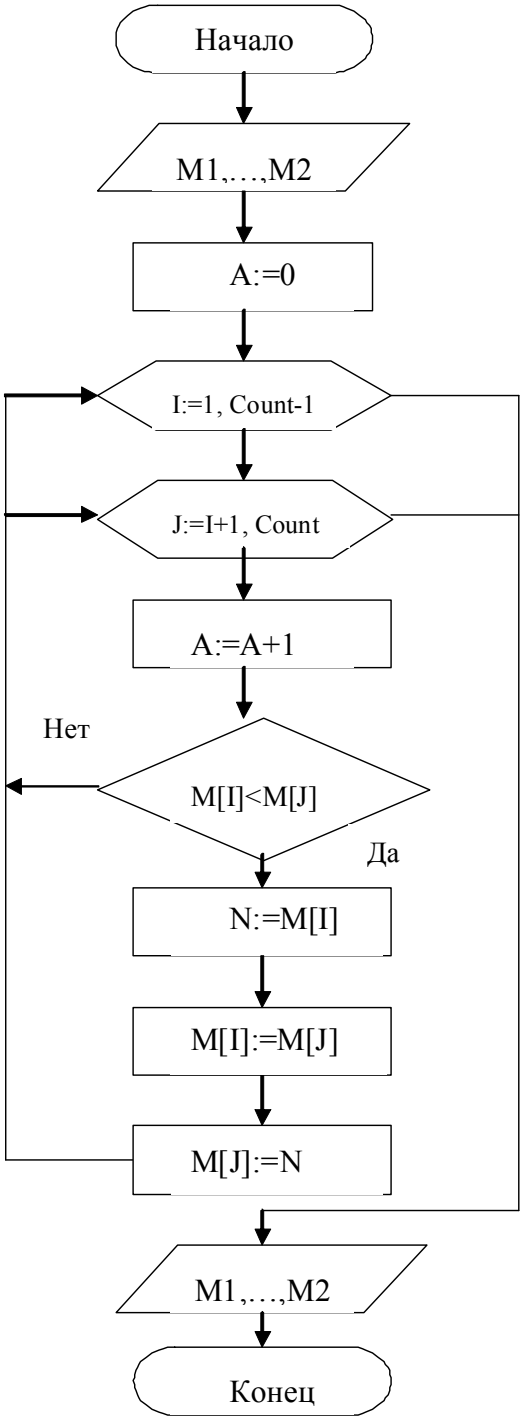


Рис.17. Блок-схема алгоритма линейной сортировки

Введем в разделе описания следующие переменные:

I – для указания позиции первого элемента в рассматриваемой части массива;

J – для указания позиции очередного сравниваемого с ним элемента;

N – для временного хранения значения первого элемента для обмена значениями с максимальным из рассматриваемой части массива;

L – параметр цикла при выводе текущего значения элементов массива в процессе сортировки для наблюдения происходящих в массиве изменений;

A – переменная, значение которой будет равно числу перестановок элементов.

Перед началом сортировки установим значение счетчика итераций *A*, равное 0. Для сортировки организуем два цикла *For*.

Внешний цикл с параметром *I,* указывающим позицию первого элемента в неотсортированной части массива, и внутренний цикл с параметром *J*, указывающим позицию очередного, сравниваемого с первым, элемента неотсортированной части массива.

Сравнение элементов запишем оператором:

```
If M[I] < M[J] Then
     Begin
          N:=M[I];
          M[I]:=M[J];
          M[j]:=N
     End;
```

Если условие M[I]<M[J] выполняется, т.е. в неотсортированной части массива найден элемент, больший, чем первый, то обменять местами эти элементы.

Обмен осуществляется следующим образом: сначала значение *M*[I] запоминается в переменной *N*, затем значение элемента массива из *J* –й позиции записывается в *I* –ю позицию, после чего в *J* –ю записывается значение переменной *N*.

Для наблюдения изменений состояния массива после каждой перестановки зададим цикл вывода значений всех элементов массива и текущее значение числа перестановок элементов *A*.

Пример. Текст программы линейной сортировки массива *M* по невозрастанию может быть записан так:

```
Program Sort_Lin;          {Линейная сортировка по невозрастанию}
Const
     Count = 20;
M:Array[1..20]Of
Byte=(9,11,12,3,19,1,5,17,10,18,,3,19,17,9,12,20,20,19,2,5);
Var
     I, J, N, L : Byte;
     A : Integer;
Begin
  Writeln('Исходный массив');
  For I:=1 To Count Do
     Write(' ', M[I]; Writeln;
     A:=0;
  For I:=1 To Count-1 Do {Изменять размер неотсортированной части массива}
{Сравниваем поочередно I –й элемент нертсортированной части массива со всеми от I+1 –го до конца}
     For J:=I+1 To Count DO
          Begin
          A:=A+1
          If M[I] < M[J] Then
```

{Если в неотсортирован-ной части массива найден элемент, больший, чем I –й, то поменять их местами}

```
                Begin
                    N:=M[I];
                    M[I]:=M[J];
                    M[J]:=N
                End;
                    For L:=1 to Count Do
                    Write(` `,M[L]);
                    Writeln (`Число итераций = `,A)
            End
    End.
```

Операции с матрицами

Матрицей A размерности n × m называется двумерный массив из n строк и m столбцов (рис.18):

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}$$

Рис.18. Матрица общего вида

где a_{ij} $\left(i = 1,2,...,n;\quad j = 1,2,...,m\right)$ — элементы матрицы A.

Наиболее типичные действия производимые над матрицами: сложение матриц, транспонирование матрицы, умножение матриц.

Сложение матриц

Пример программы сложения двух матриц A и B размерности 3×4. Схема решения задачи представлена на рис.19.

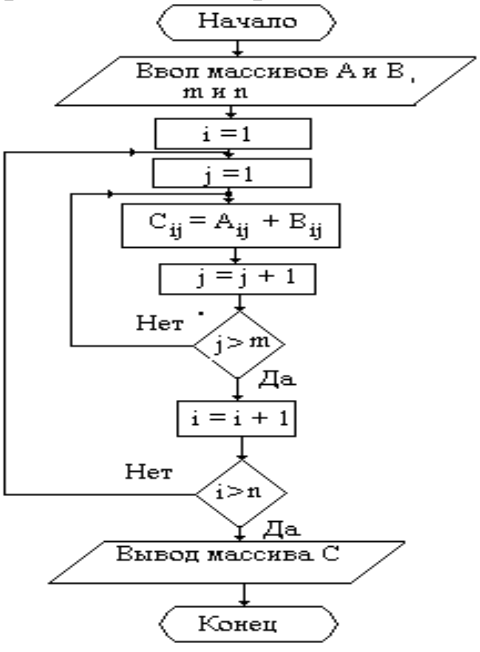


Рис.19. Блок-схема алгоритма сложения матриц.

Текст программы

```
program Summa_matrix;
Const
     m=4;
     n=3;
Type
    matrix = array[1..n,1..m] of Integer;
Var
   A,B,C : matrix;
   i,j : Integer;
BEGIN
     Writeln('Ввод матрицы A');
     for i:= 1 to n do
     begin
          writeln('Строка',i:2);
          for j := 1 to m do
                begin
                     write('Столбец',j:2,'->');
                     readln(A[i,j])
                end;
      end;
          Writeln('Ввод матрицы B');
     for i:= 1 to n do
     begin
          writeln('Строка',i:2);
          for j := 1 to m do
                begin
                     write('Столбец',j:2,'->');
                     readln(B[i,j])
                end;
     end;
     writeln('Матрица C - сумма матриц A и B') ;
     for i:= 1 to n do begin
                          for j := 1 to m do  begin
                                  C[i,j] := A[i,j] + B[i,j]
;
                                  write(C[i,j]:3)
                                                  end ;
                          writeln  ;
                          end
End.
```

Транспонирование матрицы

Пусть дана матрица $A = \{a_{ij}\}$ размерности n × m. Матрица $B = \{b_{ij}\}$ размерности m × n называется транспонированной к матрице A, если ее элементы определены по формуле $b_{ij} = a_{ji}$ $(i = 1,2,...,m;\ i = 1,2,...,n)$. При этом элементы, расположенные на главной диагонали исходной и транспонированной матриц, одни и те же

Рассмотрим **пример** транспонирования матрицы размерности 2 × 3. Блок-схема алгоритма транспонирования матрицы представлена на рис. 20.

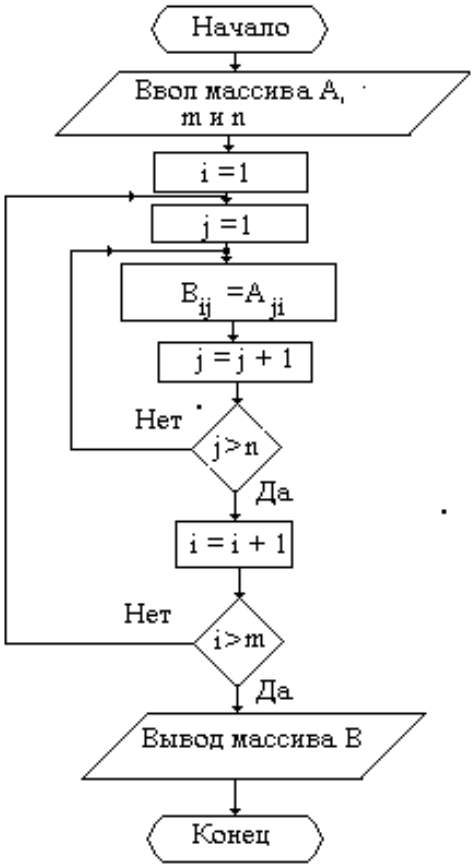


Рис.20. Блок-схема алгоритма транспонирования матрицы.

Текст программы:

```
program Transp_matrix;
Const
     m=3;
     n=2;
Var
   A : array[1..n,1..m] of integer;
   B : array[1..m,1..n] of integer;
   i,j : Integer;
BEGIN
     Writeln('Ввод матрицы A');
     for i:= 1 to n do
     begin
          writeln('Строка',i:2);
          for j := 1 to m do
                begin
                     write('Столбец',j:2,'->');
                     readln(A[i,j])
                end;
      end;
     writeln('Матрица B - Транспонированная к матрице A') ;
     for i:= 1 to m do begin
                          for j := 1 to n do  begin
                                  B[i,j] := A[j,i] ;
                                  write(B[i,j]:3)
                                               end ;
```

```
                writeln ;
                end
End.
```

Умножение матриц

Пусть даны матрица $A = \{a_{ij}\}$ размерности n × m и матрица $B = \{b_{ij}\}$ размерности m × k. Матрица $C = \{c_{ij}\}$ размерности n × k равна произведению матриц A и B, если ее элементы определены по формуле

$$c_{ij} = \sum_{l=1}^{m} a_{il} b_{lj} \quad (i = 1,2,\ldots,n;\ i = 1,2,\ldots,k).$$

Рассмотрим пример умножения матрицы A размерности 2 × 3 на матрицу B размерности 3 × 4. Матрица C, равная произведению матриц A и B будет иметь размерность 2 × 4.

Блок-схема алгоритма умножения матриц A и B представлена на рис. 21.

