МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

БИРСКИЙ ФИЛИАЛ ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО
БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БАШКИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Волкова Т.И.

ПРОГРАММИРОВАНИЕ В СРЕДЕ PASCAL ABC

Учебное пособие

Бирск 2013

УДК 681.3.06 (075)
ББК 32.973-01я7
В 37

Печатается по решению
редакционно-издательского совета
Бирского филиала Башкирского
государственного университета

Рецензент:

Набиуллин А.Р., кандидат физико-математических наук, доцент

В37 Волкова Т.И. Программирование в среде Pascal ABC: Учебное пособие для студентов очного и заочного отделений педагогических вузов по специальности «информатика». – Бирск: Бирский филиал Баш ГУ, 2013. – 141 с.

В пособии обобщен многолетний опыт работы автора по обучению основам процедурного программирования студентов педагогических вузов по специальности «информатика», в том числе и при изучении этого курса в рамках второй специальности. Уделено внимание основным понятиям языка программирования Паскаль, операторам ввода и вывода данных, составлению программ, реализующих ветвление, циклические процессы, работе с процедурами, массивами, строками, множествами, файлами и записями. Ко всем изучаемым темам прилагаются контрольные вопросы для самопроверки, задания для самостоятельной работы по рассматриваемым темам, а также лабораторные работы в 12 вариантах.

Оглавление

Раздел 1. ВВЕДЕНИЕ

Общая характеристика языков программирования

Языки программирования (ЯП) – это формальные языки для описания данных (информации) и алгоритма их обработки на ЭВМ.

По наиболее распространенной классификации все языки программирования, в соответствии с тем, в каких терминах необходимо описать задачу, делят на языки низкого и высокого уровня:

Рис. 1 Классификация языков программирования

Если язык близок к естественному языку программирования, то он называется языком высокого уровня, если ближе к машинным командам – языком низкого уровня.

В группу языков **низкого уровня** входят машинные языки и языки символического кодирования: (**Автокод, Ассемблер**). Операторы этого языка – это те же машинные команды, но записанные мнемоническими кодами, а в качестве операндов используются не конкретные адреса, а символические имена. Все языки низкого уровня ориентированы на определенный тип компьютера, т. е. являются машинно-зависимыми. **Машинно-ориентированные языки** – это языки, наборы операторов и изобразительные средства которых существенно зависят от особенностей ЭВМ (внутреннего языка, структуры памяти и т.д.).

Языки **высокого уровня** машинно–независимы, т.к. они ориентированы не на систему команд той или иной ЭВМ, а на систему операндов, характерных для записи определенного класса алгоритмов. Однако программы, написанные на языках высокого уровня, занимают больше памяти и медленнее выполняются, чем программы на машинных языках.

Основная классификация языков программирования высокого уровня основывается на их принадлежности к одному из оформившихся к настоящему времени стилей программирования, каждому из которых соответствует своя собственная модель вычислений. Соответственно выделяют четыре группы языков программирования высокого уровня:

Императивные (процедурные): основными объектами в таких языках являются переменные, операторы присваивания, стандартные алгоритмические конструкции. Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих те или иные действия. Примерами процедурных языков являются Pascal, Fortran, C, Modula, Basic и другие.

Функциональные: программа описывает вычисление некоторой функции, значения которой определяются по заданным параметрам; предназначены для решения задач нечисленного характера. Примером функционального языка является язык ***LISP*** (List Processing-обработка списков). Разработан и реализован в Массачусетском технологическом институте в 1959 г. Рассматривается специалистами как основной язык программирования систем искусственного интеллекта.

Логические: программы на таких языках не описывают действия, они задают данные и отношения между ними, после этого можно задавать вопросы. Языком логического программирования является Prolog – Programming in Logic.

Объектно-ориентированные: создаются объекты, каждый из которых отличается своими свойствами и способами взаимодействия с другими объектами. Программист задает совокупность операций, описывая структуру обмена сообщениями между объектами. К объектно-ориентированным языкам относятся Object Pascal, Delphi, Visual Basic, C++Builder.

В основе любого языка программирования лежит его ***алфавит***, т.е. совокупность основных символов языка. К ним относятся латинские буквы, цифры и некоторые специальные знаки. Из алфавита образуется ***словарь языка –*** его основные конструкции. В словарь языка входят ***величины, выражения, описания и операторы***. Правила записи основных конструкций языка называются ***синтаксисом*** языка, а правила их использования – ***семантикой*** языка (другими словами синтаксис языка программирования – это форма, а семантика – смысл его конструкций).

Понятие о системе программирования. Трансляция программ.

Система программирования – это часть программного обеспечения, с помощью которой разрабатываются все остальные программы. Система программирования состоит из:

- редактора для ввода текста программы на языке программирования
- транслятора с этого языка

- библиотеки подпрограмм
- компоновщика
- отладчика.

Человек пишет программу на языке программирования, но так как программы, которые может выполнять процессор, должны быть написаны на языке машинных команд, то программа, записанная на языке высокого уровня, должна быть сначала переведена (транслирована) на язык машинных команд. Процесс перевода называется *трансляцией*, а программа, которая осуществляет этот перевод, называется *транслятором*.

Рис. 2 Трансляция программ

В результате трансляции каждой команде языка программирования высокого уровня ставится в соответствие несколько машинных команд, а каждой команде языка программирования низкого уровня – одна машинная команда. Соответственно различают три вида трансляторов: интерпретаторы, компиляторы, ассемблеры. Ассемблер, как особый вид иногда не выделяется, т.к. является частным случаем компилятора.

Разница между интерпретатором и компилятором аналогична разнице между переводчиком устной и письменной речи. Переводчик письменной речи сначала переводит весь текст на другой язык и потом исходный текст нам уже не нужен, т.е. мы пользуемся только переводом. Соответственно компилятор полностью переводит весь исходный текст программы на язык машинных команд, и затем полученный модуль используется без участия компилятора. Интерпретатор переводит на язык машинных команд отдельный оператор и этот оператор сразу выполняется. Затем переводится следующий оператор – выполняется и т.д., то есть сколько раз программа выполняется, столько же раз она интерпретируется. Очевидно, что при этом и исходный текст программы, и интерпретатор должны находиться в оперативной памяти. В основном все

языки высокого уровня компилируются и лишь отдельные интепретируются (Бейсик, Лого, ШАЯ).

Текст программы на языке высокого уровня называется *исходным модулем*, а полученная в результате трансляции программа – *объектным модулем*, т.е. схематически трансляцию можно представить в следующем виде:

Компоновка

Объектные модули, полученные в результате трансляции, еще не готовы для выполнения, т.к. содержат в себе как текст программы на языке машинных команд, так и информацию о размещении этих команд в оперативной памяти, а также о подключении других объектных модулей. Затем объектные модули обрабатываются специальной программой компоновщиком, при этом подключаются библиотечные объектные модули, и в результате компоновки получается один **загрузочный модуль,** т.е. файл, готовый к выполнению в среде операционной системы (он имеет расширение .exe). В современных средах программирования процессы трансляции и компоновки обычно объединены и пользователь непосредственно «не видит» объектных модулей.

Отладка

Отладчик позволяет произвести процесс поиска в программе ошибок, просмотреть значения отдельных переменных в любой момент времени (для этого в нужных местах программы предусматривают точки останова) и управлять процессом отладки программы.

Понятие алгоритма и программы. Способы записи алгоритмов

Алгоритм – это строгое и точное предписание последовательности действий для решения поставленной задачи.

Любой алгоритм рассчитан на конкретного исполнителя – того, кто будет выполнять указанную в алгоритме последовательность действий.

Исполнителем алгоритма может быть человек или автоматическое устройство, способное воспринять и выполнить предусмотренные в нем действия

Исполнитель действует **формально,** т.е. он только строго выполняет команды алгоритма, не вникая в содержание поставленной задачи и автоматически выполняя некоторые правила, инструкции.

Конечное множество команд, которые воспринимает исполнитель – это **СКИ (система команд исполнителя)**

В информатике **универсальным исполнителем** алгоритмов является **компьютер.**

Свойства алгоритма:

- **дискретность**: состоит из отдельных шагов (команд)
- **понятность:** должен включать только команды, известные исполнителю (входящие в СКИ)
- **определенность:** любое действие должно быть строго и недвусмысленно определено в каждом случае, при одинаковых исходных данных всегда выдает один и тот же результат
- **конечность** (результативность): приводит к решению задачи (получению результата) за конечное число шагов
- **массовость**: должен быть применим к решению не одной задачи, а некоторого класса задач, различающихся лишь исходными данными

Программа – это

- алгоритм, записанный на каком-либо языке программирования
- набор команд для компьютера

Команда – это описание действий, которые должен выполнить компьютер:

- откуда взять исходные данные?
- что нужно с ними сделать?
- куда (как) вывести результат?

Способы записи алгоритмов:

- Словесный (на естественном языке)
- Блок-схема
- Псевдокод (система обозначений и правил)
- Язык программирования

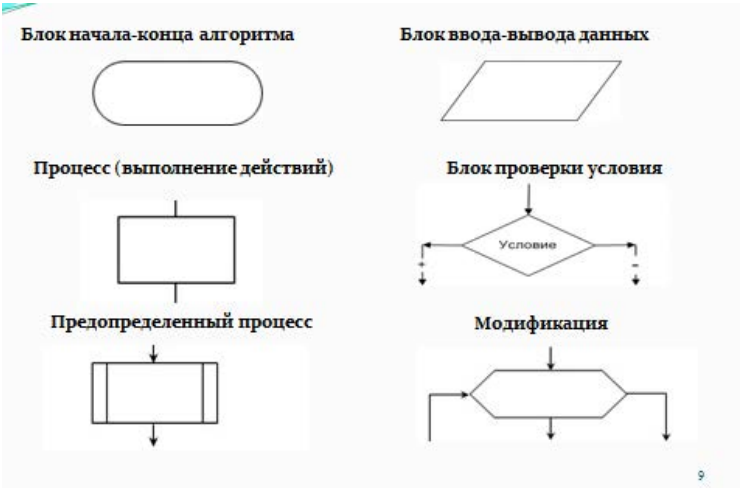
Самый простой способ – **словесный** – это способ записи алгоритма на естественном языке, но с тщательно отработанным набором слов и фраз, не допускающих повторений, синонимов, двусмысленности, лишних слов. Допускается использование математических символов.

Пример алгоритма на естественном языке:

1. Ввести в компьютер числовые значения переменных **a, b и c**.
2. Вычислить d по формуле $\mathbf{d = b^2 - 4ac}$.
3. Если **d < 0**, то напечатать сообщение "Корней нет" и перейти к п.4. Иначе вычислить и напечатать значения $\mathbf{x_1}$ **и** $\mathbf{x_2}$.
4. Прекратить вычисления.

Блок-схемой называется наглядное графическое изображение алгоритма, когда отдельные его этапы изображаются при помощи различных **геометрических фигур - блоков,** а связи между этапами (последовательность выполнения этапов) указываются при помощи стрелок, соединяющих эти фигуры. Блоки сопровождаются надписями.

Основные блоки:



ПРИМЕР. Зная длины трех сторон треугольника, вычислить площадь и периметр треугольника.

Пусть **a, b, c** - длины сторон треугольника. Необходимо найти **S** - площадь треугольника, **P** - периметр.

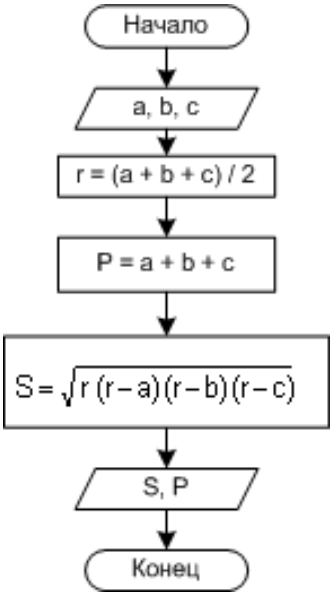
Для нахождения площади можно воспользоваться формулой Герона:

$S = \sqrt{r(r-a)(r-b)(r-c)}$ где **r** - полупериметр

Входные данные*:* **a, b, c**.

Выходные данные*:* **S, P**.

Блок-схема алгоритма



Решение любой задачи на ЭВМ состоит из следующих этапов:

1. Постановка задачи и ее анализ
2. Формализация задачи
3. Построение алгоритма
4. Составление программы на языке программирования
5. Отладка и тестирование программы
6. Проведение расчетов и анализ полученных результатов.

Контрольные вопросы

1. Назовите два основных класса языков программирования.
2. Охарактеризуйте основные группы языков программирования высокого уровня.
3. Что такое система программирования?
4. Что такое транслятор? Назовите типы трансляторов. Чем они отличаются?
5. Что такое алгоритм? Назовите и поясните его основные свойства.
6. Назовите способы представления алгоритмов
7. Перечислите основные этапы подготовки и решения задач на ПК.

Раздел 2.
ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА ПРОГРАММИРОВАНИЯ

Язык программирования Паскаль. Структура программы.

Язык программирования Паскаль был разработан в конце 1960-х годов швейцарским ученым **Никлаусом Виртом** и назван в честь французского философа и математика XVII века Блеза Паскаля. Одной из целей создания языка Паскаль Никлаус Вирт считал обучение студентов структурному программированию. До сих пор Паскаль заслуженно считается одним из лучших языков для начального обучения программированию. Его современные модификации, такие как Object Pascal, широко используются в промышленном программировании (среда Delphi).

Хотя первоначально этот язык был ориентирован для обучения студентов программированию, в дальнейшем постоянно развивался и совершенствовался, появлялись новые версии. В 1982-1984 гг. появилась версия Turbo Pascal, которая в основном вытеснила другие версии. Последние версии - Turbo Pascal 7.0 и 7.1, сочетающие в себе возможности разработки программ в средах MS-DOS и Windows. В настоящее время используются в основном системы программирования **Free Pascal** и **Pascal ABC,** а также последняя версия **- PascalABC.NET,** сочетающая простоту языка Паскаль и огромные возможности платформы .NET. В рамках данного курса мы будем рассматривать версию Pascal ABC 3.0.1, наиболее удобную для начального изучения программирования на процедурном уровне.

Программа на языке Паскаль состоит из последовательности строк, содержащих ключевые слова (слова, смысл и написание которых строго определены, их набор конечен в любом языке программирования), описания и операторы. ***Оператор*** определяет некоторую последовательность действий, т.е. задает несколько машинных команд, которые должны быть выполнены в программе. ***Описания*** определяют условия применимости тех или иных конструкций языка. Разделителем между операторами и между описаниями служит символ “;” (точка с запятой), им заканчивается каждая логическая строка программы (логическая строка может занимать на экране несколько физических строк). В конце программы ставится точка.

Для удобства восприятия текста программы принято проводить структурирование программы по левому краю: одинаковые по “весу” конструкции записывать на одинаковом расстоянии от левого края, соответственно, сдвигая вправо, подчиненные (вложенные) конструкции.

В программе могут быть операторы-комментарии, которые выделяются либо символами “{ }”, либо – “(* *)”.

Структура программы на языке Паскаль имеет следующий вид:

program <имя программы> ;
<раздел описания меток> ;
<раздел описания констант> ;
<раздел описания типов> ;
<раздел описания переменных> ;
<раздел описания процедур и функций>;
begin
<раздел операторов>;
end.

Любой раздел, кроме раздела операторов, может отсутствовать. В разделе операторов описываются основные действия, выполняемые программой. Раздел операторов также называется телом программы.

Порядок следования разделов описаний в ранних версиях Паскаля нельзя менять, однако, в Turbo Pascal возможна произвольная последовательность разделов. В целом, для сохранения стиля программирования, разделы следует располагать в порядке, указанном выше.

Раздел описания меток начинается со слова **«label»**, после которого перечисляются используемые метки. Метка ставится в программе перед любым оператором, который необходимо выделить и отделяется от оператора знаком «:». В качестве метки можно использовать числа, а в Turbo Pascal – и буквы.

Раздел описания констант начинается с ключевого слова «**const**», после которого задаются имена и значения используемых в программе констант

Раздел описания типов начинается со слова «**type**», после которого описываются имена типов величин, используемых в программе. После ключевого слова «type» типы величин описываются в следующем формате:

<имя типа>= <тип> ;

Раздел описания переменных начинается с ключевого слова «**var**», после которого описываются переменные в следующем формате:

<имена переменных> : <тип переменных> ;

Имена переменных одинакового типа разделяются запятыми, а между группами переменных разного типа ствится точка с запятой.

Имя программы, а также все имена констант, типов и переменных - это просто их обозначения в программе, по которым их можно отождествить – ***идентифицировать***. Поэтому в программировании вместо

термина «имя» обычно используется термин «***идентификатор***». Идентификаторы (имена) могут состоять из ***латинских букв, цифр и знаков подчеркивания***, но *первой обязательно должна быть буква* ***(имя не может начинаться с цифры*!).** При этом заглавные и строчные буквы не различаются. Желательно вкладывать в имя смысл, и знак подчеркивания обычно используется вместо пробела в том случае, если имя состоит из нескольких слов.

Далее более подробно рассмотрим основные конструкции языка, а именно типы данных, операции и функции, определенные над ними, а также синтаксис языка и его семантику, т.е. правила составления программ из простейших конструкций.

Величины и выражения. Оператор присваивания

Любая программа предназначена для обработки некоторых данных, которые хранятся в памяти компьютера в виде величин. ***Величина*** имеет три основных характеристики – ***имя, значение*** и ***тип***. *Имя* величины – это обозначение («метка», «бирка») той области памяти, которая отведена под данную величину. *Значение* величины – это содержимое соответствующей величине области памяти. *Тип* величины – это характеристика, определяющая множество допустимых значений величины и множество допустимых операций над ней, а также форму внутреннего представления величины в памяти компьютера. К основным типам величин относятся целый (integer), вещественный (real), символьный (char) и логический (boolean - булевский).

Величины делятся на ***константы*** и ***переменные. Константа*** – это величина, значение которой не изменяется в процессе выполнения программы. Типы констант определяются по контексту (т.е. по форме записи в тексте программы). Примеры описания констант в языке Паскаль:

```
const
x=5; {целое число}
y=2.78; {вещественное число}
s='таблица'; {строка символов- строковая константа}
f=true; {логическая величина}.
```

Переменная – это величина, значение которой можно изменять в процессе работы программы. Типы переменных устанавливаются в разделе описания переменных. Примеры описания переменных в языке Паскаль:

```
var
a, b : integer; x,y,z: real;
```

c1, c2 : char**;**
f : Boolean;

Выражение – это конструкция языка, определяющая порядок вычисления некоторого значения. Различают арифметические и логические выражения. Результатом вычисления арифметического выражения является число, а логического – true или false (булевские значения). Выражения конструируются при помощи имен переменных, констант, операций, функций и круглых скобок. Операции, входящие в выражение, имеют следующий приоритет:

1) вычисление выражений в скобках;
2) *not* - логическое отрицание;
3) **, /, div, mod, and* – умножение, деление (в том числе целочисленное деление и логическое умножение);
4) *+, -, or* – сложение, вычитание, логическое сложение;
5) <, >, <=, >=, =, <> - операции отношения.

Все выражения необходимо записывать линейно. Приведем примеры записи выражений на языке Паскаль:

Арифметические выражения:

Математическая запись	Запись на Паскале
$\frac{a \cdot b}{c + d \cdot e}$	a*b/(c+d*e)
$\frac{\sin x}{\sqrt{\cos x^2}}$	sin(x) / sqrt(cos(sqr(x)));
$\frac{e^x}{x + z}$	exp(x) / (x+z).

Логические выражения:
(x > y) and (x <z);
x or y and z.

Замечание*: Здесь приведены только отдельные примеры, иллюстрирующие запись выражений на Паскале. Подробный перечень операций и функций, а также их обозначения и смысл будут рассмотрены далее в теме «Стандартные типы данных и операции над ними».*

Мы отмечали, что значение – это содержимое области памяти, отмеченной именем величины. Для присваивания величине какого-либо

значения или для изменения существующего значения величины определена команда (оператор) присваивания, которая имеет следующий формат:

< имя величины > := < выражение >.

При выполнении команды присваивания вычисляется значение выражения, стоящего справа от знака присваивания (:=), и полученное значение заносится в область памяти с соответствующим именем величины (стоящим слева от знака присваивания).

Следует помнить:

1. *команда присваивания работает «справа-налево».*
2. *Отличия команды присваивания и операции «=» в математике*: с точки зрения математики равенство вида a=a+2 – не имеет смысла (тождественно ложно – 0=2), а с точки зрения программирования присваивание вида a:=a+2 означает, что в памяти ищется ячейка с именем a, к её содержимому (значению) добавляется число 2, и полученный результат заносится в ячейку с именем a (старое содержимое при этом стирается).

Организация ввода и вывода в программах на языке Паскаль. Линейные программы.

Линейной называется программа, последовательность записи команд в которой совпадает с последовательностью их выполнения. Простейшие линейные программы обычно сводятся к тому, что пользователь с клавиатуры вводит значения некоторых величин (числа или символы), затем эти значения обрабатываются, обычно с помощью команды присваивания, и полученный результат выводится на экран компьютера. Таким образом, в любом языке программирования должны быть команды, позволяющие организовать обмен информацией между человеком и компьютером – ввод исходных данных и вывод результатов.

Для вывода информации на экран в языке Паскаль используется специальная процедура вывода. Она имеет формат:

Write[ln](< список вывода >);

Здесь и далее в квадратных скобках будем записывать необязательную часть каких-либо конструкций языка (она может быть, а может и отсутствовать). В данном случае сочетание **ln** означает перевод курсора на следующую строку экрана по окончании вывода всех элементов списка. При её отсутствии по окончании вывода курсор остается в текущей позиции строки вывода.

Список вывода **может содержать три вида элементов:**

1) Имена переменных
2) Константы (числовые и строковые)
3) Выражения

Разделителем в списке служит запятая.

При выполнении оператора вывода на экран *соответственно* выводятся;

1) Значение переменной
2) Сама константа
3) Значение выражения

К именам переменных в списке вывода могут добавляться через знак двоеточие числа, обозначающие количество позиций на экране, отводимых под соответствующие значения переменных. Для переменных целого и символьного типа указывается одно число, а для переменных вещественного типа – два числа: первое определяет общее количество позиций, отводимых под число на экране, а второе – количество позиций, отводимых под дробную часть числа.

Приведем пример программы для иллюстрации работы оператора вывода:

Текст программы:

```
program primer_vivod;
    var a,b: integer; c:real; d:char;
begin
    a:=3; b:=5; c:=37.6; d:='@';
    writeln(a,b);
    writeln(a:8,b:10);
    writeln('a',d:6,b:6);;
    writeln(c);
    writeln(c:10:4);
     writeln('a+b=',a+b);
     writeln('a=',a,'   b=',b,d,d,d,2*a-b)
end.
```

Вывод на экран (протокол работы программы):

```
35
     3        5
a    @     5
 3.7600000000E+01
   37.6000
a+b=8
a=3   b=5@@@1
```

Замечание: в четвертой строке при выводе значения вещественного типа результат получен в форме с плавающей запятой - экспоненциальной форме (мантисса равна 3.7600000000, а запись E+01 означает 10^1). В следующей строчке это же число выводится в форме с фиксированной запятой – отводится 10 позиций на экране, из них 4 под дробную часть, одна – для точки и две для целой части (в начале остается три пробела).

При этом следует отметить, что по умолчанию вывод вещественных чисел в экспоненциальной форме используется в классических версиях Паскаля, а в среде Pascal ABC по умолчанию происходит преобразование в форму с фиксированной запятой – на рисунке 3 представлен скриншот программы и протокол ее работы:

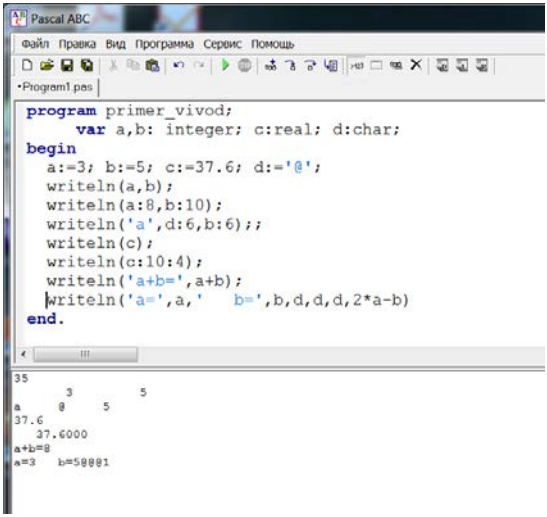


Рис. 3 Программа и протокол ее работы в среде Pascal ABC

Для ввода данных в программу с клавиатуры служит процедура ввода. Она имеет следующий формат:

Read[ln] (< список имен переменных >);

Разделителем в списке служит запятая. Сочетание ***ln*** обеспечивает переход курсора к новой строке по окончании ввода.

При выполнении оператора ввода программа приостанавливает свою работу, и ждет ввода соответствующих (указанных в списке переменных) значений. Числовые значения набираются через пробел, а символьные – без пробела. По окончании нажимается клавиша ввода: “ ↵ ” - Enter. ***Число имен переменных в списке и количество набираемых на***

клавиатуре значений должны быть равны, их типы и порядок следования также должны быть согласованы.

Для того, чтобы пользователю было понятно, что следует вводить (набирать на клавиатуре), перед оператором ввода обычно помещают оператор вывода с наводящим сообщением – подсказкой о смысле вводимых данных, например:

```
Write('введите три числа:   '); {подсказка}
Readln(a,b,c); {ввод}
```

Пример 1. Составить программу, вычисляющую площадь круга и длину окружности заданного радиуса (радиус окружности вводится с клавиатуры).

Текст программы:

```
Program krug;
  Const pi=3.1415;
  Var     R, S, L: real;
Begin
  Write ('введите радиус   ');
  Readln (R);
  S:= pi*R* R;
  L:=2*pi*R;
  Writeln ('S=', S:5:2, '  L=', L:5:2);
End.
```

Пример 2. Составить программу, которая вводит с клавиатуры два целых числа, а выводит их сумму, разность и произведение, а также значение логического выражения – true (false), если первое число больше (меньше) второго.

Текст программы:

```
program primer_2;
  var a,b: integer;
begin
  write('Введите два целых числа  ');
  readln(a,b);
  writeln('Сумма = ',a+b);
  writeln('Разность = ',a-b);
  writeln('Произведение = ',a*b);
  writeln('Первое число больше второго? ',a>b);
end.
```

Контрольные вопросы и задания

Вопросы

1) Какой из перечисленных разделов обязателен в программе:
- ✓ var
- ✓ const
- ✓ type
- ✓ раздел begin .. end.
- ✓ label

2) Какие процедуры служат в Паскале для выполнения операций ввода-вывода?
3) Напишите оператор ввода переменной K с клавиатуры.
4) Для каких целей служит оператор присваивания?
5) Чем отличаются операторы ввода Read и Readln?
6) Для каких целей служит оператор Write?
7) Чем отличаются операторы вывода Write и Writeln?
8) Для чего в процедурах вывода определяется ширина поля вывода?
9) Какие обозначения используются в форматах вывода?

Задания

Составить программы, считая, что все входные и выходные данные являются вещественными числами.

1. Даны два круга с общим центром и радиусами $R1$ и $R2$ ($R1 > R2$). Найти площади этих кругов $S1$ и $S2$, а также площадь $S3$ кольца, внешний радиус которого равен $R1$, а внутренний радиус равен $R2$:
2. Дана длина L окружности. Найти ее радиус R и площадь S круга, ограниченного этой окружностью
3. Дана площадь S круга. Найти его диаметр D и длину L окружности, ограничивающей этот круг.
4. Даны два ненулевых числа. Найти сумму, разность, произведение и частное их квадратов.
5. Найти значение функции $y = 3 \cdot x^6 - 6 \cdot x^2 - 7$ при данном значении x.
6. Найти значение функции $y = 4 \cdot (x-3)^2 - 7 \cdot (x-3)^3 + 2$ при данном значении x.
7. Дано число A. Вычислить A^8, используя вспомогательную переменную и три операции умножения. Для этого последовательно находить A^2,A^4, A^8. Вывести все найденные степени числа A.
8. Дано число A. Вычислить A^{15}, используя две вспомогательные переменные и пять операций умножения. Для этого последовательно находить A^2, A^3, A^5, A^{10}, A^{15}. Вывести все найденные степени числа A.

9. Известно, что X кг шоколадных конфет стоит A рублей, а Y кг ирисок стоит B рублей. Определить, сколько стоит 1 кг шоколадных конфет, 1 кг ирисок, а также во сколько раз шоколадные конфеты дороже ирисок.

10. Дана длина ребра куба a. Найти объем куба $V = a^3$ и площадь его поверхности $S = 6 \cdot a^2$.

11. Даны длины ребер a, b, c прямоугольного параллелепипеда. Найти его объем $V = a \cdot b \cdot c$ и площадь поверхности $S = 2 \cdot (a \cdot b + b \cdot c + a \cdot c)$.

12. Даны координаты двух противоположных вершин прямоугольника: ($x1$, $y1$), ($x2$, $y2$). Стороны прямоугольника параллельны осям координат. Найти периметр и площадь данного прямоугольника.

Стандартные типы данных и операции над ними

В языке Pascal определены ***стандартные*** типы величин (однозначно заданные синтаксисом и семантикой языка) и ***нестандартные*** типы, которые может конструировать сам программист. К ***простым стандартным типам*** языка Паскаль (простые типы определяют неделимые величины) относятся: ***целый, вещественный, символьный, булевский,*** рассмотрим их для классической версии Turbo Pascal 7.0 (в системе Pascal ABC диапазон значений немного отличается – см. встроенную справку системы Pascal ABC):

1. Целые типы – всего их пять и они различаются количеством значащих цифр и диапазоном допустимых значений, а также, соответственно, объемом занимаемой памяти (см. таблицу 2.1).

Над переменными целочисленного типа определены следующие ***арифметические операции*** : ***+, -, *, /, div, mod***. Результат выполнения операции деления всегда получается вещественного типа, остальные – целого. Div – операция выделения целой части, mod – нахождение остатка от целочисленного деления, например 8 div 3 = 2, 8 mod 5 = 3.

Таблица 2.1

Целочисленные типы данных

Тип	Диапазон	Размер (в байтах)
Shortint	-128..127	1
Integer	-32768..32767	2
Longint	-2147483648..2147483647	4
Byte	0..255	1
Word	0..65535	2

Над данными целого типа определены следующие ***операции отношения***: **=, <>, <, >, <=, >=,** вырабатывающие результат логического типа. Рассмотрим стандартные функции (см. таблицу 2.2), определенные для переменных целого типа (x – переменная целого типа):

Таблица 2.2

Стандартные функции

Запись на Паскале	**Обычная запись, пояснения**
sin(x), cos(x), arctan(x)	sin x, cos x, arctg x
ln(x)	натуральный логарифм x
exp(x)	экспонента x
abs(x)	абсолютная величина (модуль) x
sqr(x)	квадрат x
sqrt(x)	корень квадратный из x
pred(x)	возвращает число, предшествующее x
succ(x)	возвращает число, следующее за x
odd(x)	проверяет, является ли аргумент нечетным числом и возвращает результат логического типа: для четного аргумента – false; для нечетного – true.

Результат этих функций, кроме sqr(x), abs(x), pred(x), succ(x), является вещественным.

2. ***Вещественный тип данных***. Вещественные значения могут быть записаны в двух формах: в форме с фиксированной запятой и в форме с плавающей запятой (экспоненциальная форма). Значения в форме с плавающей запятой представляются в виде mEp, где m-мантисса, p – показатель степени числа 10 (порядок), например 8.7612E+2, 1.34E-3. Соответственно в форме с фиксированной запятой эти числа запишутся в виде 876.12 и 0.00134 (целая часть от дробной отделяется точкой, а не запятой!). Всего существует 5 вещественных типов, различающихся числом значащих цифр и размером (см. табл.2.3).

Таблица2.3

Вещественные типы данных

Тип	**Диапазон**	**Число значащих цифр**	**Размер в байтах**
Real	2.9e-39..1.7e38	11-12	6
Single	1.5e-45..3.4e38	7-8	4
Double	5.0e-324..1.7e308	15-16	8
Extended	3.4e-4932..1.1e4932	19-20	10
Comp	-9.2e18..9.2e18	19-20	8

Над данными вещественного типа определено четыре арифметические операции (+, -, *, /), операции отношения, восемь стандартных математических функций (те же, что и над целыми типами) и функции:

Int(x)- возвращает целую часть x;

Frac(x)- возвращает дробную часть x;

trunc(x) - отбрасывает дробную часть числа x;

round(x) - округляет до ближайшего целого числа.

В классическом Паскале нет функции возведения числа в произвольную степень. Для того чтобы вычислить $y=a^b$ обычно используют следующие преобразования:

$$y = a^b;$$
$$\ln y = b \ln a;$$
$$y = e^{b \ln a};$$
$$y = \exp(b \ln a).$$

В системе же Pascal ABC есть специальная функция **`Power(x,y)`**, где **`x,y – real`**, которая возводит x в степень y

3. Символьный тип обозначается ключевым словом ***char*** и его значением является один символ, которому поставлен в соответствие определенный код – число от 0 до 255. Символьные константы заключаются в апострофы.

Над символьным типом определены операции отношения. Пусть S1 и S2 – переменные символьного типа, тогда S1 < S2 если код(S1) < код (S2). Есть две специальные функции, позволяющие определить символ по его коду и наоборот:

ord (< символ >) – возвращает код символа ;

chr (< код >) - возвращает символ по указанному коду.

Символьная константа может записываться в тексте программы двумя способами:

-как один символ, заключенный в апострофы;

-с помощью конструкции вида **#K**, где K - код соответствущего символа, при этом значение K должно находиться в пределах 0..255;

К аргументам символьного типа применяются функции, которые определяют предыдущий и последующий символы: - ***Pred*** и ***Succ***, например, Pred('F') = 'E' ; Succ('Y') = 'Z'. При отсутствии предыдущего или последующего символов значение соответствующих функций не определено.

Для символов из интервала 'a'..'z' применима функция ***UpCase(<символ>***), которая переводит строчные буквы в заглавные (верхний регистр):'A'..'Z'.

4. Булевский (логический) тип. Величины булевского типа могут принимать два значения: true – истина, false – ложь, причем false < true. Над булевским типом определены следующие логические операции:

1) логическое умножение (конъюнкция) – and (и);
2) логическое сложение (дизъюнкция) – or (или);
3) логическое отрицание – not (не);
4) исключающее или - xor

В таблице 2.4 рассмотрены их результаты.

Таблица 2.4

Таблица истинности логических операций

X	Y	X and Y	X or Y	Not X	X xor Y
1	**1**	**1**	**1**	**0**	**0**
0	**0**	**0**	**0**	**1**	**0**
1	**0**	**0**	**1**	**0**	**1**
0	**1**	**0**	**1**	**1**	**1**

Пример 1. Дано целое положительное четырехзначное число X. Составить программу, выводящую на экран значение true, если сумма первых двух его цифр больше, чем сумма третьей и четвертой цифр, и значение false в противном случае.

Указания к решению:

Для решения задачи необходимо сначала выделить цифры четырехзначного числа. Для этого используются операции нахождения целой части и остатка от деления этого числа на 10 или соответственно степени числа 10. Например, пусть X=4276. Тогда последовательно находим цифры:

4276 mod 10 =6
4276 div 10 mod 10 = 7 или 4276 mod 100 div 10 = 7
4276 div 100 mod 10 = 2 или 4276 mod 1000 div 100 = 2
4276 div 1000 = 4

Затем составляем логическое выражение – отношение между суммой двух первых и двух последних цифр и выводим на экран его значение.

Текст программы:

```
program cifr_chisla;
  var x:word; c1,c2,c3,c4:byte;
     f:boolean;
begin
  write('введите 4-значное число ');
  readln(x);
```

```
  c4:=x mod 10; {последняя цифра}
  c3:=x div 10 mod 10; {третья цифра}
  c2:=x div 100 mod 10; {вторая цифра}
  c1:=x div 1000; {первая цифра}
  f:=(c1+c2)>(c3+c4);
  writeln(f)
end.
```

Лабораторная работа №1

Тема:

Ввод и вывод значений стандартных типов. Оператор присваивания.

Цель:

1) Освоение простейшей структуры программы.
2) Получение навыков в организации ввода/вывода значений стандартных типов.
3) Получение навыков в записи арифметических и логических выражений.

Содержание отчета (по каждому заданию):

1. Постановка задачи (условие для своего варианта)
2. Блок-схема решения (только для задания 1)
3. Текст программы (скопировать как текст из среды с сохранением форматирования (цвет, шрифт и т.д. – чтобы было видно, что это текст работающей программы)
4. Протокол отладки (тесты – контрольные примеры, результаты отладки на тестах) – сделать скрин-шоты среды – выполнение (только окно результатов).

Задание 1 Составить блок-схему и программу (по вариантам). Входные данные – переменные целого типа, выходные данные – вещественного типа.