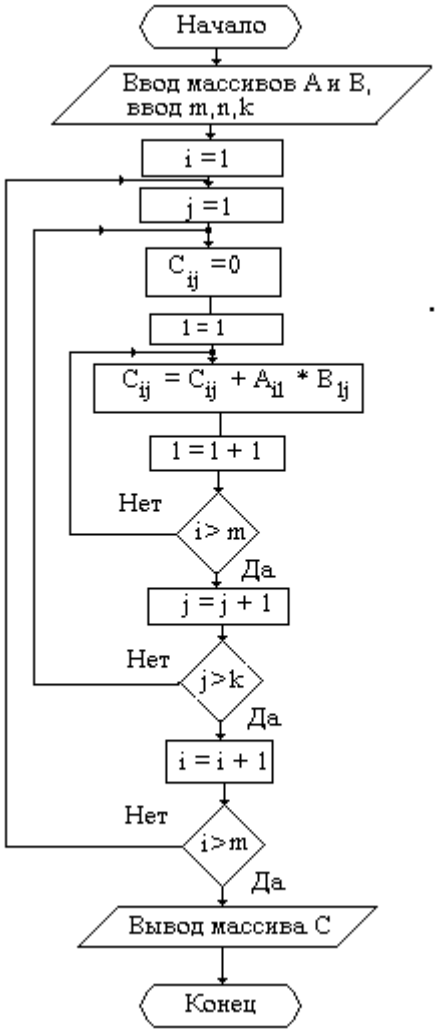


Рис.21. Блок-схема алгоритма умножения матриц.

```
Программа.
program Umnozh_matrix;
Const
     m=3;
```

```
     n=2;
     k=4;
Var
   A : array[1..n,1..m] of integer;
   B : array[1..m,1..k] of integer;
   C : array[1..n,1..k] of integer;
   i,j,l : Integer;
BEGIN
     Writeln('Ввод матрицы A');
     for i:= 1 to n do
     begin
          writeln('Строка',i:2);
          for j := 1 to m do
                begin
                     write('Столбец',j:2,'->');
                     readln(A[i,j])
                end;
      end;
          Writeln('Ввод матрицы B');
     for i:= 1 to m do
     begin
          writeln('Строка',i:2);
          for j := 1 to k do
                begin
                     write('Столбец',j:2,'->');
                     readln(B[i,j])
                end;
     end;
     writeln('Матрица C - умножение матриц A и B') ;
     for i:= 1 to n do begin
                    for j := 1 to k do  begin
                        C[i,j] := 0 ;
                    for l := 1 to m do
                     C[i,j] := C[i,j] +A[i,l] * B[l,j] ;
                             write(C[i,j]:3)
                                          end ;
                       writeln ;
                       end
End.
```

Контрольные вопросы

1. Что такое массив?
2. Как определить местоположение элемента в массиве?
3. Что такое индекс? Каким требованиям он должен удовлетворять?
4. Каким образом задается описание массива, что в нем указывается?
5. Какие основные действия могут выполняться над массивами?
6. Какие основные действия могут выполняться над элементами массива?

ПРОЦЕДУРЫ И ФУНКЦИИ

Необходимость структуризации в программировании

Под *структурным программированием* понимают такие методы разработки и записи программы, которые ориентированы на максимальные удоб-

ства для восприятия и понимания ее человеком. При прочтении программы в ее следующих друг за другом фрагментах должна четко прослеживаться логика ее работы, т. е. не должно быть «скачков» на фрагменты программы, расположенные где-то в другом месте программы.

Структурное программирование – «программирование без **go to**», т.е. не используются операторы перехода без особой необходимости. В связи с этим отдельные фрагменты программы представляют собой некоторые логические (управляющие) структуры, которые определят порядок выполнения содержащихся в них правил обработки данных. Любая программа получается построенной из стандартных логических структур, число типов которых в целом невелико.

Существенная особенность структур *следования*, *ветвления и повторения* – то, что каждая из них имеет только один вход и только один выход, что и обеспечивает логически последовательную структуру программы. Все эти структуры определяются рекурсивно, т. е. каждая из входящих в структуру групп операторов может быть одним оператором, группой операторов и может быть любой из допустимых структур – допускается вложение структур.

Так как программа задает правила обработки данных, то проектирование самих данных при изготовлении программы имеет не менее важное значение, чем проектирование правил их обработки. Очевидно, что, чем четче определены сами данные, тем легче разрабатывать правила их обработки. Простота и надежность программы существенно зависят от того, насколько удобно отдельные обрабатываемые данные объединены в некоторые структуры. При этом язык программирования Паскаль требует от программиста четкого описания вводимой в употребление структуры данных, что позволяет транслятору обеспечивать работу с каждой такой структурой и следить за ее корректностью.

Метод нисходящего проектирования программ

С массовым внедрением вычислительной техники процесс программирования постепенно превращается в промышленное изготовление программ. Для этой цели создаются разнообразные технологии программирования. Примерами таких технологий могут служить технология нисходящего программирования и технология восходящего программирования.

Технология нисходящего программирования базируется на методе программирования «сверху - вниз». Основой такого метода является идея постепенной декомпозиции исходной задачи на ряд подзадач. Сначала формулируется самая общая модель решения, отдельные детали которой на первом этапе могут быть довольно расплывчатыми. По мере разработки программы, разбивая наиболее неясные части алгоритма и добиваясь все более точных и детализированных формулировок, мы получаем все более подробное решение. Решение каждого фрагмента сложной задачи может представлять собой отдельный подпрограммный блок, называемый подпрограммой. Такой процесс детализации продолжается до тех пор, пока не станут ясны все детали решения задачи. В этом случае программу решения сложной задачи можно

представить как иерархическую совокупность самостоятельных фрагментов – *подпрограмм.*

Таким образом, *подпрограммой* называют обособленную, оформленную в виде отдельной синтаксической конструкции и снабженную именем часть программы. Использование подпрограмм позволяет, сосредоточив в них описание определенных операций, в остальной программе только указывать имена подпрограмм, чтобы выполнять эти операции. Такие вызовы подпрограмм возможны неоднократно из разных участков программы, причем при вызове подпрограмме можно передать некоторую информацию, чтобы одна и та же подпрограмма выполняла решение подзадачи для разных случаев.

Подпрограммы в языке TPascal

За наличие подпрограмм как средства структурирования программ язык программирования TPascal называется *процедурно – ориентированным*.

Подпрограммы в TPascal реализованы посредством *процедур* и *функций*. Имея один и тот же смысл и аналогичную структуру, процедуры и функции различаются назначением и способом их использования.

Процедура – это независимая именованная часть программы, которую можно вызвать по имени для выполнения определенных действий. Структура процедуры повторяет структуру программы. Процедура не может выступать как операнд в выражении. Упоминание имени процедуры в тексте программы приводит к активизации процедуры и называется ее вызовом.

Функция аналогична процедуре, но имеются два отличия:

1) функция передает в точку вызова скалярное значение;
2) имя функции может входить в выражение как операнд.

Все процедуры и функции языка TPascal делятся на две группы: *встроенные* (стандартные) и *определенные пользователем*. Первые входят в состав языка и вызываются для выполнения по строго определенному имени. Вторые разрабатываются и именуются самим пользователем. Все стандартные средства расположены в специализированных библиотечных модулях, которые имеют системные имена.

Стандартные библиотечные модули

В систему TPascal версии 7.0 включены восемь модулей: **`System`**, **`Crt`**, **`Dos`**, **`Graph`**, **`Graph3`**, **`Overlay`**, **`Printer`**, **`Turbo3`** и специализированная библиотека **`Turbo Vision`**. Модуль `System` подключается по умолчанию, все остальные должен подключать программист с помощью зарезервированного слова `Uses`. Например, **`Uses Crt, Dos, Printer;`**

Рассмотрим кратко назначение некоторых модулей.

`System` – содержит подпрограммы, обеспечивающие работу всех остальных модулей системы.

`Crt` – содержит средства управления дисплеем и клавиатурой.

`Graph3` – поддерживает использование стандартных графических подпрограмм версии Турбо Паскаль 3.0.

Graph – содержит пакет графических средств, обеспечивающих эффективную работу с адаптерами монитора.

Turbo Vision – библиотека объектно – ориентированных подпрограмм для разработки пользовательских интерфейсов.

Встроенные функции и процедуры

Модуль System подключается к программе автоматически, поэтому его имя не указывается в разделе Uses. По этой причине программе становятся доступны его встроенные процедуры и функции.

Арифметические процедуры и функции

Abs(X:Real):Real-вычисление абсолютной величины X. Тип результата совпадает с типом параметра.

ArcTan(X:Real):Real –вычисляет угол, тангенс которого равен X радиан.

Cos(X:Real):Real – вычисление косинуса X; параметр задает значение угла в радианах.

Exp(X:Real):Real – вычисление экспоненты X.

Frac(X:Real):Real – вычисление дробной части X.

Int(X:Real):Real – вычисление целой части X.

Ln(X:Real):Real – вычисление натурального логарифма X.

Pi:Real – возвращает значение числа Пи.

Sin(X:Real):Real – вычисление синуса X.

Sqr(X:Real):Real – возведение в квадрат значения X.

Sqrt(X:Real):Real – вычисление квадратного корня из X.

Random:Real–генерирует значение случайного числа из диапазона 0…0,99.

Random(I:Word):Word – генерирует значение случайного числа из диапазона 0…I.

Randomize – изменение базы генератора случайных чисел.

Скалярные процедуры и функции

Dec(X{,n}) – процедура уменьшает значение целочисленной переменной X на величину n. При отсутствии необязательного параметра n значение X уменьшается на единицу.

Inc(X{,n}) – процедура увеличивает значение целочисленной переменной X на величину n. При отсутствии необязательного параметра n значение X увеличивается на единицу.

Pred(S) – функция возвращает элемент, предшествующий S в списке значений типа. Тип результата совпадает с типом параметра.

Succ(S) – функция возвращает элемент, следующий за S в списке значений типа. Тип результата совпадает с типом параметра.

Odd(I:Integer):Boolean – возвращает *True*, если I нечетное, и *False*, если I четное.

Функции преобразования типов

Chr(I:Byte) – возвращает символ стандартного кода обмена информацией с номером, равным значению I.

Ord(S):Longint – возвращает порядковый номер значения S в множестве, определенном типом S.

Round(X:Real):Longint – возвращает значение X, округленное до ближайшего целого числа.

Trunc(X:Real):Longint – возвращает ближайшее целое число, меньшее или равное X, если X>=0, и большее или равное X, если X<=0.

Процедуры управления программой

Delay(I:Word) – задержка выполнения программы на I мс.

Exit – выход из выполняемого блока в окружающую среду. Если текущий блок является процедурой или функцией, выход осуществляется во внешний блок.

Halt(N:Word) – прекращение выполнения программы и передача управления системе программирования.

RunError(ErrCode:Word) – прекращение выполнения программы и генерация ошибки времени выполнения.

Вызов стандартной процедуры или функции

Для использования стандартной процедуры или функции к программе подключается тот или иной библиотечный модуль, в котором записана данная процедура или функция (имя модуля указывается в разделе `Uses`). Затем в программе записывается вызов процедуры или функции, для чего записывается ее имя и указываются фактические параметры, например: `Pi`, `Sin(X)`, `Chr(125)`, `Inc(X,5)`. Так как после выполнения функции ее значение присваивается имени, то имя функции используется в выражении.