1) Дана сторона квадрата a. Найти его периметр P.
2) Дана сторона квадрата a. Найти его площадь S.
3) Даны стороны прямоугольника a и b. Найти его площадь S и периметр P .
4) Дан диаметр окружности d. Найти ее длину L.
5) Дана длина ребра куба a. Найти объем куба V и площадь его поверхности S
6) Даны длины ребер a, b, c прямоугольного параллелепипеда. Найти его объем V и площадь поверхности S

7) Найти длину окружности L и площадь круга S заданного радиуса R:
8) Даны два числа a и b. Найти их среднее арифметическое
9) Даны два неотрицательных числа a и b. Найти их среднее геометрическое, то есть квадратный корень из их произведения
10) Даны два ненулевых числа. Найти сумму, разность, произведение и частное их квадратов.
11) Даны два ненулевых числа. Найти сумму, разность, произведение и частное их модулей.
12) Даны катеты прямоугольного треугольника a и b. Найти его гипотенузу c и периметр P

Задание 2. Даны вещественные x, y, z. Вычислить a, b, если

№ варта	a=	b=
1.	$\frac{\sqrt{\lvert x-1\rvert}-\sqrt[3]{\lvert y\rvert}}{1+\frac{x^2}{2}+\frac{y^2}{4}}$	$x\left(arctgz+e^{-(x+3)}\right);$
2.	$\frac{3+e^{y-1}}{1+x^2\lvert y-tgz\rvert}$	$1+\lvert y-x\rvert+\frac{\lvert y-z\rvert^3}{3}$
3.	$(1+x)\frac{y+z/\left(x^2-5\right)}{e^{-x-2}+1/\left(x^2-5\right)}$	$\cos(y-2)+\frac{x^4}{2}+\sin^2 z$
4.	$y+\frac{x}{y^2+\frac{z^2}{y+x^3/3}}$	$1+tg^2\frac{z}{2}$
5.	$\frac{2\cos\left(x-\frac{\pi}{5}\right)}{1/2+\sin^2 y+\cos^2 z}$	$y\cdot\left(arctgz-\frac{1}{\sqrt{x^2+y^2}}\right)$
6.	$\frac{1+\sin^2(x+y)}{2+\lvert z-2z/\left(1+x^2\right)\rvert}$	$\sqrt{x^2+tg^2(y+z)}$
7.	$\ln\left\lvert\left(y-\sqrt{\lvert x\rvert}\right)\left(z-\frac{y}{x+0{,}5}\right)\right\rvert$	$arctg\left(\frac{x^3-y^2+e^z}{\sin^2(x+y)}\right)$

8.	$\frac{\sqrt{\lvert y+z\rvert}-\sqrt[4]{\lvert x^2-z\rvert}}{e^x+e^y}$	$\ln\left(\left\lvert\cos^2(x+y)+\frac{x^2}{4}-\frac{z^3}{3}\right\rvert\right)$
9.	$\frac{\sqrt{x+y+z}-\sqrt[3]{x}}{1+\lvert x^3-5\rvert}$	$cos^2\left(1+arctg\frac{1}{z}\right)$
10.	$\frac{x^2+3y-\sqrt{z}}{y\cdot\sin z+e^{\lvert x\rvert}}$	$\sin^2\left(5x+\frac{1}{y}+e^z\right)$
11.	$\frac{\sqrt{\lvert x+y\rvert}-\sqrt[3]{\lvert x\rvert}}{\sin(x+y+z)}$	$\ln\left\lvert y^{\frac{3}{2}}+5\sin(x)+\sqrt[3]{z}\right\rvert$
12.	$\frac{2\sin\left(y+z-\frac{\pi}{8}\right)}{e^{x+y}+1/(z^3-5)}$	$z\cdot\left(arctgy-\frac{1}{\sqrt[3]{x^2+z^2}}\right)$

Задание 3. Написать программу, которая печатает значение логического выражения TRUE или FALSE, в зависимости от ложности или истинности следующего утверждения:

1. Сумма двух первых цифр заданного четырехзначного числа равна его последней цифре.
2. Квадрат второй цифры трехзначного числа равен сумме цифр этого числа.
3. Среди цифр трехзначного числа есть одинаковые.
4. Средняя цифра трехзначного числа в два раза больше суммы первой и третьей цифр.
5. Сумма первой и последней цифры четырехзначного числа равна второй цифре.
6. Первая и последняя цифры трехзначного числа равны.
7. Сумма второй и третьей цифр четырехзначного числа больше восьми.
8. Среди цифр четырехзначного числа есть четная.
9. Сумма первой и последней цифр четырехзначного числа больше десяти.
10. Все цифры четырехзначного числа нечетные.
11. Первая и последняя цифры трехзначного числа равны.
12. Сумма цифр четырехзначного числа больше квадрата его последней цифры.

Раздел 3. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ КОНСТРУКЦИИ И ИХ РЕАЛИЗАЦИЯ В ЯЗЫКЕ ПАСКАЛЬ

Разветвляющиеся алгоритмы и программы

В жизни нам часто приходится сталкиваться с ситуацией, когда необходимо выбрать какой-то один из нескольких вариантов действий в зависимости от некоторого условия. Классическим примером такого выбора является сюжет из сказки: «Едет добрый молодец на коне и подъезжает к развилке из трех дорог…..» (рис.3.1):

Рис. 3.1 Пример ветвления

Аналогичных примеров из жизни можно привести много, например:

ЕСЛИ пошел дождь,
ТО надо открыть зонт
ВСЕ
ЕСЛИ назвался груздем,
ТО полезай в кузов
ВСЕ
ЕСЛИ ласточки летают низко,
ТО будет дождь,
ИНАЧЕ дождя не будет
ВСЕ

В программировании для реализации такого рода ситуаций используются разветвляющиеся алгоритмы (программы):

Разветвляющимся называется алгоритм (программа), в котором в зависимости от истинности или ложности некоторого условия, выбирается один из двух (или нескольких) возможных путей продолжения алгоритма.

Эти пути продолжения называют **ветвями.**

В зависимости от количества ветвей различают:

1. **Полное ветвление** (две ветви)
2. **Неполное ветвление** (одна ветвь)
3. **Многократное ветвление** (больше двух ветвей)

На рисунке 3.2 приведена блок-схема полного и неполного ветвлений

Рис. 3.2 Иллюстрация полного и неполного ветвлений

В языке Паскаль для реализации разветвляющихся программ используется ***условный оператор***.

Он имеет следующий ***формат***:

If `<логическое выражение>`
then `<оператор 1>`
[**else** `<оператор2>`];

Если в условном операторе присутствуют обе ветви (**then и else**), то такую форму оператора называют ***полной***, если же ветвь else отсутствует – ***неполной***.

При выполнении условного оператора в полной форме вычисляется значение логического (булевского) выражения. Если оно равно ‘true’, то выполняется <оператор 1>, а <оператор 2> пропускается; если же оно равно ‘false’, то выполняется <оператор 2>, а <оператор 1> пропускается.

При выполнении условного оператора в неполной форме также сначала вычисляется значение булевского выражения. Если оно равно ‘true’, то выполняется <оператор 1>, если ‘false’- осуществляется переход к оператору, следующему за условным оператором, то есть в последнем случае <оператор 1> просто пропускается (“обходится”).

После ключевых слов ***then и else*** по правилам языка Паскаль можно записывать только один оператор, а часто бывает необходимо выполнить несколько операторов на каждой из ветвей. Тогда эти операторы объединяются в один **составной оператор** при помощи **операторных скобок,** роль которых выполняют ключевые слова ***begin* (открывающаяся скобка) и *end* (закрывающаяся скобка)**. То есть ***составной оператор*** имеет следующий ***формат:***

```
begin
   <оператор 1>;
    <оператор 2>;
     ……
    <оператор n>
end;
```

Если в разветвляющейся программе необходимо организовать более двух ветвей (многократное ветвление), то можно использовать вложенные условные операторы, то есть в качестве операторов на ветвях *then* или *else* будут также стоять условные операторы.

При этом для удобочитаемости программы необходимо выравнивать по левому краю (то есть располагать на одной вертикальной линии) соответствующую пару ключевых слов *then-else:*

```
If <логическое выражение 1>
  then
       if < логическое выражение 2>
          then <…….>
          else <…….>
  else
       if < логическое выражение 3>
          then <…….>
          else <…….>;
```

В целом же следует помнить, *что по правилам языка Паскаль* ***каждому else*** *соответствует* ***ближайший*** *стоящий перед ним* ***then (if).***

Это соглашение позволяет избежать неоднозначности в трактовке выполнения вложенных условных операторов.

В качестве примера рассмотрим классическую задачу нахождения максимального из трех чисел и проиллюстрируем ее решение несколькими способами.

Постановка задачи

Даны три целых числа. Найти максимальное из них.

Введем обозначения: исходные числа a,b,c. Результат - m (максимальное из них).

1 способ решения

Сравним первые два числа, найдем максимальное из них и результат сохраним в переменной m (полное ветвление).

Затем сравним m и c (неполное ветвление)

Ниже (рис.3.3) приведены блок-схема алгоритма, текст программы и протокол работы в среде Паскаль АВС

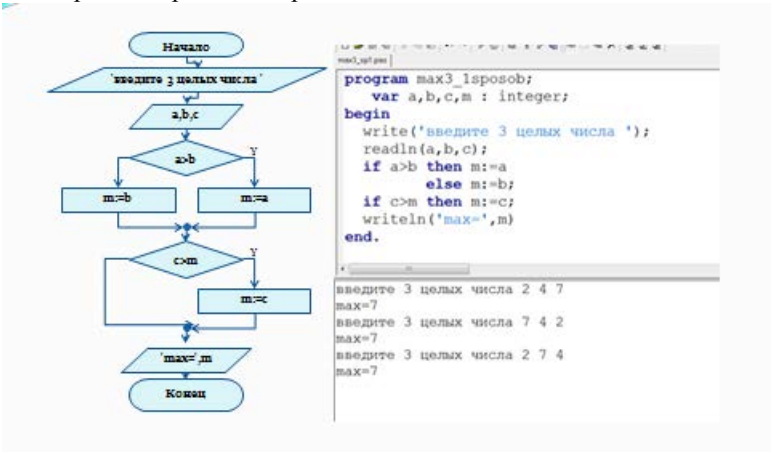


Рис. 3.3 Первый способ решения

2 способ решения

Сначала считаем, что максимальное – первое число (m:=a). Потом сравниваем последовательно m и b , m и c (неполные ветвления).

На рисунке 3.4 приведены блок-схема и текст программы:

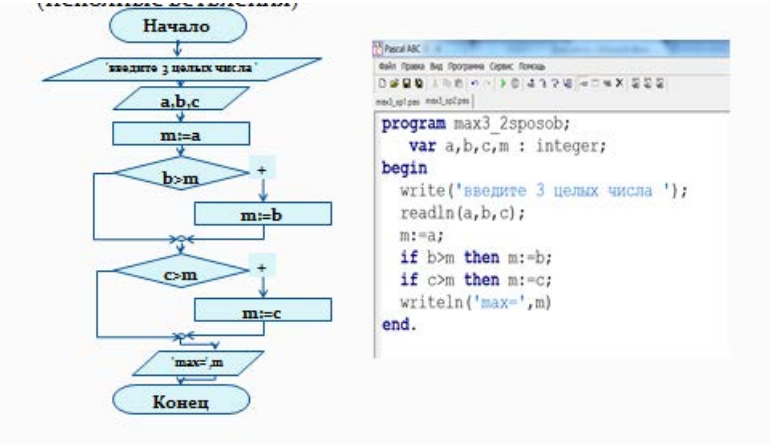


Рис. 3.4 Второй способ решения

3 способ решения (этот способ интуитивно понятный, но неэффективный)

Три последовательных неполных ветвления с составными условиями (рис. 3.5):

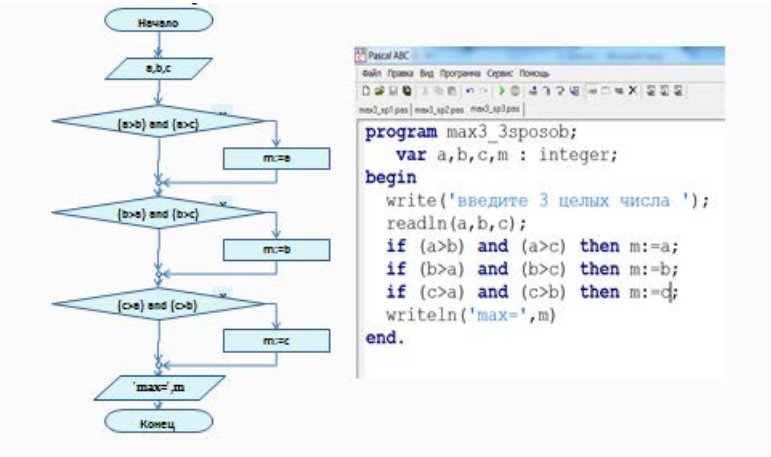
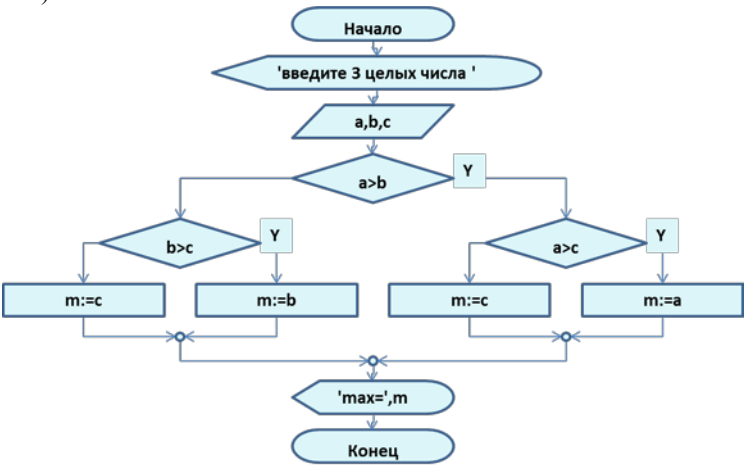


Рис. 3.5 Третий способ решения

4 способ решения

Вложенные ветвления: во внешнем сравниваем два первых числа, а на каждой из ветвей соответственно максимальное из них с третьим (рис. 3.6):



Pascal ABC

Файл Правка Вид Программа Сервис Помощь

max3_sp1.pas | max3_sp2.pas | max3_sp3.pas | max3_sp4.pas

```
program max3_4sposob;
   var a,b,c,m : integer;
begin
  write('введите 3 целых числа ');
  readln(a,b,c);
  if a>b then if a>c then m:=a
                     else m:=c
         else if b>c then m:=b
                     else m:=c;
  writeln('max=',m)
end.
```

Рис. 3.6 Четвертый способ решения

Примеры реализации полного и неполного ветвлений в языке Паскаль. Оператор перехода.

В этом параграфе рассмотрим еще несколько примеров, иллюстрирующих использование условного оператора в полной и неполной формах в языке Паскаль. Также приведем пример использования безусловного оператора (оператора перехода).

Пример 1. Даны три целых числа a,b,c. Определить, принадлежит ли число с отрезку [a,b]. Ответ на экран выдать в виде сообщения ‘Да’ или ‘Нет’.

Указания: Принадлежность числа с отрезку [a,b] определяется с помощью булевского выражения (c>=a) and (c<=b). Если оно истинно, то с принадлежит отрезку, иначе – не принадлежит. Соответственно в программе необходимо использовать полную форму условного оператора, на каждой из ветвей которого будет располагаться один оператор вывода *writeln* с текстом сообщения:

```
Program poln_vetvlen;
  Var a,b,c : integer;
   begin
     write(‘Введите a,b,c’);
     readln(a,b,c);
     If (c>=a) and (c<=b)
          then writeln(‘Да’)
          else writeln(‘Нет’);
   end.
```

Пример 2.

Даны три вещественных числа. Просуммировать те из них, которые являются положительными.

Указания: Эта программа является частным случаем задачи обработки потока данных, а именно – нахождение суммы чисел некоторой последовательности, удовлетворяющих заданному условию. Первоначально сумму необходимо обнулить, а затем по очереди проверять каждое из чисел на выполнение заданного условия, и в случае истинности условия добавлять соответствующее число к сумме с помощью оператора присваивания вида: S:=S+<число>. То есть в программе необходимо организовать три последовательных неполных ветвления:

```
Program nepoln_vetvlen;
        Var a,b,c,s : real;
      Begin
        Write('введите три числа ');
        Readln(a,b,c);
          S:=0;
        If  a>0 then s:=s+a;
        If  b>0 then s:=s+b;
        If  c>0 then s:=s+c;
        Writeln( 'Сумма равна ',s:5:2)
      End.
```

В двух рассмотренных примерах на каждой из ветвей стояло по одному оператору. Теперь рассмотрим пример, в котором на каждой ветви будет более одного оператора, то есть необходимо будет использовать составной оператор:

Пример 3. Даны два вещественных числа x и y. Возвести эти числа в квадрат, если оба числа отрицательны, и удвоить их в противном случае.

Указания: При истинности булевского выражения *(x<0) and (y<0)* необходимо возводить каждое из чисел в квадрат, то есть выполнять два оператора присваивания, при ложности – также два оператора присваивания для умножения каждого из чисел на число 2. Поэтому на ветвях ***then*** и ***else*** необходимо использовать составной оператор (не забывать ставить операторные скобки ***begin- end***):

```
Program sostavn_oper;
  var x,y: real;
begin
   write('Введите 2 числа ');
   readln(x,y);
   if (x<0) and (y<0)
      then
      begin x:=sqr(x);
            y:=sqr(y)
      end
      else
      begin  x:=2*x;
             y:=2*y
       end;
   writeln(x:6:2, y:6:2)
end.
```

Оператор перехода

Оператор перехода имеет вид:

goto <метка>.

Здесь goto – ключевое слово – перейти, то есть этот оператор позволяет изменить естественный ход выполнения программы и передать управление оператору, помеченному соответствующей меткой. Паскаль является структурированным языком программирования и любую программу на нем можно реализовать без оператора перехода. Использовать этот оператор в программах необходимо только в крайних случаях, например, для задания точек останова или аварийного завершения работы программы. Иногда (в неструктурированных языках программирования) этот оператор используют для реализации разветвляющихся программ, в Паскале же с этой целью его употреблять не стоит. Приведем простейший пример программы, в котором использование оператора перехода оправдано:

Пример 4. В программе необходимо организовать ввод целых положительных чисел, отрицательные числа и ноль не пропускать (‘блокировать’):

```
Program prim_goto;
   label 1;
   var x : integer;
begin
     1: write(‘Введите целое положительное число ’);
        readln(x);
     if (x<=0) then begin writeln (‘Повторите ввод! ’);
                          goto 1
                    end;
 end.
```

На этом примере проиллюстрирован синтаксис использования оператора перехода и меток: в качестве метки в языке Паскаль можно использовать либо целое положительное число, либо произвольный идентификатор. Метка располагается непосредственно перед оператором, на который передается управление и отделяется от него двоеточием. Предварительно метка должна быть описана в разделе *LABEL*.

Контрольные вопросы и задания.

Вопросы:

1. Какие программы называются разветвляющимися?
2. Какие форматы условного оператора существуют?
3. Как называется условный оператор, если отсутствует ветвь else?
4. Сформулируйте правило выполнения условного оператора.
5. Найдите ошибки в записи оператора: if x>2 then y:=x*x; else y=sin(x);
6. Дан фрагмент программы: a=27;
   if a>0 then a=2*a else a=a+2;
   write (a);
   Что будет выведено на экран?

Задания.

1. Дано целое число. Если оно является четным, то прибавить к нему 7; в противном случае не изменять его. Вывести полученное число.
2. Дано целое число. Если оно является положительным, то прибавить к нему 11; в противном случае вычесть из него 2. Вывести полученное число.
3. Дано целое число. Если оно является положительным, то прибавить к нему 10; если отрицательным, то вычесть из него 3; если нулевым, то заменить его на 50. Вывести полученное число.
4. Даны две переменные целого типа: *A* и *B*. Если их значения не равны, то присвоить каждой переменной сумму этих значений, а если равны, то присвоить переменным нулевые значения. Вывести новые значения переменных *A* и *B*.
5. Даны три числа. Найти сумму двух наибольших из них.
6. Даны три переменные вещественного типа: *A, B, C*. Если их значения упорядочены по возрастанию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное. Вывести новые значения переменных *A, B, C*.
7. Даны координаты точки, не лежащей на координатных осях *OX* и *OY*. Определить номер координатной четверти, в которой находится данная точка.
8. Даны действительные числа x, y и z. Вычислить :
   max (x+y+z, xyz, x-y-z)
9. Даны три вещественных числа a, b и c. Определить имеется ли среди них хотя бы одна пара равных между собой чисел.
10. Дано двузначное число. Определить, входят ли в него цифры 3, 6 или 9.

11. Из действительных чисел X и Y оставить без изменения то, которое не принадлежит интервалу [-12, 24), а которое принадлежит - заменить нулем.

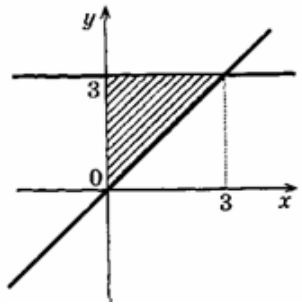
12. Если сумма трех попарно различных действительных чисел x, y и z меньше единицы, то наименьшее из этих трех чисел заменить полусуммой двух других; в противном случае заменить меньшее из x и y полусуммой двух оставшихся значений.