Процедуры

Описание процедуры включает заголовок (имя) и тело процедуры. Заголовок состоит из зарезервированного слова `Procedure`, идентификатора (имени) процедуры и необязательного, заключенного в круглые скобки, списка формальных параметров с указанием типа каждого параметра. Имя процедуры — идентификатор, уникальный в пределах программы. Тело процедуры представляет собой локальный блок, по структуре аналогичный программе. Общая структура описания процедур и функций иллюстрируется синтаксическими диаграммами, представленными на рис. 22.

Описания меток, констант, типов и т. д. действительны только в пределах данной процедуры. В теле процедуры можно использовать любые глобальные константы и переменные.

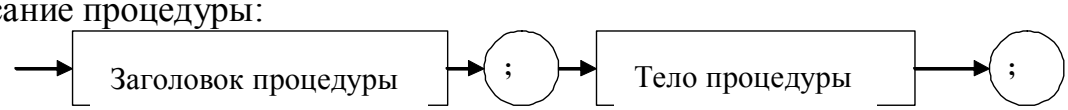
```
Procedure <имя> (Формальные параметры);
Const ...;
Type ... ;
Var   . . . ;
Begin
     <операторы>
End;
```

Пример. Опишем процедуру, которая прерывает выполнение программы и выдает соответствующее сообщение об ошибке:

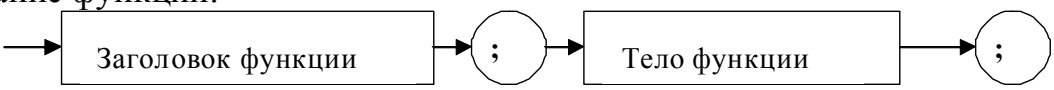
```
procedure A1(Msg: string);
     Begin
          WriteIn(' Ошибка: ', Msg) ;
          Halt(l);
```

```
End;
```

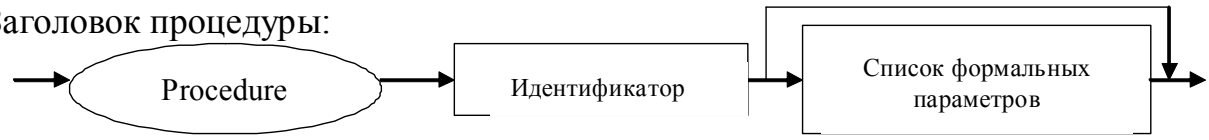
Описание процедуры:



Описание функции:



Заголовок процедуры:



Заголовок функции:

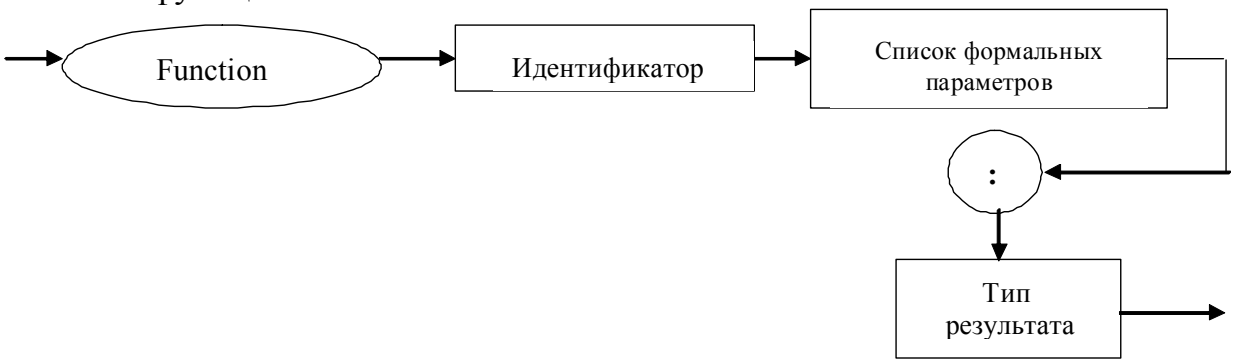


Рис. 22. Структура описания процедур и функций

В данной процедуре пользователя использована переменная `Msg` типа `string`, в которой хранится текст сообщения о характере ошибки, вызвавшей прерывание программы. Для прерывания выполнения программы используется стандартная процедура `Halt` из стандартного библиотечного модуля `System`.

Процедура не может выполниться сама, ее необходимо вызвать по имени и указать фактические параметры того же типа, что и формальные. Количество и тип формальных параметров равны количеству и типу фактических параметров.

Пример. В качестве примера приведем фрагмент программы, в котором используется описанная выше процедура `A1`:

```
program DemoProc;
    {Подсчет суммы десяти введенных целых положительных
    чисел: если будет введено отрицательное число, прервать выполнение}
Const
    Limit = 10; {Ограничение на количество вводимых чисел}
Var
    Count, Item, Sum:integer;
{$1 A1.PAS } {Директива компилятору на включение текста программы
A1.PAS в качестве подпрограммы}
```

```
Begin
    Count:= 0;
    Sum: = 0 ;
While (Count < Limit) do          {Условие выполнение цикла}
     Begin
   Count:= Count+1;
        Write('Введите ', Count, '-е целое число: ');
        Readln(Item) ;
     If Item < 0 then        {Если введено отрицательное число}
     A1('введено отрицательное число*); {Вызов процедуры}
     Sum:= Sum+Item;
     End;
     Writeln(* Сумма введенных чисел равна », Sum);
End.
```

В разделе описания программы описываются константа `Limit`, ограничивающая количество вводимых чисел; в разделе описания переменных описываются переменные `Count, Item, Sum` типа `integer`. Затем в блоке описания записана директива компилятору `{$1 A1.PAS}`, указывающая, что при компиляции данной программы в нее нужно включить в качестве процедуры программу `A1.PAS`.

В начале программы обнуляются значения количества введенных чисел `Count` и их сумма `Sum`. Потом выполняется цикл, пока очередное вводимое число меньше предельного, заданного значением константы `Limit`. Сначала устанавливается номер очередного числа, затем на экран выводится приглашение "Введите 1-е (2-е и т.п.) число", считывается значение числа с клавиатуры в переменную `Item`. Затем проверяется условие `Item<0`.

Если условие выполняется, то вызывается внешняя процедура `A1`, которой передается фактический параметр-значение "введено отрицательное число". Это значение присваивается формальному параметру `Msg` процедуры `A1`. Процедура `Abort` выводит на экран сообщение "Ошибка:" и печатает текст сообщения — значение параметра `Msg` "введено отрицательное число", после чего вызывает стандартную процедуру **`Halt(l)`**, которая прерывает выполнение программы.

Если условие `Item<0` не выполняется, то значение суммы `Sum` увеличивается на значение введенного числа `Item`, и управление передается в заголовок цикла для проверки условия `Count < Limit`. Если условие соблюдается, то тело цикла выполняется еще раз, иначе цикл завершается, а управление в программе передается на оператор, следующий за циклом, т. е. зарезервированным словом `end;`, обозначающим окончание составного оператора в теле цикла. После этого на экран выводится сообщение "Сумма введенных чисел равна" и печатается значение переменной `Sum`. На этом выполнение программы завершается.

Контрольные вопросы

1. Что понимается под структурным программированием?
2. В чем заключается метод нисходящего программирования?
3. Что называется подпрограммой?

4. Что называется параметром и каково его назначение?
5. В чем различие между стандартными и определенными пользователем подпрограммами?

Функции

Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово function, идентификатор (имя) функции, заключенный в круглые скобки, необязательный список формальных параметров и тип возвращаемого функцией значения.

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

```
Function<имя> (Формальные параметры):<тип результата>,
Const      …;
Type       …;
Var
Begin
     <операторы>
End;
```

В разделе операторов должен находиться, по крайней мере один оператор, присваивающий имени функции значение. В точку вызова возвращается результат последнего присваивания.

Обращение к функции осуществляется по имени с необязательным указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке и иметь тот же тип.

В качестве **примера** приведем программу вычисления выражения `Z= (A5+A-3)/2*AM`, в которой возведение в степень выполняется функцией `Step`[1].

```
{Программа вычисления выражения Z= (A^5+A^-3)/2*A^M}
  program DemoFunc;
  Var
       M: integer;
       A, Z, R: real;
  Function Step(N:Integer;X:Real):Real;
  Var
       I : Integer;
       Y : Real;
  Begin
       Y:=1;
       For I:=1 To N Do   {Цикл вычисления N—й степени числа X}
       Y:=Y*X;
       Step:=Y       {Присваивание функции результата вычисления степени}
  End;                       {Конец процедуры-функции}
  Begin                      {Начало основной программы}
  Write(* Введите значение числа A и показатель степени M) ;
  Readln(A,M) ;
  {Вызов функции с передачей ей фактических параметров 5, A}
```

[1] В приведенной программе отсутствует проверка переменной A на ноль. Рекомендуется протестировать программу и вставить в нее проверку для коррекной работы программы.

```
Z:=Step(5,A);
{Вызов функции с передачей ей фактических параметров 3, 1/A}
Z:=Z+Step(3,1/A);
If M=0 Then R:=1
Else
{Вызов функции с передачей ей фактических параметров M, A}
If M>0 then R:=Step(M, A)
{Вызов функции с передачей ей фактических параметров M, A}
Else R:=Step(-M,A);
Z:=Z/(2*R) ;
Writeln('Для A=',A,'14=',M=',' Значение выражения =',Z);
End.
```

В начале программы описываются переменная целого типа *M* и переменные вещественного типа *A, Z, R*, после этого описывается функция вычисления степени числа `Step` с формальными параметрами *N* и *X*, результат, возвращаемый функцией в точку вызова - вещественного типа.

В описании функции вводятся две локальных (местных) переменных *I* и *Y*. Переменная *I* служит для подсчета числа повторений цикла, а в *Y* накапливается значение степени как произведения N одинаковых сомножителей. В заключение функции `Step` присваивается значение вычисленного произведения.

В начале выполнения основной программы на экран выводится запрос «Введите значение числа A и показатель степени M» и считывается с клавиатуры значение вещественного числа *A* и целого числа *M*.

Затем выполняется оператор `Z:=Step(5,A)`. Вначале осуществляется вызов функции *Step* с передачей ей фактических параметров 5, A. Их значения присваивается формальным параметрам функции *N* и *X*. По окончании вычисления степени числа значение функции `Step`, вычисленное для фактических параметров 5 и A, присваивается переменной *Z*. Аналогично в операторе `Z:=Z+Step(3,1/A)` сначала осуществляется вызов функции `Step` с передачей ей фактических параметров 3, 1/A, после чего значение переменной *Z* увеличивается на величину возвращенного в основную программу результата вычисления функции `Step`.

Оператор `if M=0 then R:=1 else if M>0 then R:=Step(M, A) else R:=Step(-M,A)` проверяет условия M=0, M>0 и в зависимости от их соблюдения либо присваивает переменной *R* значение 1 (при M=0), либо выполняет вызов функции `Step` для фактических значений M, A или -M, A, а после вычисления значения функции `Step` присваивает его переменной R. Оператор `Z:=Z/(2*R)` выполняет вычисление значения выражения `Z/(2*R)`, а затем присваивает вычисленное значение переменной Z.

В заключение программы стандартная процедура `Writeln` выводит на экран сообщение о результате вычислений степени M числа A.

Контрольные вопросы

1. В чем состоит сходство и различие подпрограмм-процедур и подпрограмм-функций?
2. Запишите синтаксическую диаграмму определения функции.

3. Опишите последовательность событий при вызове функций.