13. Даны две тройки вещественных чисел. В каждой тройке все числа различные. Найти среднее арифметическое средних чисел каждой тройки (средним назовем такое число в тройке, которое больше наименьшего числа из данной тройки, но меньше наибольшего).

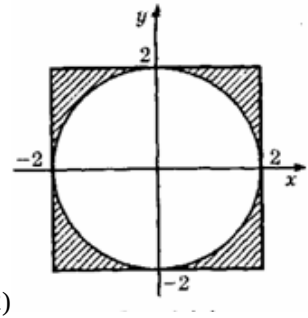
14. Определить лежит ли точка с координатами (X,Y) вне круга радиуса R с центром в точке (A,B) или внутри него.

15. Даны действительные X и Y . Определить, принадлежит ли точка с координатами (X,Y) заштрихованной области на рисунках 1)-4):

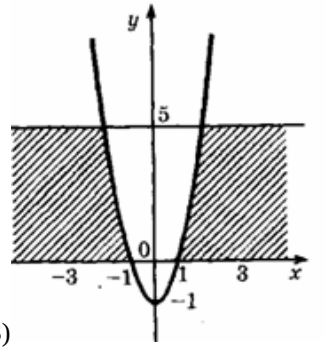
1)



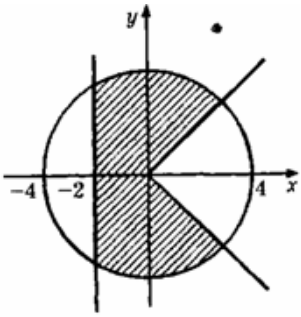
2)



3)



4)



Оператор выбора (варианта)

Для реализации разветвляющихся алгоритмов, содержащих более двух ветвей, в некоторых случаях удобно использовать специальный оператор выбора (варианта). Формат оператора:

Case <выражение-селектор> **of**
<список меток 1> : <оператор 1>;
<список меток 2> : <оператор 2>;
…….
<список меток n> : <оператор n>;
[else <оператор n+1>]
end;

Здесь: case, of, else, end – служебные слова;

<выражение-селектор> – выражение любого скалярного типа, кроме вещественного (часто просто имя переменной);

<список меток> - список разделенных запятыми значений, которые может принимать выражение – селектор, или одно его значение.

При выполнении оператора выбора вычисляется значение выражения - селектора. Затем выполняется оператор на той из ветвей, которая содержит метку, совпадающую с полученным значением. Если вычисленное значение не совпало ни с одной из меток, то:

а) выполняется ветвь *else* в случае полной формы оператора;

б) если ветви *else* нет (неполная форма), то управление передается оператору, следующему за *case* (оператор выбора работает ‘вхолостую’).

Пример 1. Составить программу, определяющую четность или нечетность остатка, полученного при делении целого положительного числа на 7. Если число делится на 7 без остатка, то также выдать соответствующее сообщение.

Возможны три случая:

1) остаток 0 – сообщение «делится нацело»;
2) остаток 2,4,6 – сообщение «остаток четный;
3) остаток 1,3,5 – сообщение «остаток нечетный».

```
program ostatok_vibor;
  var a : word;
begin
     write('введите целое положительное число  ');
     readln(a);
    case a mod 7 of
```

```
        2,4,6 : writeln('остаток четный');
        1,3,5 : writeln('остаток нечетный');
         0   : writeln('делится без остатка')
    end;
end.
```

Пример 2. Составить программу – простейший калькулятор, которая запрашивает знак арифметической операции (+,-,*,/), затем числа, над которыми нужно произвести действие, и выводит результат.

В данной программе возможны 4 ветви, причем в качестве условия, определяющего выбор той или иной ветви, выступает значение символьного типа (*char*) – знак арифметической операции. Поэтому будем использовать оператор *case*, в котором выражение-селектор – переменная *c* типа *char*. Для признака конца вычислений договоримся использовать знак точки:

```
program calculyator;
  label m1,m2;
  var c:char; x,y:integer; rez:real;
begin
  m1:write('введите знак операции (+,-,*,/)
          или . для конца вычислений ');
  readln(c);
  if c='.' then goto m2;
  write('введите числа-операнды');
  readln(x,y);
  case c of
      '+' : rez:=x+y;
      '-' : rez:=x-y;
      '* ': rez:=x*y;
      '/' : rez:=x/y;
  end;
  writeln('результат =', rez:5:2);
  goto m1;
  m2: writeln('До свидания!');
end.
```

Перечислимый и ограниченный типы

Простые (скалярные) типы данных языка Паскаль можно разделить на две группы:

стандартные (целый, вещественный, символьный, логический);

нестандартные (типы, определяемые пользователем).

То есть пользователь имеет возможность сам конструировать некоторые нестандартные типы данных для решения конкретных задач, а именно: перечисляемый (перечислимый) и ограниченный (интервальный, тип-диапазон).

Перечислимый тип данных задается списком всех возможных значений, которые могут принимать переменные этого типа. Э Значения перечисляются через запятую, заключаются в круглые скобки и могут быть объявлены либо в разделе типов (*type)*, либо в разделе переменных *(var)*. Например, при решении задач, связанных с датами, можно ввести следующие перечисляемые типы:

```
type
month=(jun,feb,mar,apr,may,june,july,aug,sep,oct,nov,dec);
  day=(mon,tue,wen,thu,fry,sat,sun);
```

и описать переменные типа *month* и *day*

```
var m1,m2 : month; d1,d2 : day;
```

Можно описать переменные m1,m2,d1,d2 непосредственно в разделе var:

```
var
m1,m2 : (jun,feb,mar,apr,may,june,july,aug,sep,oct,nov,dec);
d1,d2: (mon,tue,wen,thu,fry,sat,sun);
```

(Предпочтительнее использовать первый способ описания.)

Тот порядок, в котором перечислены значения, важен, так как фактически задает отношение порядка, при этом первое значение имеет порядковый номер 0, следующие – 1,2,3 и т.д.. Поэтому к переменным перечислимого типа применимы операции отношения, а также функции *pred, succ и ord.* Например, если выполнены приведенные выше описания, то в результате выполнения последовательности операторов:

```
m1:=(Pred(mar));
d1:=(Succ(fry));
writeln(Ord(wen));
writeln(ord(m1));
writeln(ord(d1));
```

на экран будут выведены (соответственно) числа 2,1,5.

Переменные перечислимого типа ***нельзя непосредственно вводить с клавиатуры и выводить на экран*** с помощью стандартных процедур ввода и вывода *read u write*. С этой целью обычно используется оператор выбора, который ставит в соответствие переменной перечислимого типа её порядковый номер. Этот номер используется в качестве выражения-селектора в операторе *case* при вводе. При выводе в качестве выражения-селектора используется само значение переменной перечислимого типа, а в действиях на каждой из меток стоит оператор *write* с соответствующим текстом.

Рассмотрим в качестве примера фрагмент программы, иллюстрирующий организацию ввода и вывода переменных перечислимого типа day_w - день недели:

```
Type day_w=(su,mo,tu,we,th,fr,sa);
Var d1:day_w; n:byte;
Begin
   Write('введите номер дня недели от 0 до 6');
   Readln(n);
   Case n of
     0: d1:=su;
1: d1:=mo;
2: d1:=tu;
3: d1:=we;
4: d1:=th;
5: d1:=fr;
6: d1:=sa;
   End;
  < рабочая часть программы- любые операторы>
{вывод результата}
   Case d1 of
    su: writeln('воскресенье');
    mo: writeln('понедельник');
    . . . . . .
    sa: writeln('суббота');
   end;
end.
```

Ограниченный тип данных (тип-диапазон) позволяет сузить интервал возможных значений переменных простых скалярных типов (как стандартных, так и перечислимого). Тип, из которого выбирается интервал значений, называется базовым типом для соответствующего

ограниченного типа. В качестве базового типа используется любой простой порядковый тип (все, кроме вещественного). Ограниченный тип задается либо в разделе *type* , либо в разделе *var* указанием наименьшего и наибольшего возможных значений через знак двоеточие, например:

```
type day_w=(su,mo,tu,we,th,fr,sa);
    day_o=tu..fr;
    bukva='a'..'z'; num=100..999;
var d1:day_o; c1,c2:bukva;
  n1,n2: num; n:1..9; c:'e'..'t';
```

Для переменных ограниченного типа в программах автоматически осуществляется контроль на выход значения переменной за границы диапазона, и в случае выхода выдается сообщение об ошибке. Это бывает удобно, когда такой контроль необходим (специально его программировать не надо, достаточно просто использовать ограниченный тип для соответствующей переменной).

Контрольные вопросы и задания.

Вопросы:

1. Каковы отличия оператора выбора case от условного оператора if?
2. Для чего служит выражение-селектор и какого он может быть типа?
3. Сколько меток может быть перед оператором в списке выбора?
4. Любое ли многократное ветвление можно реализовать с помощью оператора выбора?

Задания:

1. Составить программу, которая бы реализовала следующий алгоритм: переменной T присвоить значение true если сочетание день.месяц образует правильную дату, и значение false- иначе (учитывая количество дней в месяце и название месяца).
2. Составить программу, которая бы реализовала следующий алгоритм: по порядковому номеру дня года определить дату, т.е. число и месяц.
3. Мастям игральных карт присвоены порядковые номера: 1 — пики, 2 — трефы, 3 — бубны, 4 — черви. Достоинству карт, старших десятки, присвоены номера: 11 — валет, 12 — дама, 13 — король, 14 — туз. Даны два целых числа: N — достоинство ($6 \le N \le 14$) и M — масть карты ($1 \le M \le 4$). Вывести название соответствующей карты вида «шестерка бубен», «дама червей», «туз треф» и т.д..

4. Составить программу, которая бы присваивала переменной T значение true, если дата d1,m1 предшествует(в рамках года) дате d2,m2 и значение false иначе(d1 и d2-число,m1 и m2-месяц).

5. Дан номер месяца. Определить количество дней в этом месяце для невисокосного года.

6. Для целого числа K от 1 до 9 напечатать фразу "мне K лет", учитывая при этом, что при некоторых значениях K слово "лет" надо заменить на слово "год" или "года"

7. Для целого числа K в диапазоне 10-39 напечатать фразу "мы поймали K карасей", согласовав окончание слова "карась" с числом K.

8. Дано целое число в диапазоне 10–40, определяющее количество учебных заданий по некоторой теме. Вывести строку-описание указанного количества заданий, обеспечив правильное согласование числа со словами «учебное задание», например: 18 — «восемнадцать учебных заданий»,23 — «двадцать три учебных задания», 31 — «тридцать одно учебное задание».

9. Написать программу, которая по значению переменной перечисляемого типа, содержащему название страны, присваивает другой переменной перечисляемого типа название столицы этой страны.

10. Дан список дисциплин и номер семестра, когда они изучаются : История-2,1; Культурология-3,4; Философия-4,3; Ин.язык - 4,1,2,3. Составить программу, которая по номеру семестра выводит на экран список изучаемых дисциплин.

11. Робот может перемещаться в четырех направлениях:(«С» — север, «З» — запад, «Ю» — юг, «В» — восток) и принимать три цифровые команды: 0 — продолжать движение, 1 — поворот налево, –1 — поворот направо. Дан символ *C* — исходное направление робота и целое число *N* —посланная ему команда. Вывести направление робота после выполнения полученной команды.

12. Локатор ориентирован на одну из сторон света: («С» — север, «З» —запад, «Ю» — юг, «В» — восток) и может принимать три цифровые команды поворота: 1 — поворот налево, –1 — поворот направо, 2 — поворот на 180°. Дан символ *C* — исходная ориентация локатора и целые числа *N*1 и *N*2 — две посланные команды. Вывести ориентацию локатора после выполнения этих команд.

Лабораторная работа № 2

Тема: Условный оператор. Оператор выбора.

Цель:

1) Получение навыков в составлении и отладке разветвляющихся программ

2) Знакомство с перечислимыми ограниченными типами и оператором выбора.

Содержание отчета :

1) Постановка задачи для своего варианта
2) Список и назначение используемых переменных
3) Текст программы (для 1 задания - блок-схема и Паскаль, для задания 2-3 Паскаль)
4) Протокол отладки (тесты, результаты отладки на тестах).

Указания:

1) При отладке (то есть проверке правильности) любой программы, содержащей разветвления, обязательно необходимо проверить, как выполняется каждая из ее ветвей. Для этого необходимо запустить программу на выполнение столько раз , сколько существует различных ветвей, подбирая соответствующим образом исходные данные и проверяя правильность полученного результата. Зафиксировать полученные данные в тетради.

2) В задании 3 ввести и описать необходимый набор типов данных (перечислимый и ограниченный типы).

Задание 1.

Вычислить значение функции y, где

$$y = \begin{cases} f(x), & если \quad x\langle a; \\ g(x), & если \quad a\langle = x\langle b; \\ h(x), & если \quad x\rangle = b. \end{cases}$$

если $a = 0.2N;\ b = a + N;\ f(x) = 2x + N;\ g(x) = -3(N + x);$ $h(x) = N * N - X * X,$ где N- номер варианта.

Программу составить **двумя способами:**

а) вложенное полное ветвление;

б) последовательные неполные ветвления.

Задание 2.

Варианты заданий:

1. В чемпионате по футболу команде за выигрыш дается 3 очка, за проигрыш - 0, за ничью - 1. Известно количество очков, полученных командой за игру. Определить словесный результат игры (выигрыш, проигрыш, ничья).
2. Если сумма трех попарно различных действительных чисел x, y и z меньше единицы, то наименьшее из этих трех чисел заменить полусуммой двух других; в противном случае заменить меньшее из x и y полусуммой двух оставшихся значений.
3. Даны действительные числа a, b, c, d. Если a <= b <= c <= d, то каждое число заменить наибольшим из них; если a > b > c > d, то число оставить без изменения; в противном случае все числа заменяются их квадратами.
4. Год является високосным, если его номер кратен 4, однако из кратных 100 високосными являются лишь кратные 400 (например, 1700, 1800 и 1900 - не високосные года, а 2000 - високосный). Дано натуральное число n. Определить является ли високосным год с таким номером.
5. Даны четыре вещественных числа. Определить, сколько из них отрицательных.
6. Дано трехзначное число. Определить какая из его цифр является средним числом (средним будем называть число, которое больше минимального, но меньше максимального).
7. Дано двузначное число. Выяснить, различны ли его цифры. Если да, то проверить их на четность, иначе найти удвоенную сумму цифр.
8. Дано двузначное число. Если сумма его цифр четное число, то вывести половину этой суммы, иначе выяснить какая цифра нечетная, а какая четная.
9. Из данных действительных чисел a, b, c возвести в третью степень те, которые не принадлежат интервалу (4.2, 8.4).
10. Проверить, какому интервалу принадлежит данное действительное число d: (-16, 2), (4, 10) или (14, 36). Если число d не принадлежит не одному из интервалов, то выдать соответствующее сообщение.
11. Дано число h. Если оно принадлежит интервалу (8, 16), то выдать соответствующее сообщение, если же оно лежит в интервале (-12, 6), то напечатать его сигнатуру (сигнатура числа - это функция, равная -1, если число отрицательно, нулю, если число равно нулю и 1, если число положительно).
12. . Даны два вещественных числа. Условно принимая, что стандартной функции определения абсолютной величины числа нет, найти

полусумму абсолютных величин заданных чисел и квадратный корень из произведения абсолютных величин заданных чисел.

Задание 3.

1. Игральным картам условно присвоены следующие порядковые номера в зависимости от их достоинства: "валету" - 11, "даме" - 12, "королю" - 13, "тузу" - 14. Порядковые номера остальных карт соответствуют их названию. По заданному номеру карты k (6 <= k <= 14) определить достоинство соответствующей карты.
2. По названию введенной физической величины, характеризующей движение тела (координата, скорость, ускорение, время, сила) вывести ее единицу измерения.
3. Вводится одно любое слово из предложения: "Каждый охотник желает знать где сидит фазан". Вывести соответствующий этому слову цвет радуги.
4. Составить программу, которая вводит обозначение химического элемента из второго периода (Li, Be, B, C, N, O, F, Ne) и выводит его полное русское название.
5. Составить программу, которая в зависимости от порядкового номера месяца (1, 2,...,12) выводит на экран его название (январь,...,декабрь).
6. В китайском календаре годы носят название животных: Змеи, Лошади, Овцы, Обезьяны, Курицы, Собаки, Свиньи, Крысы, Коровы, Тигра, Зайца и Дракона. Каждые 12 лет цикл начинается заново. Считая, что 2000 год был годом Дракона, написать программу, которая вводит номер года от 2000 до 2012 и выводит его название по китайскому календарю.
7. В зависимости от введенного номера n (1<=n<=10) вывести букву латинского алфавита, соответствующего этому номеру.
8. Составить программу, которая в зависимости от порядкового номера пальца руки (1, 2, 3, 4, 5) выводит на экран его название(большой,указательный, средний, безымянный, мизинец).
9. Составить программу, которая в зависимости от названия столицы (Москва, Париж, Берлин, Лондон, Вашингтон) выводит на экран название соответствующего государства (Россия, Франция, Германия, Англия, США).
10. Составить программу, которая в зависимости от введенной цифры (0, 1, ... ,9) выводит на экран ее римское представление.
11. Составить программу, которая в зависимости от введенной цифры (0, 1, ... ,9) выводит ее словесное название (нуль, один, ... , девять).

12. Герои русской народной сказки "Репка" пронумерованы следующим образом: 0 - Репка, 1 - Дед, 2 - Бабка и т.д. ..., 7 - Мышка. Составить программу, которая по введенному номеру выводит название персонажа.

Циклические алгоритмы и программы

Циклом *называется многократно повторяющийся фрагмент алгоритма или программы*. Те действия, которые повторяются, называются **телом цикла (ТЦ).**

В программировании различают три типа циклов:

1) с предусловием («пока»);

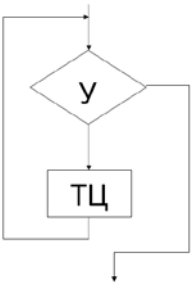
2) с постусловием («до»);

3) с параметром («для»).

Первые два типа циклов объединяют в одну группу, называемую *циклами «по условию»*, или циклами, число повторений которых не известно перед началом выполнения цикла, а *определяется истинностью (ложностью) некоторого условия.* Переменная, от значения которой зависит истинность условия (следовательно, и окончания цикла) называется ***управляющей переменной цикла***. Третий тип цикла называют *циклом с заранее известным числом* повторений – количество повторений определено в законе изменения управляющей переменной, которая носит специальное название – ***параметр.*** Параметр изменяется в заданном диапазоне с определенным шагом.

Рассмотрим блок-схемы циклов каждого типа, обобщенный формат их записи на языке Паскаль и алгоритмы их работы.

1) цикл с предусловием («пока»);

Блок-схема	Паскаль
	**while <У> do** **<ТЦ>;**

Здесь **У –** условие (булевское выражение), **ТЦ –**тело цикла.

При выполнении цикла «пока» сначала вычисляется значение булевского выражения. Если оно принимает значение **True** , то выполняется тело цикла и снова вычисляется значение булевского выражения. Если оно принимает значение **False**, то цикл заканчивается. То есть **тело цикла повторяется до тех пор, пока булевское выражение истинно.** Как только оно станет ложно – цикл закончится.