Механизм передачи параметров

Как было показано в приведенных выше примерах программ с использованием процедур и функций, в заголовке процедуры или функции может быть задан список параметров, которые называются формальными. Название "формальные" эти параметры получили в связи с тем, что в этом списке заданы только имена для обозначения исходных данных и результатов работы процедуры, а при вызове подпрограммы на их место будут подставлены конкретные значения (выражений) и имен. Этот список указывается после имени подпрограммы и заключается в круглые скобки.

Список формальных параметров, указываемых в заголовке подпрограммы, может включать в себя:

• параметры-значения;

• параметры-переменные, перед которыми должно стоять служебное слово var и за которыми указывается их тип;

• параметры-процедуры, перед которыми должно стоять служебное слово `Procedure;`

• параметры-функции, перед которыми должно стоять служебное слово `Function` и после которых указывается тип значения, возвращаемого функцией в основную программу;

• не типизированные параметры, перед которыми должно стоять служебное слово var и отсутствует указание типа.

В списке должны быть перечислены имена формальных параметров и их типы. Имя параметра отделяется от типа двоеточием, а параметры друг от друга - точкой с запятой. Имена параметров одного типа можно объединять в подсписки, в которых имена отделяются друг от друга запятой. Примеры заголовков:

```
Procedure P(Procedure B; Function C:Real;Q,W,R:Char).
Procedure A;
```

Между формальными и фактическими параметрами должно быть полное соответствие:

• формальных и фактических параметров должно быть одинаковое количество

• порядок следования фактических и формальных параметров должен быть один и тот же;

• тип каждого фактического параметра должен совпадать с типом соответствующего формального параметра.

Параметры - значения

Параметры - значения используются только для передачи исходных данных из основной программы в подпрограмму (процедуру или функцию), в списке формальных параметров они перечисляются через запятую с обязательным указанием их типов, как было, например, в выше приведенных примерах:

```
Procedure Abort (Msg: String)
Function Step(N:Integer,X:real):real;
```

Если формальный параметр объявлен как параметр-значение, то фактическим параметром может быть произвольное выражение. При вызове подпрограммы фактические параметры вычисляются и используются как начальные значения формальных параметров, т. е. осуществляется подстановка значений.

Если формальный параметр определен как параметр-значение, то перед вызовом процедуры это значение вычисляется, полученный результат помещается во временную память и передается процедуре. Даже если фактический параметр —простейшее выражение в виде константы или переменной, все равно процедуре будет передана лишь копия этой константы (переменной). В процессе выполнения подпрограммы формальные параметры могут изменяться, но это никак не отразится на соответствующих фактических параметрах-переменных, которые сохранят те значения, которые имели до вызова подпрограммы, так как меняются не они, а их копия. Поэтому параметры-значения нельзя использовать для передачи результатов из подпрограммы в основную программу.

Пример программы с использованием передачи параметров по значению:

```
Program Par_Znach;
Var
A,B : real;
{Процедура вычисления квадратов двух чисел и вывода на экран их суммы}
Procedure Sum_Square(X,Y:real); {X,Y — формальные параметры}
Begin
     X:=X*X;
     Y:=Y*Y;
     Writeln('Сумма квадратов = ', X+Y);
End;
Begin
  A:=1.5
  B:=3.4
{Вызов процедуры Sum_Square с передачей ей значений фактических параметров A и B}
Sum_Square(A,B);
End.
```

При вызове процедуры `Sum_Square` с фактическими параметрами A, B значения этих параметров копируются в соответствующие формальные параметры X, Y и дальнейшие преобразования формальных параметров X, Y внутри процедуры `Sum_Square` уже никак не влияют на значения переменных A, B.

Параметры - переменные

Параметры – переменные используются для определения результатов выполнения процедуры и в списке формальных после зарезервированного слова `Var` с обязательным указанием типа. Каждому формальному параметру, объявленному как параметр – переменная, должен соответствовать фактический параметр в виде переменной соответствующего типа, например:

```
Procedure Example(Var M, N:Integer;VarY:Real);
```

Если формальный параметр определен как параметр – переменная, то при вызове ей передается сама переменная, а не ее копия, и изменение параметра – переменной приводит к изменению фактического параметра в вызывающей программе. Следовательно, исходные данные в процедуру из программы могут передаваться как через параметры – значения, так и через параметры – переменные, а результаты работы процедуры возвращаются в вызывающую программу только через параметры – переменные.

Пример программы, использующей параметры-переменные:

```
Program Sum_Sub_Square;
Var
     A,B:Real;
     SumAB,SubAB:Real;
Begin
     Sum:=X*X+Y*Y;
     Sub:=X*X-Y*Y;
{Процедура с параметрами – переменными Sum, Sub}
Procedure Sum_Sub(X, Y : Real; Var Sum, Sub : Real);
Begin
   Sum:=X*X + Y*Y;
   Sub:=X*X-Y*Y;
End;
   Begin
     A:=1.5;
     B:=3.4;
   Writeln('Сумма квадратов чисел ',A, ' и ', B, ' =', SumAB);
   Writeln('Разность квадратов чисел ',A,'и ',B,' =',SubAB');
   End.
```

Область действия параметров

При создании программ, использующих процедуры, следует учитывать, что все объекты (метки, константы, типы, переменные, процедуры и функции), которые описываются после заголовка процедуры, называются *локальными объектами* и доступны только в пределах этой процедуры, но недоступны вызывающей программе. Все эти объекты создаются при входе в процедуру и уничтожаются при выходе из нее. Если одно и то же имя определено в нескольких процедурах, вызываемых одной и той же программой, то в каждой процедуре этому имени соответствует свой локальный объект.

Все объекты, описанные в вызывающей программе, называются *глобальными* и являются доступными внутри процедур, вызываемых этой программой. Поэтому обмен данными между программой и вызываемой ею процедурой может производиться и через глобальные переменные. Если одно и то же имя определено и в программе, и в вызываемой ею процедуре, то ему соответствует глобальный объект, но внутри процедуры глобальный объект недоступен, он как бы экранируется (маскируется) локальным объектом с таким же именем.

В TPascal допускается вложенность процедур и функций. Процедура, описанная в основной программе, в свою очередь, может иметь описания внутренних процедур и функций и т. д. При этом объекты, описанные в вы-

зывающей процедуре, являются глобальными по отношению к вызываемой процедуре.

Можно схематически изобразить структуру блоков некоторой программы, как показано на рис.23.

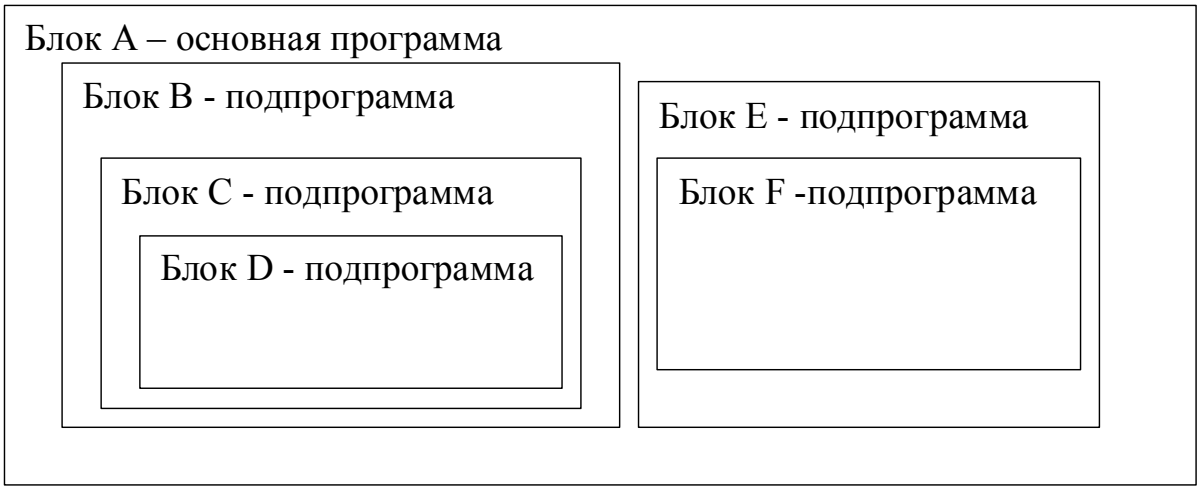


Рис.23. Структура блоков программы

Для доступа к объектам, описанным в различных блоках, требуется соблюдать следующие правила:

1. Имена объектов, описанных в некотором блоке, считаются известными в пределах данного блока, включая и все вложенные блоки.

2. Имена объектов, описанных в блоке, должны быть уникальны в пределах данного блока и могут совпадать с именами объектов из других блоков.

3. Если в некотором блоке описан объект, имя которого совпадает-с именем объекта, описанного в объемлющем блоке, то это последнее имя становится недоступным в данном блоке (оно как бы экранируется одноименным объектом данного блока).

Если применить эти правила к предыдущей схеме, можно сказать, что объекты, описанные в блоке B, известны (видимы), кроме самого блока B еще и блоках C и D, но невидимы в блоке A. Объекты, описанные в блоке F, известны только в пределах этого блока.

Пример

```
Program Exampl_proc;
Procedure P;
Procedure A;
    Var
     J:Integer;
   {Локальная переменная J является глобальной по отношению к процедуре B}
Procedure B;
    Var
     J:Integer;
{Локальная переменная J экранирует глобальную переменную J, описанную в вызывающей процедуре A}
Begin
     Writeln(J) ;
```

```
End;
     Begin
     J:=1;
     B;                    {Вызов процедуры B}
     End;
     Begin
     A;                    {Вызов процедуры A}
     End.
```

В качестве примера с вложенными подпрограммами рассмотрим пример программы, определяющей количество сверхпростых чисел в натуральном ряду чисел, не превышающих 1000. Сверхпростым называется число, если оно простое и число, полученное из исходного числа, при записи цифр исходного числа в обратном порядке (перевертыш) тоже будет простым. Например, 13 и 31 — сверхпростые числа.

Пример

```
{Подсчет количества сверхпростых чисел < 1000}
Program Sverx__Prost;
Var
     X,K:Integer;          {Описание глобальных переменных X, K}
{Функция проверки числа X на принадлежность к множеству простых чисел}
Function Prost(Y:Integer):Boolean;
Var
{Описание локальных переменных D, I, Flag}
     D,I:Integer;
     Flag : Boolean;
Begin
     D:=0;
     Flag:=False;
     For I: =2 To Y-1 Do
     If Y mod 1=0 Then
     D:==D+1;
     If D=0 Then
     Flag:=True;
     Prost:=Flag;
{Нашли простое число, имеющее только два делителя: единицу и само это число}
End;
{Перевертыш простого числа}
Function Povorot(Y:Integer):Integer ;
Var
     S:Integer;            {Описание локальной переменной S}
Begin
     If Y<10 Then
          S:=Y;
     If (Y>10) and (Y<100) Then
          S:=Y mod 10*10+Y div 10;
     If (Y>==100) and (Y<1000)Then
          S:=Y mod10*100+Y mod100 div 10*10+Y div 100;
          Povorot:=S;
End;
{Начало основной программы}
Begin
```

```
     Writein(' ':20,'Сверхпростые числа ') ;
     K:=2;                {2 и 3 — простые числа}
     For X:=4 To 999 Do
          Begin
```

{Вызов функции определения простого числа}

```
          If Prost(X) Then
          If Prost(Povorot(X)) Then
```

{Вызов функции определения простого числа для числа-перевертыша, полученного вычислением функции Povorot(x) от простого числа}

```
            Begin
               Writeln(X); {Печать сверхпростого числа} K:=K+1;
            End;
          End;
          Writeln('Всего найдено ',K,' сверхпростых чисел < 1000');
End.
```

Использование функций и процедур является естественным дополнением к технике программирования, позволяющим создавать качественные программы.