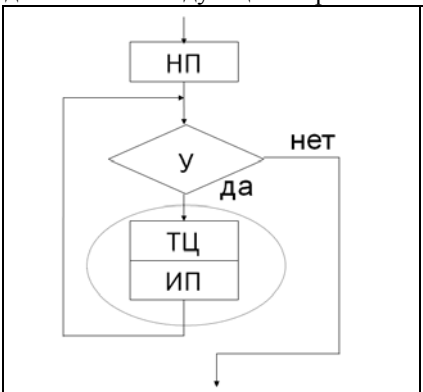
Возможна такая ситуация, что тело цикла не выполнится ни разу, а именно, в том случае, если булевское выражение сразу примет значение *False.*

В теле цикла и булевском выражении присутствует переменная, называемая управляющей переменной цикла. Перед началом цикла ей задается некоторое значение – начальное присваивание, а ***в теле цикла обязательно должен быть оператор, изменяющий значение управляющей переменной по некоторому закону.*** Обычно этот оператор – последний в теле цикла. Если значение управляющей переменной не изменять, то возникает ситуация зацикливания – тело цикла будет повторяться бесконечное число раз, так как не будет изменяться значение зависящего от этой переменной булевского выражения, определяющего условие продолжения (окончания) цикла.

Поэтому на начальных этапах изучения программирования для уменьшения количества ошибок в составлении циклических программ с методической точки зрения целесообразно выделить еще два блока, образующих этот цикл:

1. блок начальных присваиваний (**НП**);
2. блок изменения управляющей переменной цикла (**ИП**);

С учетом этого в самом общем виде структура цикла пока на языке блок-схем и соответственно Паскале для вычислительных алгоритмов представляется следующим образом:

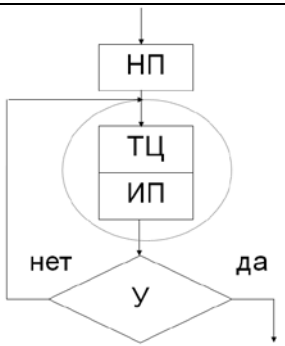



```
<НП>
while <У> do
 begin
      <ТЦ>
       <ИП>
 end;
```

Анализ задач на составление алгоритмов с циклом «пока» целесообразно проводить по схеме:

1. Определить действия, образующие тело цикла **(ТЦ)** и выделить в них изменяющуюся переменную.
2. Проанализировать блоки начальных присваиваний **(НП)** и изменения управляющей переменной **(ИП).**
3. Записать условие продолжения **(У)** и условие окончания цикла.

2) цикл с постусловием («до»);

Блок-схема	**Паскаль**
	**<НП>** **repeat** **<ТЦ>** **<ИП>** **until <У> ;**

При выполнении цикла «до» сначала выполняется тело цикла, а потом вычисляется значение булевского выражения. Если значение булевского выражения ложно, то снова выполняется тело цикла, если истинно – то цикл заканчивается. То есть **тело цикла повторяется до тех пор, пока булевское выражение ложно.** Как только оно станет истинным – цикл закончится.

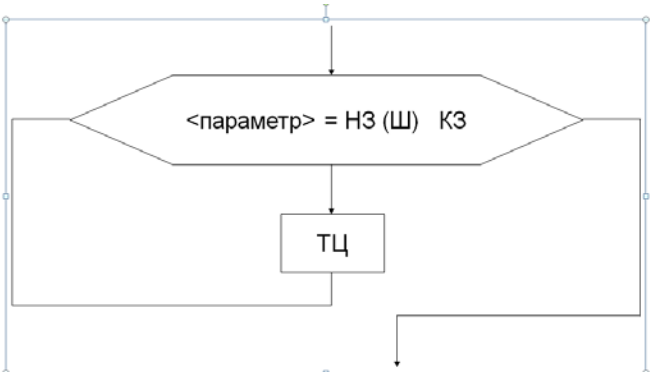
В отличии от цикла «пока» в этом цикле тело цикла выполняется хотя бы один раз всегда. Еще одно отличие – ***цикл «пока »*** повторяется до тех пор, пока условие ***истинно,*** а цикл **«до»** - пока условие ***ложно***. ***То есть в цикле «пока»*** **после ключевого слова while записывается условие продолжения цикла,** ***а в цикле «до»*** **после ключевого слова until записывается условие окончания цикла.** В остальном эти циклы схожи.

3) цикл с параметром («для»)

Следует заметить, что цикл с параметром в разных языках программирования реализован по-разному. В языке Паскаль накладываются довольно-таки жесткие ограничения на переменную-параметр цикла и шаг его изменения. Поэтому здесь мы рассмотрим сначала общую форму цикла «для» на языке блок-схем и алгоритм его работы, а особенности

его реализации в языке Паскаль будут рассмотрены в следующем параграфе.

Блок-схема



Напомним, что вытянутый шестиугольник на языке блок-схем означает блок модификации параметра.

Здесь

ТЦ - тело цикла***;***

<параметр> - это переменная, изменяющаяся на некотором интервале (диапазоне) по заданному закону (играет роль управляющей переменной для этого типа цикла);

НЗ и ***КЗ*** - начальное и конечное значения диапазона изменения параметра, причем начальное значение может быть как меньше конечного, так и больше;

Ш – шаг изменения параметра, может быть как положительным, так и отрицательным.

Алгоритм выполнения цикла "для":

Обозначим р-параметр

1. **р:= НЗ**;
2. **р внутри отрезка [НЗ, КЗ] ?**
   если **"да",** то к **п.3**, если **"нет"**, то **конец цикла;**
3. выполняется **ТЦ**
4. **р:= р+шаг**;
5. **К п.2;**

Замечания

1) Следует обратить внимание на связь между значениями параметра и количеством повторений тела цикла

n :=целая часть**(abs(КЗ - НЗ))/шаг+1.**

2) В блоке модификации (изменения) параметра цикла по сути дела объединены блоки **НП, ИП и У** циклов с потсусловием и предусловием (автоматически реализуется механизм изменения управляющей переменной и работы цикла). Поэтому существует правило: **в теле цикла параметр цикла изменять нельзя!**

Реализация циклических алгоритмов в языке Паскаль. Примеры.

Рассмотрим более подробно реализацию каждого из типов циклов в языке Паскаль (теоретически и на примерах).

Цикл с параметром («для»)

Цикл с параметром в Паскале имеет две формы:

1) ***for*** *<параметр>*:= *<НЗ>* ***to*** *<КЗ>* ***do*** *<ТЦ>*;
2) ***for*** *<параметр>*:= *<НЗ>* ***downto*** *<КЗ>* ***do*** *<ТЦ>*;

Здесь *<ТЦ>* (тело цикла) – простой или составной оператор. Если он составной, то нужно не забывать ставить операторные скобки ***begin-end;***

<параметр> - это переменная любого простого скалярного типа, кроме вещественного;

<НЗ> и *<КЗ>* - начальное и конечное значения диапазона изменения параметра, причем начальное значение может быть как меньше конечного, так и больше:

to **-** используется, если *НЗ< КЗ,*

downto **–** используется, если *НЗ> КЗ.*

Соответственно, при использовании служебного слова ***to*** при каждом следующем повторении цикла параметр изменяется по закону: **<*параметр*>:= succ(<*параметр*>);**

При использовании ***downto*** **–** по закону:

<*параметр*>:= pred(<*параметр*>).

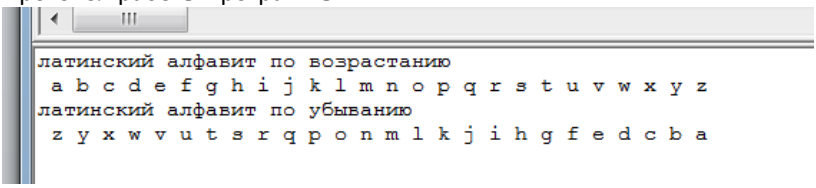
Очевидно, что в том случае, когда параметром цикла является переменная целого типа, значение параметра изменяется либо с шагом +1 (***to)***, либо -1 (***downto),*** то есть параметр выступает в качестве счетчика числа повторений цикла.

Таким образом, тело цикла выполняется столько раз, сколько значений элементов соответствующего параметру типа лежат в диапазоне (на отрезке) *<НЗ>* - *<КЗ>*.

Пример 1. Составить программу, последовательно выводящую на экран малые буквы латинского алфавита: сначала от начала к концу алфавита, а потом – наоборот:

```
Program cikl_param;
  var c:char; {с-параметр цикла}
Begin
  Writeln('латинский алфавит по возрастанию');
  For c:='a'to 'z' do write(c:2);{отводим 2 позиции на символ, чтобы между буквами был пробел}
  Writeln;{переводим строку}
  Writeln('латинский алфавит по убыванию');
  For c:='z' downto 'a' do write(c:2);
  Writeln
end.
```

Протокол работы программы:



Пример 2. Составить программу построения таблицы значений функции $y = \sin(x)$ на отрезке $[a,b]$ с шагом h.

В данном примере требуется вычислить и вывести на экран значения функции $y = \sin(x)$ при x=a, a+h, a+2h и т.д.. То есть независимая переменная x изменяется от *НЗ=a* до *КЗ=b* с некоторым шагом h. Эту переменную можно было бы взять в качестве параметра, если бы она была целого типа и шаг был бы равен +1 или -1. В общем же случае x – переменная вещественного типа и шаг отличен от 1. Однако можно подсчитать количество точек N в таблице – определить, сколько шагов длиной h укладывается на отрезке [a,b] и добавить 1 (точек на 1 больше, учитывая крайние две):

N=trunc((b-a)/n) + 1;

(здесь функция trunc выделяет целую часть в том случае, когда длина отрезка не кратна величине шага).

Для решения задачи будем использовать цикл с параметром i – счетчиком количества узловых точек таблицы, изменяющимся от 1 до N. До начала цикла зададим x *начальное значение a* (***x:=a***) и после каждого повторения тела цикла будем увеличивать значение x на шаг, то есть *изменять по закону*: ***x:=x+h.***

Значения переменных *a,b,h* вводим с клавиатуры.

```
Program tab_func;
  var x,y,a,b,h : real; i,n : integer;
begin
  write('введите границы отрезка и шаг: ');
  readln(a,b,h);
  n:= trunc((b-a)/h) + 1;
  x:=a;{начальное присваивание}
  for i:=1 to n do
    begin {начало тела цикла}
      y:=sin(x);{вычисление функции}
      writeln(x:6:2, y:6:2);{вывод на экран}
      x:=x+h; {изменение значения x}
    end;{конец цикла}
end.
```

Протокол работы программы:

```
введите границы отрезка и шаг: -2 2 0.5
 -2.00 -0.91
 -1.50 -1.00
 -1.00 -0.84
 -0.50 -0.48
  0.00  0.00
  0.50  0.48
  1.00  0.84
  1.50  1.00
  2.00  0.91
```

Циклы с предусловием («пока») и постусловием («до»)

Как отмечено выше, в том случае, когда некоторую последовательность действий надо повторять до тех пор, пока истинно (или ложно)

некоторое условие, используются циклы типа «пока» или «до». При этом заранее не известно, сколько повторений будет выполнено.

Цикл «пока» в языке Паскаль имеет следующий ***формат:***

while <*булевское выражение*> ***do*** <*ТЦ*>***;***

здесь

<*ТЦ*>- тело цикла – простой или составной оператор (в случае составного не забывайте использовать операторные скобки).

Пример1. Вычислить сумму S=1+2+3+4+… . Вычисления прекратить, как только значение S станет больше заданного числа x (x вводится с клавиатуры). Дополнительно определить, сколько слагаемых просуммировано.

Согласно рассмотренной в предыдущем параграфе схеме, анализ циклических алгоритмов следует начинать с выяснения того, какие действия необходимо повторять, то есть какие действия образуют тело цикла. ***Данная задача относится к обширному классу типовых задач программирования*** – задач на вычисление сумм (произведений). При вычислении сумм в теле цикла всегда к значению суммы добавляется одно (очередное) слагаемое с помощью присваивания вида **S:=S+<слагаемое>.** Для конкретной суммы обычно это слагаемое является выражением, зависящим от значения некоторой переменной величины. Поэтому следует записать тело цикла и продумать, как будет изменяться участвующая в теле цикла независимая переменная величина при каждом новом повторении цикла.

В нашей задаче к сумме добавляется сначала ***1,*** потом ***2***, потом - ***3*** и т.д., то есть в общем виде – переменная ***k***, которая при каждом повторении цикла увеличивается на 1. То есть тело цикла имеет вид **S:=S+k.** До начала цикла следует выполнить начальное присваивание ***k:=1,*** а перед каждым новым повторением цикла изменять ее по закону ***k:=k+1.*** Условие окончания цикла: ***S>x,*** а соответственно условие продолжения - ***S<=x*** (взаимно противоположное условие):

```
Program pr1_cikl_poka;
  Var x,S,k : integer;
Begin
 Write('введите x '); readln(x);
 S:=0;{сумму всегда обнуляем перед циклом}
 k:=1; {первое слагаемое}
   while S<=x do
     begin
       S:=S+k;
```

```
      k:=k+1
    end;
  writeln('сумма = ',S, 'число слагаемых ',k-1);
end.
```

Замечание: *Обратите внимание, что значение k по окончании цикла на единицу больше, чем число просуммированных слагаемых. Можно было значению k до начала цикла присвоить 0, тогда в теле цикла сначала бы надо было увеличивать k на единицу, а потом добавлять его к сумме. По окончании цикла число слагаемых было бы равно k (самостоятельно составьте этот вариант программы).*

Пример 2. Составить программу, подсчитывающую сумму цифр некоторого натурального числа N.

Количество цифр в числе неизвестно, поэтому в теле цикла мы будем выполнять следующие действия:

1) выделять из числа крайнюю правую цифру при помощи операции c:=N mod 10;
2) добавлять её к сумме: S:=S+c;
3) «отбрасывать» эту цифру при помощи оператора N:=N div 10.

Продолжать этот процесс будем до тех пор, пока число N не станет равным 0.

Таким образом, необходимо организовать цикл «пока», условием продолжения которого является ненулевое значение числа N, то есть условие N<>0 (условие окончания – N=0).

```
Program sum_cifr;
  Var N:longint; c,s:byte;
Begin
  Write('Введите число ');
  Readln(N);
  S:=0;{начальное присваивание для суммы}
  While N<>0 do
    Begin
      C:=N mod 10;
      S:=s+c;
      N:=N div 10
    End;
  Writeln('сумма цифр равна ',s)
End.
```

Теперь рассмотрим второй цикл – с постусловием («до»).
Напомним, что ***цикл «до»*** в языке Паскаль имеет следующий ***формат:***

repeat
<тело цикла>
until *<булевское выражение>****;***

здесь ключевое слово ***repeat*** переводится – повторять, ***until***-до:

Тело цикла повторяется до тех пор, пока булевское выражение ложно. *Как только оно станет истинным – цикл закончится.* В отличии от цикла «пока» в этом цикле тело цикла выполняется хотя бы один раз всегда. В *цикле «пока» после ключевого слова* ***while*** *записывается условие продолжения цикла, а в цикле «до» после ключевого слова* ***until*** *записывается условие окончания цикла.*

Пример 3. В качестве примера проиллюстрируем, как рассмотренный выше **пример 1** реализуется с помощью цикла «до», внося по ходу программы операторы-комментарии:

```
 Program pr1_cikl_do;
   Var x,S,k : integer;
Begin
  Write('введите x '); readln(x);
  S:=0;{сумму всегда обнуляем перед циклом}
  k:=0; {подготовка слагаемых}
     repeat {начинаем цикл}
        k:=k+1;{увеличиваем слагаемое на 1}
        S:=S+k;{добавляем к сумме}
     until S>x ;{условие окончания цикла}
  writeln('сумма = ',S, ' число слагаемых ',k);
end.
```

Замечание: *Ключевые слова* ***repeat-until*** *играют роль операторных скобок для тела цикла.*

Пример 4. Дано действительное число x. Составить программу, вычисляющую произведение:

$$P=x*(x+0.2)*(x+0.4)*(x+0.6)*\ldots*(x+1.8)*(x+2)$$

Для накапливания (вычисления) произведений в теле цикла в общем случае используем оператор вида:

P:=P*<очередной сомножитель>

В данном примере: **P:=P*(x+a),** где a – переменная величина, которая для первого сомножителя равна 0, для второго – 0.2, для третьего – 0.4 и т.д.. То есть она изменяется по закону: ***a:=a+0.2***, конечное значение рав-

но 2. Начальные присваивания: для произведения P:=1, для управляющей переменной a:=0. Условие окончания цикла: ***a>2***.

```
program proizv_cikl_do;
  Var a,P,x: real;
begin
  Write('введите x='); readln(x);
  P:=1; a:=0;
      Repeat
        P:=P*(x+a);
        a:=a+0.2
      until a>2;
    writeln(P, a)
end.
```

Вложенные циклы

В программах часто возможны такие случаи, что телом какого-либо цикла так же является цикл, его телом – снова цикл и т.д., то есть один цикл содержит внутри себя другой цикл и т.д.. Такие циклы называются ***вложенными.*** Цикл, целиком содержащийся внутри другого цикла, называется ***внутренним***, а цикл, содержащий в себе другой цикл – ***внешним.*** Как внешний, так и внутренний циклы могут быть циклами любых видов.

Рассмотрим пример. Пусть требуется вывести на экран таблицу умножения чисел от 2 до 9 (таблицу Пифагора):

1	2	3	4	5	6	7	8	9
2	4	6	8	……			16	20
3	6	9	12	……..			24	27
……								
8	16	24	32	……			64	72
9	18	27	36	……			72	81

В этой таблице номер строки обозначим переменной ***i*** (*1,2, … ,9*) , а номер столбца – переменной ***j*** (*1,2, … ,9*). На пересечении *i* -ой строки и *j* -го столбца стоит число, равное произведению ***i*j***. Для того, чтобы получить одну строку этой таблицы (***i***-ю), мы должны для ***j,*** изменяющегося от 1 до 9, выполнить умножение ***i*j***, то есть организовать цикл с параметром ***j***. Всего строк будет 9. Поэтому для того, чтобы получить все строки таблицы, нам надо организовать цикл с параметром ***i***, а телом этого цикла будет цикл с параметром ***j***. Таким образом, мы приходим к

конструкции вложенных циклов: внешний по параметру ***i***, внутренний по параметру ***j***.

Телом внутреннего цикла будет оператор вывода произведения ***i*j***, то есть вывод на экран одного числа таблицы. Внутренний цикл выводит на экран одну строку таблицы, состоящую из 9 чисел, при фиксированном значении параметра внешнего цикла ***i*** (номера строки). Благодаря изменению номеров строк во внешнем цикле мы получим на экране 9 строк, то есть всего тело внутреннего цикла выполнится 81 раз. По окончании внутреннего цикла (перед переходом к повторению внешнего) необходимо перевести строку на экране при помощи оператора *writeln:*

```
program tab_Pifagor;
  var i,j: 1..9;
begin
  for i:=1 to 9 do {внешний цикл}
    begin
      for j:=1 to 9 do {внутренний цикл}
        write(i*j:4); {конец внутреннего цикла}
      writeln; {перевод строки}
    end; {конец внешнего цикла}
end.
```

Контрольные вопросы и задания.

Вопросы:

1. Что называется циклом? Тело цикла.
2. Какие типы циклов бывают?
3. Как программируются циклические алгоритмы с заранее известным числом повторений цикла?
4. Как программируются циклические алгоритмы с заранее неизвестным числом повторений цикла?
5. В чем проявляется ограниченность цикла FOR?
6. В чем сходства и отличия оператора WHILE и оператора REPEAT?

Задания:

1. Даны целые числа K и N ($N > 0$). Вывести N раз число K.
2. Даны два целых числа A и B ($A < B$). Вывести в порядке возрастания все целые числа, расположенные между A и B (включая сами числа A и B), а также количество N этих чисел.

3. Даны два целых числа A и B ($A < B$). Вывести в порядке убывания все целые числа, расположенные между A и B (не включая числа A и B), а также количество N этих чисел.

4. Дано вещественное число — цена 1 кг яблок. Вывести стоимость 1,2, …, 10 кг яблок.

5. Дано вещественное число — цена 1 кг печенья. Вывести стоимость 0.1, 0.2, …, 1 кг печенья.

6. Дано целое число N (> 0). Найти сумму $1 + 1/2 + 1/3 + \ldots + 1/N$

7. Вычислить суммы

$$S = \sum_{i=1}^{100} \frac{1}{3i+2} \qquad S = \sum_{i=1}^{100} \frac{1}{i^2}$$

8. Дано целое число N (> 0). Найти квадрат данного числа, используя для его вычисления следующую формулу: $N^2 = 1 + 3 + 5 + \ldots + (2 \cdot N - 1)$.

9. Дано вещественное число A и целое число N (> 0). Найти A в степени N: $A^N = A \cdot A \cdot \ldots \cdot A$ (числа A перемножаются N раз).

10. Дано целое число N (> 0). Найти произведение $N! = 1 \cdot 2 \cdot \ldots \cdot N$ (*N–факториал*). Чтобы избежать целочисленного переполнения, вычислять это произведение с помощью вещественной переменной и вывести его как вещественное число.

11. Найти сумму $2^2+2^3+2^4+ \ldots +2^{10}$. Операцию возведения в степень не использовать.

12. Определить количество натуральных чисел из интервала от 100 до 500, сумма цифр которых равна 15.

13. Дано натуральное число. Определить сумму его делителей.

14. Найти все двузначные числа, которые делятся на 7 или содержат цифру 7.

15. Напечатать таблицу значений функции y=sin(x) на отрезке [-3,3] с шагом 0,25

16. Дано целое число N (> 0). Используя один цикл, найти сумму $1! + 2! + 3! + \ldots + N!$

17. Дано вещественное число X и целое число N (> 0). Найти значение выражения

$$1 + X + X^2/(2!) + \ldots + X^N/(N!)$$

18. Дано вещественное число X и целое число N (> 0). Найти значение выражения

$$X - X^3/(3!) + X^5/(5!) - \ldots + (-1)^N \cdot X^{2N+1}/((2 \cdot N+1)!)$$

19. Дано целое число N (> 0). Найти наименьшее целое положительное число K, квадрат которого превосходит N.

20. Дано целое число N (> 1). Найти наибольшее целое число K, при котором выполняется неравенство $3^K < N$.

21. Дано целое число N (> 1). Вывести наименьшее из целых чисел K, для которых сумма $1 + 2 + \ldots + K$ будет больше или равна N, и саму эту сумму.

22. Дано целое число N (> 0). Найти число, полученное при прочтении числа N справа налево.

23. Дано целое число N (> 0). Определить, имеется ли в записи числа N цифра «4».

24. Дано целое число N (> 0). Определить, имеются ли в записи числа N нечетные цифры.

25. Дано целое число N (> 1). Определить, является ли оно *простым*, то есть не имеет положительных делителей, кроме 1 и самого себя

26. Даны целые положительные числа A и B. Найти их *наибольший общий делитель* (НОД), используя *алгоритм Евклида*:

27. Дано целое число N (> 1). Последовательность *чисел Фибоначчи* определяется следующим образом: $F_1 = 1$, $F_2 = 1$, $F_k = F_{k-2} + F_{k-1}$, где $K = 3, 4, \ldots$. Проверить, является ли число N числом Фибоначчи.

28. Дано целое число N (> 1). Найти первое число Фибоначчи, большее N.

29. Дано целое число K и набор ненулевых целых чисел; признак его завершения — число 0. Вывести количество чисел в наборе, меньших K.

30. Дано целое число K и набор ненулевых целых чисел; признак его завершения — число 0. Вывести номер первого числа в наборе, большего K. Если таких чисел нет, то вывести 0.

Лабораторная работа № 3

Тема: Организация циклов в программе.
Цель: Получение навыков в использовании циклов

Содержание отчета для каждого задания:

1. Постановка задачи.
2. Словесное описание идеи структуры алгоритма, обоснование выбора типа цикла.
3. Текст программы.
4. Протокол отладки (тесты, результаты отладки на тестах).

Задание 1. Используя все три вида операторов цикла составить программы табулирования функции $y = f(x)$ на отрезке [a, b] с шагом $h = (b - a)/m$, где m – заданное число.

Вариант	Функция	a	b	m
1	xsin(x)	0	3π	10
2	sin (1/x)	π/8	2/π	15
3	cos (1/x)	π/4	4/π	20
4	$\sin(x^2)$	π/6	2π/3	10
5	$\cos(x^2)$	π/3	3π/2	15
6	sin(x)+tg(x)	0	π/4	20
7	cos(x)+ctg(x)	π/4	π/2	10
8	tg(x/2)	0	2π/3	15
9	tg(x/2)+cos(x)	π/2	π	20
10	ctg(x/3)+sin(x)	π/4	π/2	10
11	x^2+sin(x)	-2π	2π	20
12	x^3+exp(x)	-9	3	12

Задание 2.

Используя операторы цикла с предусловием и постусловием, найти сумму ряда с точностью $\varepsilon=10^{-3}$ и $\varepsilon=10^{-4}$, общий член которого a_n (см.вариант). Определить, сколько членов ряда просуммировано.

Вариант	a_n	Вариант	a_n
1	$(2n-1)/2^n$	6	$2^n/n!$
2	$10^n/n!$	7	$(-1)^n/n!$
3	n!/(2n)!	8	$(-1)^n 3^{2n}/(2n)!$
4	$3^n n!/(3n)!$	9	$5^{2n+1}/(2n+1)!$
5	$(-1)^n$ (n+1)/n!	10	$2^n n!/(2n)!$
6	$n^2/3^n$	12	(n-3)/(2n+1)!

Указание: 1. Считать, что точность достигнута, если $abs(a_n) < \varepsilon$.

2. Для получения следующего члена ряда использовать рекуррентную формулу, выражающую a_n+1 через a_n. Для этого вычислить их отношение.

Например, $an = 2(n!)^2 / (3(2n)!)$

$$\frac{a_{n+1}}{a_n} = \frac{2((n+1)!)^2 3(2n)!}{3(2n+2)!2(n!)^2} = \frac{2(n!)^2 (n+1)^2 3(2n)!}{3(2n)!(2n+1)(2n+2)2(n!)^2} = \frac{(n+1)^2}{(2n+1)2(n+1)} = \frac{n+1}{2(2n+1)}$$

Откуда, $a_{n+1} = a_n (n+1)/(2(2n+1))$.

Задание 3 (по вариантам)

1. Из чисел от 10 до 99 вывести те, сумма цифр которых равна n и само число делится на m.
2. Найти все натуральные двухзначные числа, которые делятся на n или содержат цифру n.
3. Среди натуральных трехзначных чисел найти те, сумма цифр которых равна заданному числу A, а само число при делении на 4 дает остаток 3.
4. Среди натуральных трехзначных чисел найти те, сумма квадратов цифр которых делится на A, а само число делится на A+1.
5. Найти все натуральные трехзначные числа, сумма цифр которых кратна заданному натуральному числу B и само число также делится на B.
6. Среди четырехзначных натуральных чисел выбрать те, у которых все четыре цифры различны, а само число кратно 5.
7. Найти все трехзначные натуральные числа, сумма цифр которых равна B, а само число состоит из разных цифр.
8. Найти все четырехзначные натуральные числа, у которых сумма крайних цифр равна сумме средних цифр, а само число делится на число A.
9. Найти все трехзначные натуральные числа, которые состоят из разных цифр, а само число делится на 3.
10. Найти все симметричные четырехзначные натуральные числа (например, 7557,1221), которые делятся на заданное число A.
11. Найти все натуральные трехзначные числа, сумма крайних цифр которых равна заданному натуральному числу A, а само число делится на A.
12. Среди четырехзначных натуральных чисел выбрать те, у которых все цифры четные, а их сумма делится на 8.

Раздел 4. ПРОЦЕДУРЫ И ФУНКЦИИ. МОДУЛИ. ПРОГРАММИРОВАНИЕ ГРАФИКИ

Процедурное программирование. Понятие подпрограммы.

Языки программирования, ориентированные на операционную систему MS DOS, относятся к категории процедурных языков программирования (ALGOL, Фортран, Бейсик (MSX, GW, Q), Паскаль, Си). Программы, написанные на процедурных языках, представляют собой последовательность операторов – инструкций, которые выполняются одна за другой по тексту программы. Для небольших программ при этом не требуется какой-то дополнительной организации, но если размер программы большой, то ее текст становится громоздким и неудобным для восприятия человеком - **неструктурированным. Для структурирования** таких программ их разбивают на **отдельные блоки, которые называют *подпрограммами*.**

Каждая **подпрограмма имеет свое имя** и выполняет некоторую законченную последовательность действий. Если последовательность действий выполняется над некоторыми переменными, то после имени подпрограммы указывают **имена этих переменных - *параметры***. Обращение к подпрограмме (ее вызов) происходит по имени, после которого указывают значение параметра.

В небольших программах подпрограммы удобно использовать в тех случаях, когда одинаковые действия, выполняемые группой операторов над значениями некоторых переменных, требуется выполнить несколько раз в различных местах программы. В этом случае использование подпрограмм дает возможность сократить текст программы, т.к. соответствующая последовательность действий записывается один раз, а вызывается несколько раз (рис.4.1):

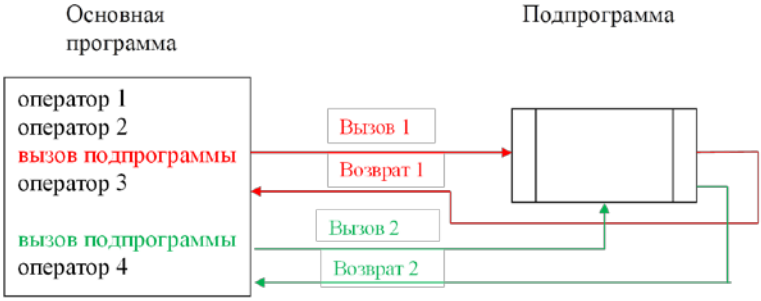


Рис.4.1 Взаимодействие основной программы и подпрограммы

В некоторых языках программирования вместо термина *подпрограмма* используется термин *процедура* (Pascal). Кроме того, частным случаем подпрограммы (процедуры) является ***функция.*** *Функция – это подпрограмма, в результате работы которой получается одно скалярное значение и оно присваивается имени функции.*

В процедурном программировании существуют два типа данных (переменных): **ЛОКАЛЬНЫЕ и ГЛОБАЛЬНЫЕ**. Переменные, описанные в основной программе, называют **глобальными** переменными. Переменные, описанные в подпрограмме (процедуре), называются **локальными** переменными.

Локальные данные находятся внутри подпрограммы и предназначены для использования только этой подпрограммой. Локальные данные недоступны никому, кроме самой подпрограммы (ни основной программе, ни другой подпрограмме) и не могут быть изменены другими подпрограммами. Если возникает необходимость совместного использования одних и тех же данных несколькими подпрограммами, то такие данные должны быть объявлены как глобальные. Любая подпрограмма имеет доступ к глобальным данным.

Проиллюстрируем это (рис. 4.2):

схематически:

Рис.4.2 Локальные и глобальные переменные

Допускается любой уровень вложенности процедур и функций. Любая программа, процедура и функция представляет собой блок со своей областью описаний и может содержать внутри этого блока описание других процедур и функций, а так же обращений к ним.

Блок, содержащий в своем разделе описаний другой блок, называется внешним. Блок, содержащийся в разделе описаний другого блока, называется внутренним или подблоком.

Объекты, описанные внутри какого-либо подблока, являются по отношению к нему локальными и недоступны внешним блокам. Объекты, описанные в некотором внешнем блоке, доступны и могут быть использованы в любом его подблоке, то есть они являются глобальными по отношению к этим подблокам. Только объекты, локальные для некоторого блока, являются глобальными для всех его подблоков (под объектами понимаются имена констант, переменных, типов процедур и функций).

Схематически изобразим структуру блоков некоторой программы (рис.4.3):

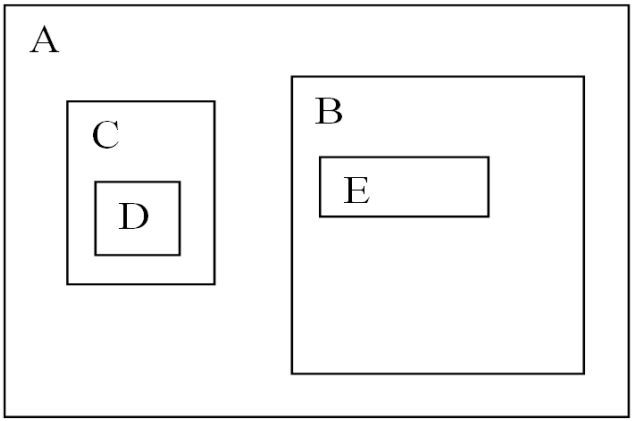


Рис. 4.3 Вложенность процедур и функций

Здесь A – основная программа, B,C,D,E – подпрограммы (процедуры или функции).

Переменные, описанные в A, являются глобальными по отношению ко всем процедурам и функциям, описанные в C - глобальные для D и т.д.. Процедура E доступна в B, но недоступна в C и т.д.

Различают **стандартные (встроенные) процедуры и функции, и процедуры и функции, определяемые пользователем.** Стандартные процедуры и функции содержатся в библиотечных модулях системы программирования. В языке Pascal (Delphi) эти модули подключаются к программе при помощи ключевого слова

uses <имя модуля>.

Каждый модуль содержит в себе набор процедур и функций для решения определенного класса задач. Например, в языке Pascal стандартные арифметические функции содержатся в модуле **system**, который автоматически подключается к любой программе. Модуль **crt** содержит

процедуры и функции для работы с текстовым режимом экрана и звуком. Модуль **graph (graphABC)** содержит процедуры и функции для работы с графикой. Соответственно для их подключения в программе на Pascale необходимо записать после имени программы следующую строчку:

program <имя программы>;
uses crt, graphABC;

Кроме стандартных процедур и функций пользователь может разрабатывать свои процедуры и функции. Они описываются в разделе описаний основной программы и по своей структуре аналогичны программе на языке Паскаль. После того, как эти процедуры и функции описаны пользователем, можно обращаться к ним по имени (вызывать их) в основной программе также как и к стандартным процедурам и функциям.

Рассмотрим более подробно работу с процедурами и функциями, определяемыми пользователем, в языке Паскаль.

Процедуры

Описание процедуры имеет вид:

Procedure *<имя процедуры> [(< список формальных параметров >)];*

< раздел описаний процедуры >

Begin

< тело процедуры >

End;

Обращение к процедуре (вызов ее в основной программе) имеет вид:

< имя процедуры > [(список фактических параметров)];

Формальные параметры - это имена переменных, которые используются при описании процедуры в ее заголовке и также в теле процедуры.

Фактические параметры - это значения (константы, выражения или имена переменных), которые подставляются на место формальных параметров при обращении к процедуре. Количество формальных и фактических параметров, их последовательность и их типы должны быть согласованы.

Различают ***параметры-значения и параметры-переменные.*** Перед параметрами-переменными в описании процедуры ставится служебное слово **Var.**

Параметры-значения используются для передачи значений переменных из основной программы в процедуру, то есть служат как бы ***аргументами процедуры (входными параметрами).***

Параметры-переменные возвращают ***результат*** работы процедуры в основную программу, то есть являются как бы ***выходными переменными для процедуры.***

При изменении значения формального параметра-переменной во время выполнения процедуры одновременно происходит изменение значения фактического параметра. А при изменении формального параметра-значения изменение соответствующего ему фактического параметра не происходит.

Говорят, что ***обмен значениями*** между фактическими и формальными ***параметрами-переменными*** происходит ***по значению***, а ***параметрами-значениями по ссылке***. В принципе, параметры-переменные можно использовать и в качестве входных переменных процедур, особенно если надо сэкономить память.

При описании процедуры, в списке формальных параметров, имена переменных одного типа указываются через запятую, после последнего имени ставится двоеточие и указывается тип параметра. Параметры разных типов разделяются точкой с запятой, например:

Procedure pr1 (x,y:integer; z:real; var t:real);

Пример 1.Составить программу, выводящую на экран четыре горизонтальных линии из символов ” * ”:

1 линия-7 символов, 2 линия-19 символов, 3 и 4 линии- любое количество символов (<80, соответствующие числа обозначим k3 и k4 и будем вводить их с клавиатуры).

В результате работы программы на экране должны получиться следующие линии:

1. * * * * * * *
2. * * * * * * * * * * * * * * * * * * *
3. * * * * * . . . . . * * *. . . . .* * * - всего k3
4. * * * . . . . * * * - всего k4

В этой задаче четыре раза повторяются одинаковые действия, а именно, вывод строки из “*”, но количество “*”в каждом случае различно. Вывод N звездочек на экран будет представлять собой цикл с параметром, повторяющимся N раз, и в теле цикла на экран будет выводиться одна звездочка при помощи операторaWrite (‘*‘).

Для того, чтобы в программе четыре раза не повторять этот цикл оформим его в виде подпрограммы, для которой входным значением

(параметром - значением) будет одно число N. Результатом работы этой процедуры будет вывод на экран строки звездочек.

В основной программе вызовем эту процедуру 4 раза, задавая соответствующие ***фактические значения n***

```
Program pr_line;
    var  k3, k4 : integer ;
    {Описание процедуры}
    Procedure zvezd ( n : integer );
        Var  i : integer ;
    Begin
      For i : = 1 to n do
         Write ( ' * ' );
      Writeln; { для перевода строки}
    End ; {конец процедуры}
  Begin { Основная программа }
    Write (' Введите k3 и k4  ' );
    Readln( k3, k4 );
    Zvezd( 7 ); {первый вызов процедуры}
    Zvezd ( 19 ); {второй вызов процедуры }
    Zvezd ( k3 ); {третий вызов процедуры}
    Zvezd ( k4 ); {четвертый вызов процедуры}
  End.
```

Пример 2.Составить программу поиска максимального из четырех чисел, используя процедуру максимума из двух чисел .

В отличие от процедуры первого примера, в которой не было параметра-переменной, так как никакие значения в основную программу не передавались, в процедуре поиска максимума из двух чисел будет два параметра-значения (аргументы процедуры), из которых ищется максимальное, и один результат (параметр-переменная) - значение максимума из двух чисел.

Формальным параметрам-значениям дадим имена *x*,*y*, а формальному параметру-переменной - *z*. То есть заголовок описания процедуры поиска max из двух чисел будет иметь вид:

procedure max_2 (x,y:real; var z:real);

Для поиска максимума из нескольких чисел будем последовательно находить максимум из двух и результат помещать в переменную *m;* на каждом следующем шаге сравнивать следующее число с *m* и хранить в *m*. То есть в основной программе будем три раза обращаться к процедуре *max_2:*

```
program max_4;
        var a, b, c, d, m : real;
        procedure max_2 (x, y : real; var z : real);
        begin
              if x >y then z:=x else z:=y;
        end;
begin
      write ( 'Введите 4 числа     ');
      readln (a,b,c,d);
      max_2 (a, b, m);  {первый вызов, в m - max из 2-х}
      max_2 (m, c, m); {второй  вызов, в m - max из 3-х}
      max_2 (m, d, m); {третий вызов, в m - max из 4-х}
      writeln ('max=',m)
end.
```

Самостоятельно составить программу вторым способом: находим max_2 из 1 и 2 числа, затем max_2 из 3 и 4 чисел, затем max_2 из получившихся двух максимумов

Функции

Как мы уже отмечали, в том случае, когда результатом работы процедуры является одно скалярное значение, эту процедуру можно оформить в виде функции.