Контрольные вопросы

1. Каковы отличия параметров – значений от параметров – переменных, особенности их описания и применения.
2. Чем отличаются локальные и глобальные параметры? Какова область их действия?

ФАЙЛЫ

Общие сведения

Файлом называется совокупность данных, записанная во внешней памяти под определенным именем. Файловый тип данных или файл определяет упорядоченную совокупность произвольного числа однотипных компонент.

Целесообразность применения файлов диктуется следующими причинами.

1. Ввод больших объемов данных, подлежащих обработке, утомителен и требует большого времени. Гораздо удобнее создать отдельный файл данных, который может быть подготовлен заранее и, самое главное, применяться неоднократно.

2. Файл данных может быть подготовлен другой программой, становясь, таким образом, связующим звеном между двумя разными задачами, а также средством связи программы с внешней средой.

3. Программа, использующая данные из файла, не требует присутствия пользователя в момент фактического исполнения.

Описания файлового типа

Любой файл имеет три характерные особенности. Во-первых, у него есть имя, что дает возможность программе работать одновременно с несколькими файлами. Во-вторых, он содержит компоненты одного типа. Таким компонентом может быть любой тип, кроме файлового. Например, допускается файл записей, но нельзя создать «файл файлов». В-третьих, длина вновь создаваемого файла никак не оговаривается при его создании и огра-

ничивается только емкостью устройств внешней памяти. Все это позволяет считать файлы одной из наиболее фундаментальных структур данных.

В большинстве случаев файлы состоят из текстовых строк, или записей. Для описания файла используется словосочетание **File Of.** Синтаксическая диаграмма для файловых типов выглядит так (рис.24):

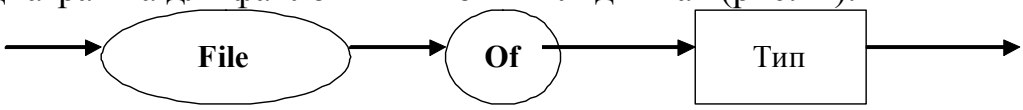


Рис.24. Синтаксическая диаграмма файловых типов

Для доступа к файлу описывается специальная файловая переменная, которая считается представителем файлов в программе. Если файл состоит из записей, дополнительно описывается переменная для доступа к полям записи (обозначим ее R).

Формат:

```
Type
     <имя типа> = <тип компонентов>;
Var
     <F> : file of <имя типа>;
     <R> : <имя типа>;
```

Файл можно представить как потенциально бесконечный список значений одного и того же (базового) типа. Все элементы файла считаются пронумерованными, начальный элемент имеет нулевой номер.

В любой момент времени программе доступен только один элемент файла, на который ссылается **текущий указатель** (указатель обработки). Часто позицию размещения доступного элемента называют **текущей позицией.**

Как правило, все действия с файлом (чтение из файла, запись в файл) производятся поэлементно, причем в этих действиях участвует тот элемент файла, который обозначается текущим указателем. В результате совершения операций текущий указатель может перемещаться, настраиваясь на тот или иной элемент файла.

По способу доступа к элементам различают файлы последовательного и прямого доступа.

Файлом последовательного доступа называется файл, к элементам которого обеспечивается доступ в такой же последовательности, в какой они записывались.

Файлом прямого доступа называется файл, доступ к элементам которого осуществляется по адресу элемента.

Например, для поиска нужного элемента в последовательном файле необходимо, начиная с нулевого перемещать указатель обработки до тех пор, пока он не будет указывать на искомый элемент, а при поиске нужного элемента в файле прямого доступа достаточно указать номер его позиции. При организации данных в файле последовательного доступа нельзя одновременно читать данные из файла и записывать данные в файл, так как для чтения некоторого элемента последовательного файла указатель обработки помещен на данный элемент, а для записи нового элемента этот указатель одновременно должен быть в конце файла.

Компилятор TPascal поддерживает три типа файлов: *текстовые, типизированные, нетипизированные.*

Средства обработки файлов

Файловая система на TPascal наиболее полно использует возможности операционной системы по передаче данных. Каждому файлу в языке ставится в соответствие файловая переменная определенного типа, поэтому перед началом работы с файлом необходимо установить данное соответствие. Для этого в языке используется процедура

```
Assign (Var F; Name: string);
```

где `F` — переменная любого файлового типа, а строковое выражение `Name` содержит полное имя файла, удовлетворяющее требованиям операционной системы

Процедура *Assign* всегда предшествует другим процедурам работы с файлам, так как ставит в соответствие конкретному файлу на внешнем устройстве логическую файловую переменную языка, к которой впоследствии будут обращаться другие файловые процедуры. Недопустимо использование процедуры *Assign* для уже открытого файла. Это значит, что если было назначено имя конкретного набора данных файловой переменной с помощью процедуры Assign, а затем этот файл был открыт, то, прежде чем использовать ту же файловую переменную для нового набора данных, необходимо с помощью процедуры *Close* закрыть этот файл.

1). Для работы с файлом, прежде всего, необходимо его открыть. В языке TPascal предусмотрены для этого две процедуры:

`Reset(``VarF:File);` — открывает существующий файл;

`Rewrite``(VarF:File);` — создает и открывает новый файл.

При описании обеих процедур параметр `File` означает файловую переменную любого типа. Открытие внешнего файла с помощью процедуры `Reset` в случае его отсутствия на диске может привести к ошибке при выполнении программы. Подобные ошибочные ситуации в операциях ввода-вывода позволяет отслеживать специальная функция `lOresult`.

Пример. Стандартное открытие файла.

```
Assign(F, ' ');
Reset(F) ;
```

При назначении файловой переменной пустой строки происходит автоматическая ссылка на стандартный файл ввода, которому в модуле `system` соответствует устройство `CON`. С открытием такого файла появляется возможность ввода данных с клавиатуры.

2). Процедура *`Rewrite`* создает и открывает новый файл. Использование этой процедуры требует особого внимания. *При попытке создать и открыть новый файл с именем уже существующего на диске набора данных действие процедуры* **`Rewrite`** *сведется к удалению этого набора и созданию нового пустого файла с тем же именем.*

Если процедура *Rewrite* используется для текстового файла, то к открываемому набору данных в текущем сеансе открытия файла могут быть применимы только операции записи.

3). Операция закрытия файла является логическим окончанием работы с любым открытым файлом. Для этого служит процедура `Close (Var F) ;`

Использование процедуры `Close` позволяет устранить связь файловой переменной с внешним файлом, установленную ранее с помощью процедуры `Assign.`

Пример. Полная цепочка команд для создания простого текстового файла с именем `WORK.TXT`:

```
Var
     F: Text;
     Begin
          Assign (F, 'WORK. TXT ') ;
          Rewrite (F);
          Write(F, 'Простой текстовый файл');
          Close (F) ;
     End.
```

К языковым средствам обслуживания файлов необходимо отнести процедуры переименования и удаления неоткрытых файлов. Использование этих процедур не зависит от типа файла.

```
Rename(VarF; NewName:String);
```

Процедура переименовывает неоткрытый файл F любого типа. Новое имя задается строкой `NewName.`

```
Erase(var F);
```

Процедура удаляет *неоткрытый* внешний файл любого типа, задаваемый переменной F.

Обе процедуры *нельзя использовать для уже открытых файлов*. В противном случае могут возникнуть нежелательные последствия со стороны операционной системы. Единственным шагом перед использованием процедур, является установка связи между внешним файлом с конкретным именем и файловой переменной. Операции удаления и переименования осуществляются только для *реально существующих* файлов, иначе возникает ошибка выполнения программы.

Пример: Удаление или переименование файла.

```
Var
     F: File;
     Ch: Char;
     St: String;
Begin
     Write('Введите имя файла: ');
     Readln(St);                          {Чтение имени})
    Assign(F,St);                    {Назначить имя файловой переменной}
    Write('Удалить файл  (У), Переименовать(П), Выход(В)');
    Readin(Ch) ;
     Case Ch Of
     'У','y':Erase (F);                   {Удаление файла}
```

```
    'П','п':Begin
              Write('Введите новое имя файла: ');
              Readin(St)
              Rename(F,St);              {Переименование файла}
       End;
    'В','в':Halt(1) ;
  End;                    {Case}
End.
```

В приведенном примере альтернативный выбор тех или иных действий зависит целиком от того, что будет введено с клавиатуры. Этот вариант программы не позволяет обработать ошибочные ситуации в случае, если файла с именем *St* не существует на диске.

Текстовые файлы

Текстовый файл можно рассматривать как последовательность символов, разбитую на строки длиной от 0 до 256 символов. Для описания используется стандартный тип `Text`:

```
Var
    F:text;   {F – файловая переменная}
```

Каждая строка завершается маркером конца строки. На практике такой маркер представляет собой последовательность из двух символов: перевод строки `Chr(13)` и возврат каретки `Chr(10)`. Эти два символа задают стандартные действия по управлению текстовыми файлами. Стандартно открываемые предопределенные файлы `Input` и `OuTPascal ut` в модуле `System` имеют тип `Text`.

У текстовых файлов своя специфика. Специальные расширения стандартных процедур чтения `Read` и записи `Write` разрешают работать со значениями несимвольного типа. Другими словами, последовательность символов автоматически преобразуется к значению того типа переменной, которая используется в файловых операциях.

1). Вызов `Read(F,Ww)`, где `Ww` — переменная типа Word, осуществляет чтение из файла F последовательности цифр, которая затем интерпретируется в число, значение которого и будет присвоено переменной `Ww`. В случае если вместо последовательности цифр идет любая другая последовательность символов, использование такого оператора приводит к ошибке выполнения программы.

Открытие текстового файла можно произвести двумя стандартными способами:

- поставить в соответствие файловой переменной имя файла (процедура `Assign`), открыть новый текстовый файл (процедура `Rewrite`);
- поставить в соответствие файловой переменной имя файла (процедура `Assign`), открыть уже существующий файл (процедура `Reset`).

2). Текстовый файл в силу свой специфики во время работы допускает какой – либо один вид операции: чтение или запись. В связи с этим для работы с текстовыми файлами появляется еще одна процедура открытия файла:

```
Append(VarF:Text);
```

Эта процедура открывает уже существующий файл и позиционирует указатель обработки на конец файла. После такого открытия текстовый файл можно только дополнять информацией, начиная с конца последней строки. Ограничения, накладываемые на процедуру *Append*, такие же, как у процедур `Reset` и `Rewrite`.

3). Для обработки текстовых файлов используются процедуры `Read` и `Write`, обеспечивающие соответственно чтение и запись одной строки в текстовый файл. Использование специальных разделителей строк файла позволило ввести в состав языковых средств процедуры: `Readln`, обеспечивающую те же действия, что и `Read`, и дополнительно – чтение до маркера конца строки и переход к новой строке; `Writeln`, обеспечивающую запись всех величин с обязательной установкой маркера конца строки в файл. Общий вид представления процедур следующий:

```
Readln(VarF:Text;V1,[V2, … Vn]);
Writeln(VarF:Text;V1,[V2, … Vn]);
```

где V1…Vn переменные разных типов.