Описание функции имеет вид:

Function *<имя> (<список формальных параметров>): <тип результата>*

<раздел описаний>

begin

<тело функции>

end;

Обращение к функции (ее вызов) имеет вид:

<имя функции> (<список фактических параметров>)

Результат работы функции присваивается ее имени, поэтому в теле функции обязательно должен быть оператор присваивания вида:

<имя функции>:=<значение результата>.

После того, как функция описана, её можно использовать в качестве операнда по имени в правой части оператора присваивания при вычислении значений каких-то выражений. То есть использование функций, определенных пользователем, аналогично использованию встроенных

стандартных функций (sin(x), cos(x) и т.д.). Поэтому при решении задач из какой-либо конкретной области пользователь обычно составляет свою библиотеку функций, оформляет эти функции в виде отдельного файла и подключает этот файл к основной программе. Подключаемый файл может быть либо текстовым файлом, либо специально оттранслированным файлом-модулем.

Пример: Составить программу вычисления значения выражения

$$C = \frac{m!n!}{(m-n)!} \qquad \text{где } (m > n)$$

Здесь трижды надо вычислить факториал, поэтому оформим вычисление факториала произвольного числа в виде функции. Функция для вычисления факториала будет иметь один формальный параметр, обозначим его буквой **k**.

Для вычисления k! необходимо организовать цикл с параметром i, который будет изменяться от 1 до k, а в теле цикла накапливать произведение **P:=P*i.** Начальное присваивание: P=1. Переменные P и i будут локальными переменными и описываются в разделе описаний функции:

Текст программы:

```
Program  primer_func;
        Var  m,n:integer;  C:real; {глобальные переменные}
         Function fact (k:integer):longint;
               Var i:integer; P:longint; {локальные переменные}
          Begin {тело функции}
               P:=1;
               For i:=1 to k do
                     P:=P*i;
               Fact:=P
          End;
Begin {основная программа}
        Write ('m,n=');
        Readln (m,n);
        C:=fact (m) * fact(n) /fact(m-n);
        Writeln (c)
End.
```

Вложенность процедур и функций. Побочные эффекты.

Как мы уже отмечали, допускается любой уровень вложенности процедур и функций, но иногда при использовании вложенных процедур и функций возникают побочные эффекты, выражающиеся в том, что вносятся нежелательные изменения (искажения) данных в программе. В основном это искажение подпрограммой значений глобальных переменных. Для избежания побочных эффектов необходимо:

1. Разделить все параметры на параметры-значения (для исходных данных) и параметры-переменные (для результатов).
2. Не использовать по возможности в подпрограммах глобальных переменных.
3. В функциях не использовать параметров-переменных.
4. Подбирать имена локальных и глобальных переменных так, чтобы они не совпадали.
5. Совпадение имен формальных и фактических параметров допускается, но для большей наглядности также лучше сделать их различными.

Пример: Составить программу вычисления значения выражения

$$V = \frac{\max(x, y, z) + \max(x + y, x - y, x + z)}{\max(xz, xy, yz)},$$

используя функцию нахождения максимума из 3 чисел (max3) и функцию нахождения максимального из 2 чисел (max2), причем max2 должна быть вложенной по отношению к max3.

```
Program vlog_func;
  Var x,y,z,v : real;
    Function max3(a,b,c:real) : real;
      Var d : real;
      Function max2(e,f:real):real;
      Begin
        If e>f then max2:=e else   Max2:=f
      End;
     Begin
        d:=max2(a,b);
        d:=max2(d,c);
        Max3:=d
      End;
    Begin
      Write (‘ Введите 3 числа’);
      Readln (x,y,z);
       V:=(max3(x, y ,z)+max3(x+y, x-y, x+z))/
```

```
            (max3(x*y, x*z, y*z));
        writeln ('v=',v:6:4);
    End.
```

Графические процедуры и функции модуля GraphABC

Во всех языках программирования есть стандартный набор процедур и функций, позволяющий строить графические изображения. Как известно, изображение на экране монитора формируется из точек (пикселей) определенного цвета. Положение каждой точки задается ее координатами.

В качестве *экранных координат* используют порядковые номера пикселов по горизонтали и вертикали. Они могут принимать только целочисленные значения. Началом отсчета является левый верхний угол экрана. Значения координаты ***x*** (абсциссы) отсчитывается слева направо, а значения ***y*** (ординаты) - сверху вниз.

Векторное изображение формируется из простых геометрических фигур (**графических примитивов**) - точек, отрезков, прямоугольников, окружностей, эллипсов.

В среде Pascal ABC библиотека стандартных графических процедур и функций хранится в модуле **GraphABC,** поэтому для работы в графическом режиме необходимо подключение этого модуля, и первой инструкцией программы должна быть инструкция **uses GraphABC**. При работе с графическим окном в среде PascalABC удобно все данные вводить и выводить в этом окне. Совмещать работу с текстом и графикой в одном окне можно, подключив модули **CRT** и **GraphABC** одновременно.

Графический экран (окно) PascalABC по умолчанию содержит 640 точек по горизонтали и 400 точек по вертикали. Размеры графического окна можно изменять с помощью процедуры ***SetWindowSize(w,h)***, где ***w*** и ***h*** – ширина и высота окна.

Основные инструменты рисования - **перо** и **кисть**. Параметрами пера (***Pen***) задаются свойства линий и контуров фигур (толщина в пикселях, цвет…), а параметрами кисти (***Brush***) - их закраска.

Цвет может задаваться константами стандартных цветов, либо значениями каждой из трех составляющих в модели RGB (красной, зеленой, синей) от 0 до 255.

Подробную информацию о процедурах и функциях модуля GraphABC можно получить с помощью встроенной справки. Для этого надо

в главном меню системы выбрать пункты Помощь→Содержание, а затем раскрыть ветвь Стандартные модули→Модуль GraphABC (рис. 7):

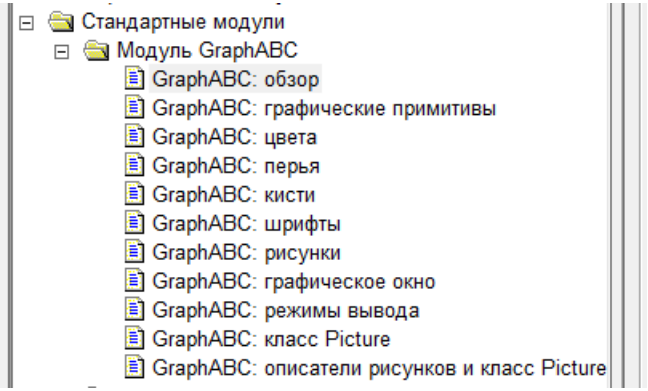


Рис. 4.4 Описание графических процедур и функций

Например, при выборе пункта GraphABC:обзор, появляется следующая информация:

Модуль GraphABC содержит константы, типы, процедуры, функции и классы для рисования в *графическом окне*. Они подразделяются на следующие группы:

- Графические примитивы
- Действия с цветом
- Действия с пером
- Действия с кистью
- Действия со шрифтом
- Действия с рисунками: описатели
- Действия с рисунками: класс Picture
- Действия с графическим окном
- Задание режимов вывода

Далее по соответствующей гиперссылке переходим к нужной нам информации.

Рассмотрим графические возможности системы программирования Pascal ABC на примерах.

Пример 1. Составить линейную программу, иллюстрирующую основные графические возможности системы программирования Pascal ABC. Программа выводит в графическое окно следующий рисунок (рис. 4.5):

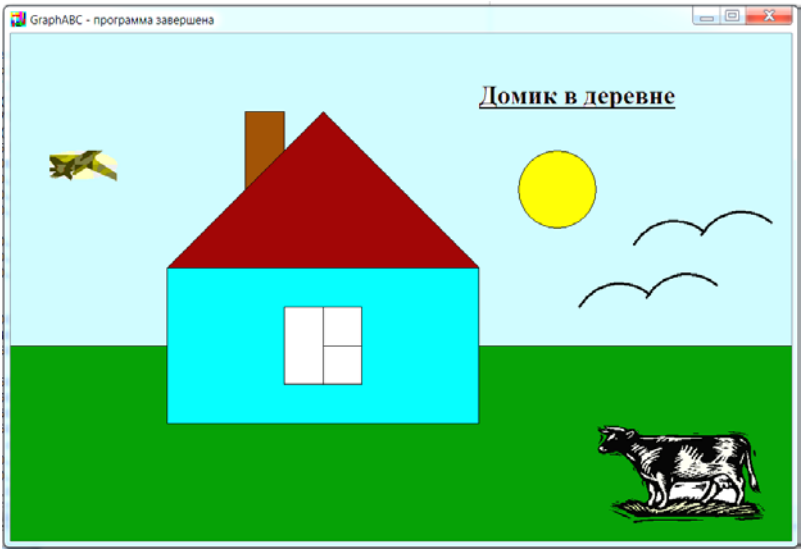


Рис. 4.5 Пример простейшей графической программы

Приведем текст программы:

```
program primer_grafiki;
        uses GraphABC;
         var pic: integer;
begin
         SetFontSize(20);
         SetFontName('Times New Roman');
         SetFontStyle(fsBoldUnderline);
         TextOut(600,60, 'Домик в деревне');
         pic:=LoadPicture('корова.bmp');
         DrawPicture(pic,750,500);
         pic:=LoadPicture('самолет.bmp');
         DrawPicture(pic,50,150);
         SetWindowSize(1000,650);
         Rectangle(200,300,600,500);
         Rectangle(350,350,450,450);
         line (200,300,400,100);
         lineTo(600,300);
         line (400,350,400,450);
         line (400,400,450,400);
         line (300,200,300,100);
```

```
lineTo(350,100);
lineTo(350,150);
line (600,400,1000,400);
line (0,400,200,400);
circle (700,200,50);
setpenwidth(3);
arc (850,300,60,50,150);
arc (936,288,60,50,150);
arc (780,380,60,50,150);
arc (866,368,60,50,150);
floodfill(100,50,clSkyBlue );
floodfill(400,200,clMaroon);
floodfill(250,400,clAqua);
floodfill (730,210,clYellow);
floodfill (330,150,clBrown);
floodfill (100,500,clGreen);
```

end.

Здесь изображения коровы и самолета загружаются из файлов 'корова.bmp' и 'самолет.bmp' соответственно.

Графические процедуры пользователя. Построение графических композиций. Мультипликация.

Используя стандартные процедуры и функции модуля GraphABC, можно составить собственные процедуры, позволяющие рисовать какой-то фрагмент рисунка, а затем на его основе строить графические композиции по заданному закону, а также программировать движущиеся изображения. Рассмотрим соответствующие примеры, в основе которых лежит процедура рисования ракеты, представленной на рисунке 4.6:

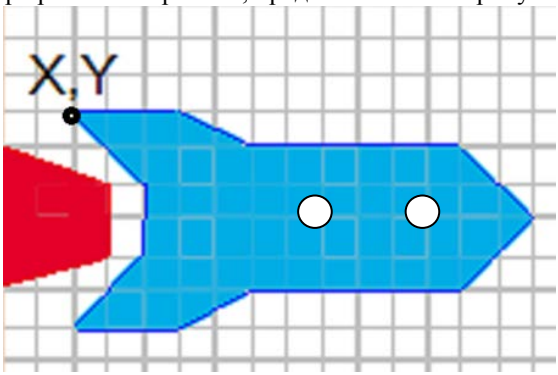


Рис. 4.6 Изображение ракеты

Для рисования ракеты в процедуре будем использовать метод узловой точки, то есть выбираем точку (на рисунке она помечена черным кружком и указаны координаты Х,У), относительно которой будем задавать все операторы рисования. Такой подход дает возможность «привязать» рисунок к узловой точке, и при изменении ее кординат фигуру можно строить в произвольной выбранной области экрана. Кроме координат узловой точки выберем переменные, определяющие размер фигуры, их можно выбрать несколько, но в данном случае выберем одну переменную, равную стороне клетки, обозначим ее буквой А.

Итак, наша процедура рисования ракеты будет иметь три параметра – x,y,a. Это будут параметры-значения (аргументы процедуры). Задачу можно сформулировать следующим образом:

Составить процедуру рисования ракеты: raketa(x,y,a:integer), изображенной на рисунке 9.

Аргументы процедуры: x,y – начальная точка, a – размер клетки

Приведем текст процедуры, обратив внимание, что для установки начальной точки рисования используется стандартная процедура **MoveTo(x,y: integer),** которая передвигает невидимое перо к точке с координатами (x,y). Построение остальных лини контура ракеты выполняется с помощью процедуры **LineTo(x,y: integer),** которая рисует отрезок от текущего положения пера до точки (x,y); координаты пера при этом также становятся равными (x,y).

```
procedure raketa(x,y,a:integer);
 begin
  setPenWidth(2);
  SetPenColor(clblue);
  MoveTo(x,y);
  Lineto(x+3*a,y);  Lineto(x+5*a,y+a);
  Lineto(x+11*a,y+a); Lineto(x+13*a,y+3*a);
  Lineto(x+11*a,y+5*a); Lineto(x+5*a,y+5*a);
  Lineto(x+3*a,y+6*a); Lineto(x,y+6*a);
  Lineto(x+2*a,y+4*a); Lineto(x+2*a,y+2*a);
  Lineto(x,y);
  circle(x+6*a,y+3*a,a div 2);
  circle(x+9*a,y+3*a,a div 2);
  FloodFill(x+2*a,y+a,clAqua);
  {sled}
  SetPenColor(clred);
  MoveTo(x-2*a,y+a);
  Lineto(x+a,y+2*a);  Lineto(x+a,y+4*a);
```

```
  Lineto(x-2*a,y+5*a); Lineto(x-2*a,y+a);
  FloodFill(x,y+3*a,clred);
end;
```

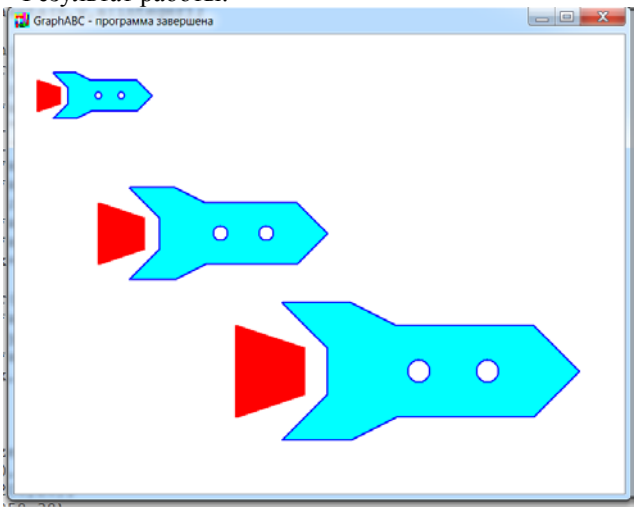
Рассмотрим теперь несколько программ, использующих данную процедуру.

Программа 1

Изобразить на экране три ракеты разных размеров в разных местах экрана.

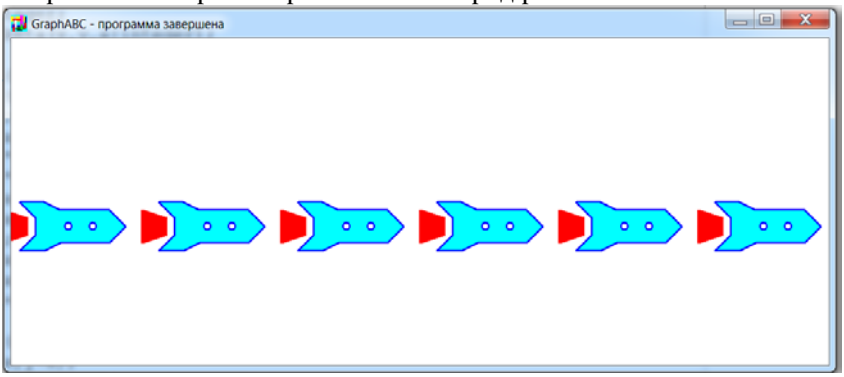
```
program raketa_proc_1;
 uses graphABC;
 procedure raketa(x,y,a:integer);
  begin
   setPenWidth(2);
  {текст процедуры}
  end; {конец процедуры}
begin {основная программа}
   SetWindowSize(800,600);
   raketa(50,50,10); {первый вызов процедуры-верхняя маленькая ракета}
  raketa(150,200,20); {второй вызов – средняя ракета}
  raketa(350,350,30); {третий вызов - нижняя большая ракета}
end.
```

Результат работы:



Программа 2

Изобразить на экране горизонтальный ряд ракет.



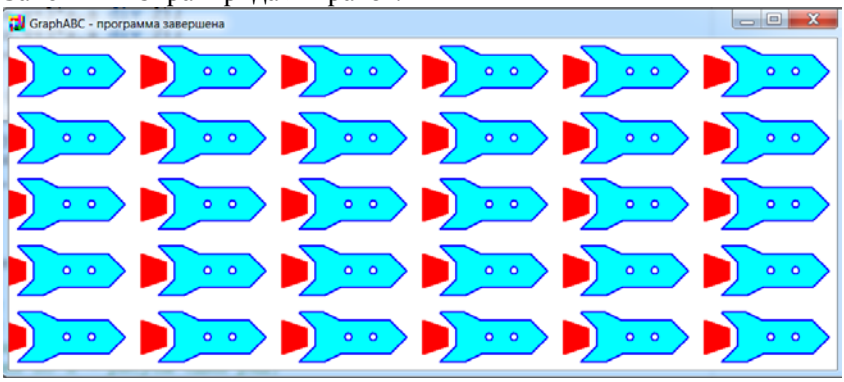
Анализ программы:

Организуем цикл, в теле которого будет рисоваться одна ракета с помощью вызова процедуры **raketa(xr,200,ar).** Здесь **ar –** константа, определяющая размер ракеты, **200** – координата y (можно взять другую константу), **xr –**координата x – переменная, которая будет изменяться в цикле от начального значения (например 10), до конечного – размеры экрана по горизонтали минус длина ракеты (она равна **13* ar).** Перед очередным повторением цикла изменяем **xr** по закону**: xr:=xr+17*ar** (вместо 17 можно взять другое число, большее 13 – размеров ракеты).

```
program raketa_cikl_2;
uses graphABC;
 var xr,ar:integer;
 procedure raketa(x,y,a:integer);
.....................................
Begin {основная программа}
  SetWindowSize(1000,400);
  xr:=10; ar:=10;
 repeat
    raketa(xr,200,ar);
    xr:=xr+17*ar
 until xr>1000-13*ar
end.
```

Программа 3

Заполнить экран рядами ракет:



Для решения задачи необходимо организовать вложенные циклы: внешний – по координате y, а внутренний – по координате x. Во внутреннем цикле будет рисоваться один ряд ракет при фиксированном значении y. Приведем текст программы с комментариями:

```
program raketa_cikl_3;
        uses graphABC;
        var xr,ar, yr : integer;
        procedure raketa(x,y,a:integer);
      ……….
begin {основная программа}
     SetWindowSize(1000,400);
       ar:=10;
       yr:=10; {нач присв для внешн цикла}
       repeat {цикл по y - рисуем ряды}
        xr:=10; {нач присв для внутр цикла}
        repeat {цикл по x - рисуем один ряд}
          raketa(xr,yr,ar);
          xr:=xr+17*ar
        until xr>1000-13*ar; {ряд закончен}
       yr:=yr+8*ar {переход к след ряду – увеличиваем  y}
      until yr>400-6*ar; {экран заполнен}
 end.
```

Программа 4

Изобразить на экране движущуюся ракету (по горизонтали)

Основной принцип создания движущихся изображений – принцип мультипликации:

1. рисуем объект в некоторой точке;
2. задерживаем изображение ненадолго;
3. стираем изображение;
4. изменяем местоположение – координаты узловой точки;
5. к пункту 1

То есть необходимо организовать цикл, в теле которого повторяются действия 1-4. Алгоритм решения задачи аналогичен программе 2, только добавляются действия задержки и стирания. Для того, чтобы стереть объект, его надо повторно прорисовать цветом фона или просто нарисовать прямоугольник цвета фона поверх изображения. Задержку организуем при помощи процедуры **Delay(ms: integer)**, которая осуществляет задержку выполнения программы на **ms** миллисекунд. Эта процедура содержится в модуле CRT, поэтому его надо подключить к программе наряду с модулем graphABC.