Процедура `Read` обеспечивает ввод данных общим потоком из одной строки, а `Readln` приводит к обязательному переходу к следующей строке текстового файла, т. е. ввод данных осуществляется из различных строк. Все выше сказанное в равной мере отностся к операциям записи с помощью процедур `Write` и `Writeln`.

4). При организации ввода – вывода используются специальные языковые средства в виде функций `Eoln, Eof, SeekEoln, SeekEof`.

Функция `Eoln` возвращает булевское значение *True*, если текущая файловая позиция находится на маркере конца строки или вызов `Eof(F)` возвратил значение *True*. Во всех других случаях значение функции будет *False.*

Функция `Eof(VarF:Text)` возвращает булевское значение *True*, если указатель конца файла находится сразу за последним компонентом, и *False* – в противном случае.

Пример: Просчитать последовательность длиной шесть символов из первой строки текстового файла EXAMPLES.PAS.

```
Var
     F: Text;
     St: string[6];
Begin
      Assign(F,'EXAMPLE PAS');{файл EXAMPLE PAS должен существовать}
Reset(F);
While not Eoln(F) Do
     Begin                              {Проверка конца строки}
          Read(F,St:);
          Writeln('St = ', St);          {Вывод на экран)
     End;
     Readin(F);                         {Переход к следующей строке)
     Close(F) ;
End.
```

Функция `SeekEoln(VarF:Text)` возвращает булевское значение *True* при достижении маркера конца строки, причем указатель файла пропускает все пробелы и знаки табуляции, предшествующие маркеру. В противном случае функция возвращает значение *False*.

Функция `SeekEof(VarF:Text)` возвращает значение *True*, если указатель файла находится на маркере конца файла. Эта функция также пропускает все пробелы и знаки табуляции, предшествующие маркеру, и выполняет автоматический пропуск маркера конца строки. Характерным примером использования этих функций может служить чтение числовых величин из текстового файла, когда необходимо пропустить обработку разделяющих эти числа пробелов или знаков табуляции.

Пример 1: Составим программу, обеспечивающую создание на диске текстового файла и запись в него текста.

В разделе описания опишем тип `Fil:TypeFil=Text`. В разделе описания переменных запишем описания следующих переменных: `F1` - переменная типа `Fil`, которая будет выполнять функции файловой переменной, т. е. представлять текстовый файл в программе; `Name` - переменная типа `String`, которая будет принимать значение имени создаваемого файла (для записи до 8 символов имени, точки и 3 символов расширения требуется строка длиной 12 символов); `Txt` – переменная строкового типа, которая будет принимать значение вводимой с клавиатуры последовательности символов.

Блок описания программы будет записан так:

```
Type
     fil = text;
Var
     F1:Fil;
     Name:string[12] ;
     Xt:string
```

В начале программы выведем запрос имени файла и считаем его значение в переменную `Name`, затем, используя стандартную процедуру **`Assign`**, поставим в соответствие файлу с введеным именем переменную `F1` и откроем вновь созданный файл процедурой `Rewrite`. Этот фрагмент программы запишем таким образом:

```
Write('Введите имя файла для записи текста >');
Radln();
Writeln;
Assign(F1, Name);
Rewrite(F1)
```

После открытия файла запишем приглашение на ввод текста для записи в файл. Считывание текста по строкам запишем оператором цикла `Repeat`, завершение ввода текста зададим условием `Until Txt= ''`. как только вместо строки символов будет введен пробел, т. е. нажата клавиша Enter, цикл завершится и файл закроется. Это можно записать следующим образом:

```
Writeln('Вводите текст для записи(для окончания на-
жмите Enter):');
     Writeln;
     Repeat
```

```
        Write(':>');
    Readln(Txt);
    Until Txt = '';
    Close(F1);
```

После завершения ввода текста в файл выведем на экран сообщение: «Ввод окончен, нажмите Enter». Для того, чтобы окно исполнения программы (экран пользователя) не закрывалось, запишем процедуру `Readln`, которая будет ожидать нажатия клавиши *Enter*.

Полный текст программы записи текстового файла на диск буде таков:

```
Program Write_txt_file; { Запись текстового файла на диск}
Type
     Fil = Text;
Var
     F1:Fil;
     Name:string[12];
     Txt : string;
Begin
     Write('Введите имя файла для записи текста >');
     Readln (Name);
     Writeln;
     Assign (F1,Name);
     Rewrite(F1);
     Writeln('Вводите текст для записи (для окончания нажмите
Enter):') ;
     Writeln;
     Repeat
     Write(':>') ;
     Readln(Txt) ;
     Writeln(Fl,Txt) ;
     Until Txt=' ' ;
     Close(Fl) ;
     Writeln;
     Writeln('Ввод окончен, нажмите Enter');
     Readin;
End.
```

Контрольные вопросы

1. Что такое файл? Для каких целей используются файлы?
2. Какими причинами диктуется целесообразность применения файлов?
3. Зачем используется специальная файловая переменная? Как устанавливается соответствие файловой переменной файлу во внешней памяти?

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Лабораторная работа 1. Программирование линейных и разветвляющихся вычислительных процессов

Цель работы - освоить на практике программирование несложных линейных и разветвляющихся вычислительных процессов, используя для этого простейшие операторы ввода - вывода, присваивания, условные, научиться отладке программ и подготовке с этой целью тестов.

Варианты заданий

Задача 1. Составить программу для вычисления и печати указанных величин при заданных значениях исходных величин (табл. 6), причём каждое подвыражение должно вычисляться по одному разу. Для обеспечения последнего условия ввести в случае необходимости промежуточные переменные.

Таблица 6

№ вар.	*Формула*	*Исходные данные*	*Печатаемые величины*
1	$y = \frac{e^{\sin x}(x\cos^3 x - \sin x)}{\cos^2 x}$	$x = -0{,}13$	x, $\sin x$, $\cos x$, y
2	$b = (t + \frac{1}{t})e^{-at^2}$; $c = (t + \frac{1}{t})e^{at^2}$	$t = 0{,}9$; $a = 0{,}21$	a,t,b,c
3	$y = 2 \cdot 10^{-3} x(1 + \ln x + \ln^2 x)$	$x = 1{,}5$	x, $\ln x$, y
4	$y = \frac{1}{2}(x^2 - \frac{1}{2})\sin^2 x + \frac{1}{4}\sqrt{1 - x^2}$	$x = 0{,}17$	x, $\sin x$, $\sqrt{1 - x^2}$, y
5	$s = \sqrt{3}e^{-x}(1 + \frac{x}{1+x^2})$; $t = \sqrt{3}(1 + \frac{x}{1+x^2})$	$x = 0{,}3$	x, $1 + \frac{x}{1+x^2}$, e^{-x}, s, t
6	$p = \frac{3t}{\ln^2 t}(\ln t - 1)^2$	$t = 2{,}85$	t, $\ln t$, p
7	$y = \lvert x - \ln(1 + x^2)\rvert$; $z = \frac{\ln(1+x^2)}{e^x}$	$x = 0{,}41$	x, $\ln(1 + x^2)$, y, z
8	$t = (1 + \mathrm{tg}^2 ax)e^{-x}$; $s = \frac{\mathrm{tg}\, ax}{e^x}$	$a = 0{,}8$; $x = 0{,}36$	a, x, $\mathrm{tg}\, ax$, e^x, t, s
9	$a = \frac{2 \cdot 10^{-2} x}{1 + \sin^2 x}$; $b = \frac{2 \cdot 10^{-2} e^x}{1 + \sin^2 x}$	$x = 0{,}11$	x, $1 + \sin^2 x$, e^x, a, b
10	$y = (2x)^{\frac{1}{2}} \sin x + e^{1 - \sqrt{2x}}$	$x = 0{,}62$	x, $\sqrt{2x}$, y

Задача 2. Составить программу для вычисления и печати указанных величин при заданных значениях исходных величин (табл. 7).

Таблица 7

№ вариан-та	*Формула*	*Исходные данные*	*Печатаемые величины*
1.	$y = \arcsin\left(x^2\right) + \sqrt[3]{x^2+1}$	$x = 0{,}8$	*x, y*
2.	$z = \left\lvert x^{\frac{a}{b}} - \sqrt[3]{\frac{a}{b}} \right\rvert$	$a = 1{,}8$ $b = 5{,}6$	*a,, b, z*
3.	$y = e^{-bt} \cdot \sin(bt+a) - \sqrt[3]{\lvert bt+a \rvert}$	a = -0,6 b = 1,7 t = 0,73	*a, b, t, y*
4.	$s = \arccos\left(\frac{x}{b}\right) + \left(\frac{b}{x} + \sin x\right)^2$	$x = 2{,}8$ $b = 4{,}1$	*x, b, s*
5.	$y = \cos^2(ax)^3 - \frac{x}{\sqrt{x^2+ax}}$	x = -2,9 $a = 0{,}7$	*x, a, y*
6.	$y = \frac{1}{3}(\sin x - x)^2 + \log_2(\sin x - x)^2$	$x = 0{,}56$	$x, y, (\sin x - x)^2$
7.	$y = \arcsin^2\left(\frac{x}{a}\right)$	$x = 2{,}41$ $a = 4{,}5$	*x, y*
8.	$z = x\left(1+\sin^2\frac{p}{x}\right) + \lg\left(1+\sin^2\frac{p}{x}\right)$	$p = 1{,}42$ $x = 0{,}5$	$x, p, z, \left(1+\sin^2\frac{p}{x}\right)$
9.	$y = \frac{x}{8\left(x^2+1\right)} - \frac{1}{16} arctg^2\frac{x}{8}$	$x = 12$	*x, y*
10.	$z = \sqrt[3]{x+1} + \cos(x+1)$	$x = 2{,}5$	$x, \sqrt[3]{x+1}, z$

Задача 3. Составить программу вычисления указанных величин при произвольных значениях исходных величин.

1. $y = \begin{cases} \sin(x-3), & \text{если } \lvert x-3 \rvert < 4 \\ \sin\left(\frac{1}{x-3}\right), & \text{если } \lvert x-3 \rvert \ge 4 \end{cases}$

2. $v = \begin{cases} (t+\sin t)^2, & \text{если } \sin t < \cos t \\ (t+\cos t)^2, & \text{если } \sin t \ge \cos t \end{cases}$

3. $Q = x^2 + y^2 + \begin{cases} x^3, & \text{если } x > y \\ y^3, & \text{если } x \le y \end{cases}$

4. $t = \begin{cases} \frac{a}{2} e^{\lvert 1-at \rvert}, & \text{если } a > t \\ \frac{a}{2} e^{\sqrt{\lvert 1-at \rvert}}, & \text{если } a \le t \end{cases}$

5. $z = \begin{cases} x^2 + y^2, & \text{если } x^2 < y^2 \\ \sqrt{x^2+y^2}, & \text{если } x^2 \ge y^2 \end{cases}$

6. $y = \begin{cases} \sin^2 3x, & \text{если } 3x < 2 \\ \frac{1}{\sin^2 3x + 4{,}2}, & \text{если } 3x \ge 2 \end{cases}$

7. $v = \begin{cases} 4 + t^3, & \text{если } \lvert t \rvert \le 2 \\ \frac{1}{4+t^3}, & \text{если } \lvert t \rvert > 2 \end{cases}$

8. $y = \begin{cases} t^2 + 2, & \text{если } t > 2 \\ \sin(t^2+3), & \text{если } t \le 2 \end{cases}$

9. $y = \begin{cases} \sin(x^2+y^2), & \text{если } x < y \\ \cos(x^2+y^2), & \text{если } x \ge y \end{cases}$

10. $z = \begin{cases} \frac{1-\sin^3 x}{x}, & \text{если } x \ne 0 \\ 0.29, & \text{если } x = 0 \end{cases}$

Задача 4. Составить программу вычисления указанной величины при произвольных значениях исходных величин .

1) $y = \begin{cases} x^2 e^x, & \text{если } x \le 0 \\ x + \ln(1+x), & \text{если } 0 < x < 1 \\ 3^x, & \text{если } x \ge 1 \end{cases}$

2) $z = \frac{1}{4}\sin x + \begin{cases} 10 + x, & \text{если } x < 0 \\ 2 + x^2, & \text{если } 0 \le x \le 2 \\ 2x, & \text{если } x > 2 \end{cases}$

3) $r = \begin{cases} \operatorname{arctg}\frac{y}{x}, & \text{если } x > 2 \\ y - x, & \text{если } x \le 2 \text{ и } y > 1 \\ \frac{x}{1+y^2}, & \text{если } x \le 2 \text{ и } y \le 1 \end{cases}$

4) $y = \begin{cases} \frac{a}{b}\sqrt{a^2+b^2}, & b > 1 \\ -a, & b \le 1 \text{ и } a^2 + b^2 < 3 \\ -b, & b \le 1 \text{ и } a^2 + b^2 \ge 3 \end{cases}$

5) $z = \begin{cases} (x^2+1)e^x, & \text{если } |x| \le 1 \\ \frac{|x|}{x^2+1}, & \text{если } 1 < |x| < 2 \\ 1 + x + x^2, & \text{если } |x| \ge 2 \end{cases}$

6) $s = \begin{cases} x - y, & x > y \\ x^2 + y^2, & x \le y \text{ и } x^2 + y^2 < 1 \\ \frac{x}{x^2+y^2}, & x \le y \text{ и } x^2 + y^2 \ge 1 \end{cases}$

7) $y = \begin{cases} x\sin^2 x, & \text{если } \sin x < 0 \\ 0{,}5x, & \text{если } 0 \le \sin x < 0{,}5 \\ e^{\sin x}, & \text{если } \sin x \ge 0{,}5 \end{cases}$

8) $k = \begin{cases} e^x|x|, & \text{если } x < 1 \\ 3x, & \text{если } 1 \le x \le 2 \\ \frac{x}{x+5}, & \text{если } x > 2 \end{cases}$

9) $y = \frac{x^3(2+x)}{x^2+1} + \begin{cases} 4 + x, & \text{если } x < 1 \\ 2x, & \text{если } 1 \le x \le 2 \\ x, & \text{если } x > 2 \end{cases}$

10) $z = \begin{cases} x + 2y, & \text{если } x < y \\ 2x + y, & \text{если } x \ge y \text{ и } x < 4 \\ y, & \text{если } x \ge y \text{ и } x \ge 4 \end{cases}$

Задача 5. Составить программу для вычисления наибольшего или наименьшего из двух указанных выражений при произвольных значениях исходных величин, причем каждое выражение вычислять не более одного раза.

Таблица 8

№ варианта	*Формула*
1	$y = \max\left\{\frac{x}{1+e^{-x}};\ \frac{e^{-x}}{1+x^2}\right\}$
2	$r = \min\left\{\frac{\ln\alpha}{1+\alpha};\ \frac{\alpha^2}{1-\ln\alpha}\right\}$
3	$u = \max\left\{\frac{x\ln x}{x^2+\ln^2 x};\ \frac{x}{1+x^2\ln^2 x}\right\}$
4	$t = \min\left\{\frac{r-3}{r+e^r};\ \frac{e^r - 3e^{-r}}{r}\right\}$
5	$z = \max\left\{\frac{\ln x}{\sqrt{1+x^2}};\ \frac{2\ln x}{1+\sqrt{x}}\right\}$

№ варианта	*Формула*
6	$\boldsymbol{c} = \min\left\{\dfrac{\boldsymbol{a}^2+\boldsymbol{b}^2}{\mathrm{e}^{\boldsymbol{a}^2+\boldsymbol{b}^2}};\quad \dfrac{\sqrt{\boldsymbol{a}^2+\boldsymbol{b}^2}}{4}\right\}$
7	$\boldsymbol{m} = \max\left\{\mathrm{e}^{-\boldsymbol{x}^2}(\boldsymbol{x}^3+2);\quad \dfrac{\boldsymbol{x}^3+2}{4}\mathrm{e}^{-2\boldsymbol{x}}\right\}$
8	$\boldsymbol{p} = \min\left\{\boldsymbol{x}\,\mathrm{e}^{-\boldsymbol{x}^2}+4;\quad 4\,\mathrm{e}^{-\boldsymbol{x}}+\boldsymbol{x}\right\}$
9	$\boldsymbol{a} = \max\left\{\dfrac{\boldsymbol{x}^2\,\mathrm{e}^{-\boldsymbol{x}}}{\mathrm{e}^{\boldsymbol{x}}+4};\quad \dfrac{(\boldsymbol{x}-1)\sqrt{\boldsymbol{x}}}{1+\sqrt{\boldsymbol{x}}}\right\}$
10	$\boldsymbol{y} = \min\left\{\dfrac{4\boldsymbol{x}\,\mathrm{e}^{-\boldsymbol{x}}}{1+\boldsymbol{x}};\quad \dfrac{2+\boldsymbol{x}}{6+\mathrm{e}^{-\boldsymbol{x}}}\right\}$
11	$\boldsymbol{y} = \min\left\{\dfrac{\lvert\boldsymbol{x}\rvert^3\,\mathrm{e}^{3\boldsymbol{x}}}{1+\mathrm{e}^{3\boldsymbol{x}}};\quad \dfrac{\mathrm{e}^{3+\boldsymbol{x}}}{1+\mathrm{e}^{-\boldsymbol{x}}}\right\}$
12	$\boldsymbol{y} = \max\left\{\dfrac{1}{4}\boldsymbol{x}^2\,\mathrm{e}^{\boldsymbol{x}};\quad \mathrm{e}^{-\boldsymbol{x}}\cos\dfrac{\pi}{4}\boldsymbol{x}\right\}$
13	$\boldsymbol{z} = \max\left\{\dfrac{1}{1+\boldsymbol{x}^2}\ln\dfrac{1+\boldsymbol{x}^2}{2+\boldsymbol{x}^2};\quad \dfrac{\mathrm{e}^{\boldsymbol{x}}\cos\boldsymbol{x}}{\sqrt{2+\cos\boldsymbol{x}}}\right\}$
14	$\boldsymbol{z} = \max\left\{2\sqrt{\boldsymbol{x}\,\mathrm{e}^{-\boldsymbol{x}^2\sin\boldsymbol{\alpha}}};\quad \boldsymbol{x}^2\,\mathrm{e}^{-\frac{\boldsymbol{x}^{2\sin\boldsymbol{\alpha}}}{2{,}83}}\right\}$
15	$\boldsymbol{s} = \max\left\{\boldsymbol{x}^2\sqrt{\dfrac{\boldsymbol{x}}{1-\boldsymbol{x}}};\quad \dfrac{\boldsymbol{x}^3}{\sqrt{1+\boldsymbol{x}^2+\boldsymbol{x}^6}}\right\}$

Лабораторная работа № 2. Циклические вычислительные процессы

Цель работы - освоить на практике программирование простейших циклических процессов, используя для этого операторы цикла, научиться отладке программ и подготовке с этой целью тестов.

Варианты заданий

Задача 1. Составить программу для табулирования функций *f(x)* и *g(x)* при изменении *x* от *a* до *b* с шагом *h*. В первой колонке печатать *x*, во второй - *f(x),* в третьей - *g(x).* Исходные данные приведены в табл.9.

Таблица 9

Вариант	*f(x)*	*g(x)*	*a*	*b*	*h*
1.	$0{,}5\sin^2 x$	$\dfrac{2x}{4+x^2}\ln(3+x)$	0,3	0,36	0,01

2.	$\frac{4arctgx}{1+x^2}$	$\frac{3{,}5x\sin x}{7x^2+2x+1}$	0,12	0,22	0,02
3.	$\frac{1}{\ln 2}\ln\left\|\frac{x-1}{x+1}\right\|$	$x\sin^2(x-3)+x-3$	0,1	3,6	0,5
4.	$\frac{2\cdot 10^{-2}\ln x}{x}$	$\frac{x}{x^2-4x+5}$	1,2	2,2	0,2
5.	$\frac{xe^{x-1}}{x^2+1}$	$(4-x)\ln(1+\|x\|)$	0,3	1,1	0,1
6.	$2xe^{x^2-x}$	$\frac{1}{2}xe^{x(x^2-1)}$	0,5	1,5	0,2
7.	$\left(\sqrt{1-x^2}+x\right)\sin\frac{2}{x}$	$\sqrt{1-x^2}-x^2\sin\frac{2}{x}$	0,1	0,15	0,01
8.	$\frac{x}{3{,}56}arctgx$	$\frac{x^2-3x+3}{x^2-x+1}$	0,5	0,7	0,02
9.	$(2x^2-\ln x)(2-x)$	$e^x(x-2)$	1,5	1,6	0,01
10.	$\frac{1}{2}(x-1)e^{x-3}$	$\frac{x-3}{x+1}e^{(x-1)^2}$	2,0	2,7	0,1

Задача 2

Напечатать значения x, при которых выполняется неравенство (табл.10) при изменении x от a до b с шагом h. Значения переменных a, b и h задать самостоятельно.

Таблица 10

№ варианта	Неравенство
1	$x\lg(1+x^2)>(x+2^{-x})arctg^2x$
2	$\frac{tgx}{\lg 1{,}4}>\frac{x}{1{,}4}$
3	$2{,}3\sqrt{x}>3-\frac{1}{x}$
4	$e^x>2{,}5\ln(1+x^2)$
5	$0{,}8xarctgx>\ln(1+x^2)$
6	$\ln(1+x)>\frac{2arctgx}{1+x}$
7	$1+\ln\left(x+\sqrt{1+x^2}\right)>\sqrt{1+x^2}$
8	$\ln x>1+\frac{2(x-1)}{x+1}$
9	$1{,}2\sin x<x-\frac{x^3}{6}+\frac{x^3}{120}$
10	$\sin x+tgx>2{,}1x$

Задача 3

Вычислить $\boxed{T = \sum_{i=1}^{n} f(x_i),}$, $f(x_i)$, x_i, n – *заданы в таблице 11*, $i \in [1, n]$

Таблица 11

№ варианта	$f(x)$	x_i	n
1	$sin^2 \frac{\pi}{x}$	0,9+0,1i	8
2	$x(1+tgx)$	0,1+0,1i	10
3	$e^{-x} \cos x$	0,1i	12
4	$\frac{x^2}{1+x^2} \sin x$	0,1+0,2i	9
5	$\frac{x}{1+\sqrt{x}} + \ln x$	1+0,5i	10
6	$(x+1)e^{-x} \sin x$	0,2+0,1i	11
7	$(1+x)^{-2} \ln(2+\lvert x \rvert)$	0,5+0,01i	12
8	$x^{1+x} tg \frac{x}{x+1}$	0,1+0,2i	10
9	$(x+1)xe^{-x}$	0,2i	14
10	$(1+ln\,x)e^{-x} \sin^2 x$	1+0,05i	15

Лабораторная работа № 3. Операции с массивами

Цель работы - изучение операций с одномерными и многомерными массивами.

Варианты заданий

Задача 1. Создать два одномерных массива по правилам $a_i = f_1(i)$, где $i = K_1, \ldots, K_2$, $b_j = f_2(j)$, где $j = L_1, \ldots, L_2$ и образовать из них двумерный массив по правилу $C_{ij} = f_3(a_i, b_j)$, где $i = K_3, \ldots, K_4$; $j = L_3, \ldots, L_4$. Исходные данные представлены в табл.12.

Указания:

Задать числа $K_1, K_2, K_3, K_4, L_1, L_2, L_3, L_4$ в начале программы в виде констант и использовать эти константы при описании массивов. Вывести на печать массивы A и B в виде строк, а массив C в виде таблицы; перед выводом каждого массива указать его имя.

Таблица 12

№ вариан-та	K_1	K_2	L_1	L_2	K_3	K_4	L_3	L_4	$f_1(i)$	$f_2(j)$	$f_3(a,b)$
1.	1	10	2	8	3	6	3	7	$\sin i$	e^{-j}	$a^2 - 5b$
2.	0	6	-2	3	0	4	-1	2	$\cos i$	tgj	$a \bullet b$
3.	3	9	7	14	4	8	7	10	$\sqrt{i}$	$j^2 + 5$	$3a + b$

4.	5	12	-5	0	10	12	-4	0	$\lg i$	$j-4{,}5$	$a + 2b$
5.	8	16	-3	5	11	14	1	5	$i+\sqrt{i}$	$\cos j$	$3a + 4b$
6.	-5	2	0	7	-3	1	3	6	$\sin i$	$j^2 - j^3$	$a + b - c$
7.	4	9	1	9	4	7	5	9	$i-\sqrt{i}$	$j\sqrt{j}$	$a(b + 3)$
8.	-3	5	3	5	0	4	4	5	$\cos i$	tgj	$2a - b$
9.	2	10	2	10	2	8	2	6	$\pi i+2{,}5$	$\lvert j-9\rvert$	$a + ab$
10.	7	11	1	5	7	10	1	4	$\ln i$	j^4	$b(a + 4)$

Задача 2.

Написать программу для обработки матрицы в соответствии с вариантом задания (табл.13).

Таблица 13

№ варианта	Имя матрицы и размеры	Действия	Условия и ограничения
1	A(4, 5)	Вычислить и запомнить сумму и число положительных элементов первого столбца матрицы.	$a_{ij}>0$
2	A(N,M)	Вычислить и запомнить суммы элементов каждой строки матрицы. Результаты отпечатать в виде одного столбца.	N<50 M<5
3	B(N, N)	Вычислить сумму и число элементов матрицы, находящихся под главной диагональю.	N<12
4	C(N, N)	Вычислить сумму и число положительных элементов матрицы, находящихся над главной диагональю.	$c_{ij}>0$, N<15
5	D(K, K)	Записать на место отрицательных элементов матрицы нули и вывести ее на печать в общепринятом виде.	K<10
6	D(5, 5)	Записать на место отрицательных элементов матрицы нули, а на место положительных - единицы. Вывести на печать в общепринятом виде.	
7	F(4, 5)	Транспонировать матрицу и вывести на печать элементы главной диагонали Результаты разместить в одной строке.	
8	P(4,6)	Найти наибольший и наименьший элементы матрицы и вывести на печать их номера.	
9	A(4, 6)	Найти среднее арифметическое элементов каждой строки матрицы.	
10	B(5, 4)	Найти среднее геометрическое каждого столбца матрицы	$b_{ij}>0$

Лабораторная работа № 4. Операции с файлами

Цель работы - изучение программирования операций чтения из файлов и записи в файлы

Варианты заданий

Задача 1. Написать программу, которая по заданному одномерному массиву a_i *(i=1,2,...N)* создает два новых массива того же размера $b_i = f_1(a_i)$ и $c_i = f_2(a_i)$ и записать результаты в файл с указанным именем. Проверить результат работы программы, просмотрев содержимое файла с результатами. Исходные данные приведены в табл.14.

Указания

1. Исходный массив должен быть задан внутри программы при его описании.

2. Содержимое файла должно иметь примерно такой вид:

x[1]= 3.3 x[2]= 5.1 x[3]= 0.7
sin(x[1])=-0.158 sin(x[2])=-0.926 sin(x[3])= 0.644
sqrt(x[1])= 1.816 sqrt(x[2])= 2.258 sqrt(x[3])= 0.837

3. Исходные значения массива a_i следует выбрать таким образом, чтобы они входили в область определения функций f_1 и f_2 (табл.14).

Таблица 14

Вариант	*N*	f_1	f_2	*Имя файла*
1.	3	ln	sin	1.res
2.	4	lg	√	2.dat
3.	5	sin	exp	3.dan
4.	4	tg	ln	4.txt
5.	3	ctg	sin	5.lst
6.	5	cos	lg	6.res
7.	4	√	tg	7.dat
8.	3	exp	ctg	8.dan
9.	4	lg	√	9.txt
10.	5	tg	ln	10.lst

где *N* - количество элементов в массиве

Задача 2.

Составить программу для вычисления матричного выражения. (см. задание в таблице 15). Числовые значения элементов матриц задать самостоятельнослучайным образом.

Ввод исходных матриц производить из заранее созданного файла.

Вывести в заранее созданный другой файл исходные данные, промежуточные результаты (матрицы), конечный результат.

Таблица 15

№ варианта	*Матричное выражение*
1	$((Q^T_{34}+D_{43})H_{32})^T=?$
2	$(B^T_{23}+H_{32})(E_{22}+D_{22})=?$
3	$(Q^T_{34}D_{34}+E_{44})^T=?$
4	$(E_{33}+H_{33}+D^T_{33})Q_{34}=?$
5	$((E_{44}+D^T_{44})Q_{43}-B_{43})^T=?$
6	$((H_{34}B_{43})^T+E_{33}-D_{33})^T=?$
7	$((D_{34}+B_{34})D_{43})^T+E_{33}=?$
8	$(D^T_{34}(E_{33}+B_{33}+H_{33}))^T=?$
9	$D_{43}(E_{33}+H_{33})^T+Q^T_{34}=?$
10	$(D_{33}+E_{33})^T+H_{34}Q_{43}=?$

Лабораторная работа 5. Процедуры и функции

Цель: освоить на практике создание процедур и функций пользователя.

Варианты заданий

Задача 1.

Составить процедуры для вычисления и вывода на печать вычисляемых векторов (табл.16), где:

A=(2,1; -3,5; 0); B =(3,2; -0,2; 0,3; 0,4)

D=(2,1; -0,2); F=(3,1; -2,0);

G=(5,1; 0,24 -0,3; 0,4); C=(2,3; 0,1; -0,2; -0,3)

Таблица 16

№ варианта	*Исходные вектора*	*Вычисляемые вектора*	*Элементы вычисляемых векторовпо отношению к элементам исхордных*
1)	A, B	X, Y	Модуль элемента
2)	F, G	Y, Z	На единицу больше
3)	F, C	P, T	На три меньше
4)	B, D	T, V	В пять раз меньше
5)	D, A	V, W	В десять раз больше
6)	D, F	N, X	Куб элемента
7)	F, D	Y,V	Модуль элемента
8)	A, C	V, Z	Синус элемента
9)	F, A	V, R	Косинус элемента
10)	F, B	R,X	Тангенс элемента

:

Задача 2.

Варианты индивидуальных заданий приведены в табл.17.

Таблица 17

№ варианта	***Задачи***
1.	Используя процедуры написать программу вычисления значений массива, элементами которого будут отношения максимального значения выпуска продукции за полугодие к минимальному.
2.	Написать программу вычисления площади выпуклого четырехугольника ABCD, заданного длинами сторон AB, BC, CD, DA и диагональю AC. Диагональ делит выпуклый четырехугольник на два треугольника, к которым применима формула Герона. Для вычисления площади треугольника составить подпрограмму.
3.	Дана матрица Z(3,4) из целых чисел. Если сумма первых индексов четных элементов меньше суммы вторых индексов нечетных элементов, то заменить положительные элементы нулями, иначе поменять местами вторую и последнюю строки матрицы. Полученную матрицу вывести на экран. Написать программу без функций и процедур. Написать программу, оформив замену положительных элементов матрицы нулями в виде процедуры, а вычисление суммы первых индексов четных элементов, в виде функции.
4.	Даны три матрицы L(10,12), N(10,10), K(10,12). Отсортировать их по возрастанию элементов столбцов каждой матрицы. Отсортированные матрицы вывести на экран. Расчеты оформить процедурой.
5.	Даны два вектора H(8) и B(9), состоящие из чисел типа WORD, и число j. Если сумма длин векторов меньше j, то поменять местами наибольшие элементы этих векторов, иначе найти сумму наименьших элементов этих векторов. Написать программу без функций и процедур. Написать программу, оформив ввод значения векторов в виде процедуры, а вычисление длины вектора в виде функции.
6.	Составить программу вычисления значений массива, элементами которого являются отношения значений двумерного массива к минимальному (в %). За 100% принять значение минимального элемента, который нужно предварительно отыскать. Вычисление значений элементов нового массива оформить в виде подпрограммы.
7.	Дана матрица Q(3,3) состоящая из чисел типа byte. Если сумма элементов, стоящих выше главной диагонали, меньше суммы элементов, стоящих ниже ее, то поменять местами диагонали матрицы, иначе поменять местами минимальный и максимальный элементы третьего столбца матрицы. Вывести полученную матрицу. Написать программу без функций и процедур. Написать программу, оформив вывод матрицы в виде процедуры, а нахождение наименьшего элемента в виде функции.
8.	Дана матрица F(3,4), состоящая из чисел типа Longint. Поменять местами наибольший и наименьший элементы и вывести полученную матрицу в виде прямоугольной таблицы. Написать программу без функций и процедур. Написать программу, оформив замену диагоналей матрицы в виде процедуры, а вычисление суммы элементов, стоящих выше главной диагонали, в виде функции.

№ варианта	***Задачи***
9.	Используя процедуры написать программу вычисления значений массива, элементами которого будут суммы значений выпуска продукции поквартально
10.	Дан вектор R(8), состоящий из чисел типа integer. Если меньше половины его элементов четные, то первые три элемента возвести в куб, если нет, то разделить на два его четные элементы. Написать программу без функций и процедур. Написать программу, оформив деление на два его четных элементов в виде процедуры, а возведение в куб - в виде функции.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1) *Бураков П.В., Гусева С.В*. Введение в программирование. Учебное пособие.-СПбГИТМО (ТУ), 2004
2) *Фаронов В.В*. Турбо Паскаль 7.0. Начальный курс: Учебное пособие.-М.: «Нолидж», 1997.-616 с.
3) *Немнюгин С.А., Перколаб Н.В*. Изучаем Turbo Pascal. СПб: Питер, 2005, 313 с.
4) http://www.progbeg.narod.ru/turbo_pascal.html
5) http://school.uni-altai.ru/cs