Приведем текст программы с комментариями:

```
program raketa_ dinam_4;
      uses graphABC, CRT;
      var xr,ar, yr : integer;
      procedure raketa(x,y,a:integer);
          ……….
begin {основная программа}
  SetWindowSize(1200,400);
   xr:=0; ar:=10; yr:=200;
  repeat
   raketa(xr,yr,ar);  {рисуем}
   delay(100);  {задержка}
   SetPenColor(clwhite); {устанавливаем цвет фона для
контура}
   setbrushcolor(clWhite); {устанавливаем цвет фона для
заливки}
   Rectangle(xr-2*ar,yr, xr+14*ar, yr+7*ar); {рисуем прямо-
угольник цветом фона - стираем}
   xr:=xr+ar  {изменяем положение}
  until xr>1200
end.
```

То, насколько реалистично будет выглядеть иллюзия движения, определяется величиной задержки (у нас сейчас 100 мс) и приращением **ar** (у нас сейчас 10). Введите текст программы и поэкспериментируйте с этими величинами!

Программа 5

Еще один способ создания движущихся изображений – использование готовых рисунков. Напомним, что для работы с файлами-рисунками используются стандартные функции и процедуры.

Функция **function LoadPicture(fname: string)**: integer; **n:=LoadPicture(fname)** загружает рисунок из файла с именем fname в оперативную память и возвращает описатель рисунка в целую переменную n; если файл не найден, то возникает ошибка времени выполнения. Загружать можно рисунки в формате .bmp, .jpg или .gif.

Процедура **DrawPicture(n,x,y,w,h: integer)** выводит рисунок с описателем n в позицию (x,y) графического окна, масштабируя его размеры к ширине w и высоте h.

Сформулируем **постановку задачи:**

Изобразить на экране движущуюся ракету: ракета хранится в файле-рисунке с именем rak_1.jpg (см. рис. 4.7), в файле rak_2.jpg –фон (прямоугольник цвета фона). Рисунки имеют размер 150*120 пикселей.

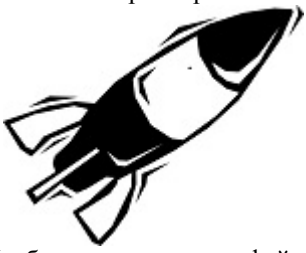


Рис.4.7 Изображение ракеты – файл rak_1.jpg

В соответствии с направлением ракеты движение организуем по диагонали экрана – из левого нижнего угла в правый верхний.

Приведем сначала текст программы, а затем дадим пояснения:

```
program raketa_dinam_ris_5;
  uses graphABC, crt;
  var a,b,k,i,hx,hy,x,y,n1,n2:integer;
begin
  a:=1000;
  b:=500;  k:=250;
  SetWindowSize(a,b);
```

```
  x:=-100;  y:=b; hx:= a div k; hy:= b div k;
  n1:=LoadPicture('rak_1.jpg');
  n2:=LoadPicture('rak_2.jpg');
 for  i:=1 to k do
   begin
    DrawPicture(n1,x,y,75,60);  {рисуем}
    delay(50);  {задержка}
    DrawPicture(n2,x,y,75,60); {стираем}
    x:=x+hx; y:=y-hy;  {изменяем положение}
   end;
end.
```

Для задания ширины и высоты окна мы использовали переменные a и b с той целью, чтобы можно было легко изменять размеры окна. Переменная k – количество повторений цикла показывает, сколько раз мы высвечиваем рисунок. Приращение координат мы вычисляем по формулам **hx:= a div k; hy:= b div k** для того, чтобы обеспечить их изменение пропорционально размерам экрана, в результате чего ракета движется именно в правый верхний угол экрана. При изменении координат x увеличивается, а y – уменьшается. В процедурах DrawPicture числа 75 и 60 взяты как половина размеров рисунка в пикселях для того, чтобы уменьшить рисунок в 2 раза, не искажая пропорции.

Лабораторная работа № 4

Тема: Организация программ с использованием процедур и функций.

Цель:

1) Получить навыки в составлении графических программ с использованием стандартных процедур и функций модуля GraphABC.
2) Получить навыки в составлении процедур и функций пользователя для решения вычислительных задач.
3) Провести сравнительный анализ использования процедур и функций для решения конкретной задачи.
4) Изучить механизм передачи параметров по значению и по ссылке.

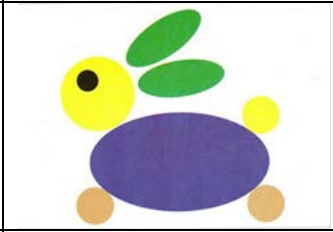
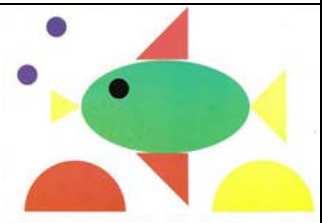
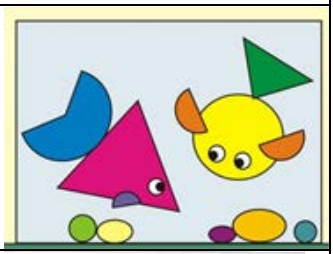
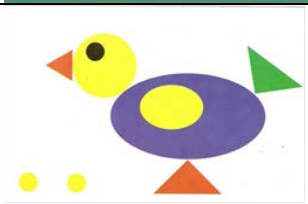
Содержание отчета:

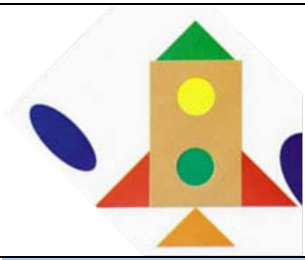
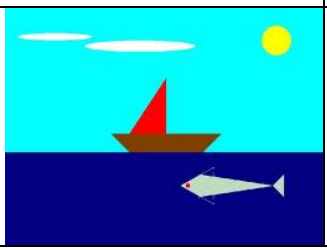
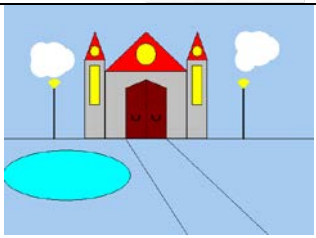
1) Постановка задачи.
2) Описание используемых процедур и функций, смысла и назначения их параметров.

3) Текст основной программы, анализ обращения к процедурам (функциям) и механизма передачи параметров.
4) Протокол работы программ, выводы.

Задание 1 (линейные графические программы)

Используя стандартные процедуры и функции модуля GraphABC, составить программу изображения рисунка (по номеру варианта):

№	Рисунок	№	Рисунок
1		2	
3		4	
5		6	
7		8	

9		10	
11		12	

Задание 2 (процедуры пользователя)

Составить программу для решения задачи своего варианта двумя способами: а) с использованием процедур

б) с использованием функций

1) Пятиугольник задан координатами своих вершин. Найти его площадь.
2) Два треугольника заданы координатами своих вершин. Определить, площадь какого из них больше.
3) Даны четыре натуральных числа. Найти их НОД.
4) Даны четыре натуральных числа. Найти их НОК.
5) Даны четыре натуральных числа. Вывести на экран то из них, сумма цифр которого – наибольшая.
6) Даны четыре натуральных числа. Просуммировать те из них, которые не содержат цифру 5.
7) Четыре точки на плоскости заданы своими координатами. Определить, какие две из них расположены наиболее близко друг к другу.
8) Найти значение выражения $y=a_4x^4+a_5x^5+a_2x^{-2}+a_3x^{-3}$, где a_i и x вводятся с клавиатуры. Вычисление степени оформить в виде процедуры (функции) через умножение.
9) Даны четыре натуральных числа. Определить количество цифр в каждом из них.
10) Два треугольника заданы координатами своих вершин. Определить радиусы окружностей, вписанных в треугольники.
11) Два треугольника заданы координатами своих вершин. Определить радиусы окружностей, описанных возле треугольников.

12) Даны четыре натуральных числа. Определить разность между наибольшим и наименьшим из них.

Задание 3 (графические процедуры пользователя)

Используя метод узловой точки, составить процедуру рисования фигуры (по вариантам).

№ варианта	**Фигура**
1 и 7	кораблик
2 и 8	легковой автомобиль
3 и 9	грузовой автомобиль
4 и 10	паровоз
5 и 11	самолет
6 и 12	танк

С помощью этой процедуры составить две программы:

1) нарисовать на экране три таких фигуры разных размеров в разных местах экрана.
2) Реализовать движение фигуры по экрану по горизонтали слева направо.

Раздел 5. СТРУКТУРИРОВАННЫЕ (СЛОЖНЫЕ) ТИПЫ ДАННЫХ. МАССИВЫ

Сложные типы данных языка Паскаль

Рассмотренные нами ранее типы данных называются простыми или скалярными. Основными свойствами данных этих типов являются неделимость и упорядоченность. Сложные типы данных образуются из простых объединением их некоторую группу, которой присваивается одно имя и при этом определяется, каким образом можно обратиться к отдельному элементу группы. В свою очередь сложные типы данных также могут объединяться в группы.

В языке Паскаль к основным сложным типам данных относятся:

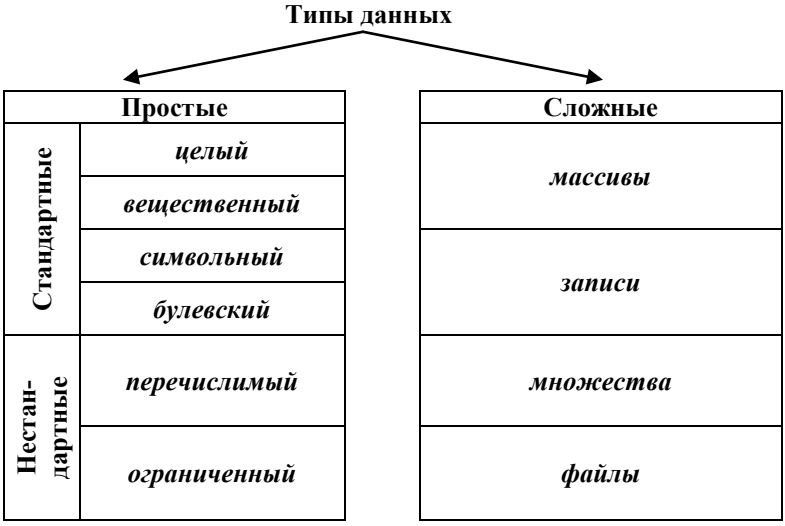
1.Регулярный тип – массив

2.Комбинированный тип – запись

3.Множественный тип – множество

4.Файловый тип – файлы

Таким образом, структуру типов данных языка Паскаль схематически можно представить следующим образом:.



Массивы

В программировании под ***массивом*** понимается ***упорядоченная*** последовательность ***однотипных величин***, обозначаемых ***одним именем.*** Упорядочение производится по номеру элементов в массиве - индексу. Индекс записывается в квадратных скобках сразу после имени массива. Само имя образуется по тем же правилам языка Паскаль, что и для простых типов. В зависимости от количества индексов, однозначно определяющих элементы массива, различают одномерные (1 индекс), двумерные (2 индекса), трехмерные и т.д. массивы.

Например.

A – имя массива

A[5] - пятый элемент одномерного массива

A[5,8] – элемент двумерного массива

Одномерные и двумерные массивы имеют наглядное практическое представление. Примером одномерного массива может служить линейная таблица или вектор:

Таблица-одномернвый массив с именем A:

1	2	3	4	5	6	7	8

- индексы элементов

Пример двумерного массива - это прямоугольная таблица или матрица:

Таблица - двумерный массив с именем B:

8	5	2	-5	0
3	11	-5	-7	4
12	45	9	10	23

При этом 1-й индекс показывает номер строки, а 2-й индекс - столбца. Например, значение элемента B[2,3] равно -5, а значение Элемента B[3,2] равно 45.

Таким образом, массив имеет следующие основные характеристики:

1.Размер – число элементов в массиве;

2.Имя массива.

3.Индексы элементов.

4.Значение элемента.

Характеристики 3 и 4 связаны с конкретным элементом массива (определяют его). Имя массива и размерность характеризуют массив в целом. Размерность массива задается при его описании, и при этом в памяти компьютера резервируется непрерывная область, в которую в дальнейшем заносятся значения элементов. В языке Паскаль при описании массива указывается максимально возможное значение элементов,

причем это число первоначально определяется в разделе констант, или непосредственно указывается в диапазоне индексов. При этом память компьютера расходуется неэффективно в тех случаях, когда при непосредственной обработке массива фактически используются массивы, для многих из которых размерность значительно меньше указанного в описании числа. Для устранения указанного недостатка и более эффективного использования памяти нужно использовать динамическую память.

Массив можно описать, либо введя в разделе *type* соответствующий тип, либо непосредственно в разделе переменных *var*:

Общий формат описания:

Type

<имя типа>=array[<диапазон индексов >] of
< тип элементов массива >;

или

var

< имя массива >: [<диапазон индексов >] of
< тип элементов массива >;

Тип индексов может быть любым скалярным, кроме вещественного. Обычно тип индексов - это ограниченный тип, например.

```
type
    mas1=array[1. . . 30] of integer;
    mas2=array[1. . 10,1. . 15] of real;
var
     a:mas1;
     b:mas2;
```

или

```
var
     a:array [1 . .30] of integer;
     b:array [1. .10,1 . .15] of real;
```

В любых программах, связанных с обработкой массивов, всегда присутствует фрагмент ввода элементов массива в память компьютера и часто фрагмент вывода массива на экран.

Рассмотрим их более подробно, оформив в виде процедур.

а) ввод элементов массива

Ввод элементов массива можно осуществлять двумя способами: ввод с клавиатуры или с помощью функции случайных чисел. При вводе с клавиатуры основным повторяющимся оператором будет оператор

read (a [i]) – для одномерного массива

read (b[i,j])- для двумерного массива,

где *a* , *b* - имена массивов, i,j – индексы элементов.

При этом индексы i,j должны пробегать все свои значения, поэтому для ввода необходимо организовать цикл с параметром, в качестве которого будет выступать индекс(ы) переменных. Для двумерного массива это будут вложенные циклы, причем обычно внешний – по индексу строки i, а внутренний - по индексу столбца j.Так как мы хотим оформить эти фрагменты в виде процедуры, то входным параметром для нее будет число элементов массива, а параметром-переменной (результатом) - сам полученный массив. Описания массивов будут заданы в основной программе (Считаем, что описаны как выше массивы *a* , *b*).

Процедура ввода:

```
procedure vvod_odn_mas (n:byte; var a:mas1);
      var i: byte;
begin
      writeln ('введите элементы массива через пробел ')
      for i:=1 to n do read (a[i]);
      readln;
end;
```

В некоторых задачах можно не набирать на клавиатуре значения элементов массива (особенно если их много, а конкретные величины могут быть произвольными числами из заданного диапазона). Для этого в любом языке программирования есть возможность получать случайные числа из заданного интервала. В Турбо Паскале случайное число получается с помощью функции ***random (m)***, которая выдает случайное число от 0 до m-1. Функция ***random*** без аргументов генерирует вещественное случайное число из интервала (0,1). Для получения случайного целого числа ***из отрезка [a,b]*** нужно использовать следующее выражение:

random (b-a+1)+a

Например, если нужно получить число из отрезка ***[-30,40],*** то используем выражение ***random (71)-30***. или *trunc ((b-a+1)*random + a).*

Перед первым обращением к функции случайных чисел необходимо инициализировать датчик случайных чисел с помощью процедуры ***randomize***. Обычно эта процедура выполняется один раз в начале программы. Рассмотрим процедуру формирования элементов двумерного массива *b* с помощью случайных чисел (считаем, что тип массива описан в программе, как выше):

```
procedure vvod_dvum_sluch (n,m:byte; var b:mas2)
   var i,j:byte;
begin
   for i:=1 to n do
      for j:=1 to m do
```

```
    b [i,j]:=random (101);
end;
```

Массив, сформированный с помощью функции случайных чисел, обычно надо вывести на экран, чтобы видеть значения элементов, с которыми программа работает дальше. Вывод получившихся значений можно осуществлять в той же процедуре, либо оформить отдельно:

```
procedure print_dvum_mas (n,m:byte; var b:mas2);
      var i,j : byte;
begin
      for i:=1 to n  do
        begin
          for j:=1 to m do
              write (b[i,j]:5); {вывод одной строки – внутр цикл}
              writeln {перевод строки}
        end; {конец внешнего цикла}
  end; {конец процедуры}
```

Замечание 1: Так как в описаниях массивов должно быть указано количество элементов в массиве (в диапазоне индексов), то это значение можно указать в разделе констант, либо непосредственно задавать при описании. В обоих случаях необходимо брать число «с запасом», а конкретное количество элементов в массиве вводить с клавиатуры при каждом запуске программы. Очевидно, что последнее значение не должно превышать зарезервированное в константе.

Типовые алгоритмы обработки массивов

Большинство задач, связанных с обработкой массивов, используя метод пошаговой детализации, можно свести к небольшому числу типовых (стандартных) алгоритмов. К ним относятся:

1. Вычисление суммы (произведения) элементов массива.
2. Нахождение суммы (произведения) элементов массива удовлетворяющих некоторому условию или определение количества элементов, удовлетворяющих некоторому условию.
3. Нахождение максимального (минимального) элемента массива и(или) его индекса.
4. Формирование нового(ых) массива(ов) из элементов заданного(ых) массива(ов).
5. Изменение элементов исходного массива.
6. Сдвиг (перестановка) элементов в массиве (циклический сдвиг).
7. Поиск заданного элемента массива
8. Сортировка элементов массива.

Рассмотрим некоторые из них.

Пример 1. Дан одномерный массив целых чисел размерностью n, полученный с помощью случайных чисел. Требуется построить из него два массива, в первый из которых записаны элементы исходного массива, меньшие половины максимального значения, а во второй – остальные.

Эта задача сводится к двум типовым: 3 и 4.

Третий тип - задача поиска максимума в массиве решается классическим способом: сначала считается, что максимальный элемент – первый. Затем организуется цикл, в котором просматриваются все элементы, начиная со второго, и сравниваются с максимальным. Если текущий элемент больше, то заменяем значение максимального.

Для формирования новых массивов (4 тип) снова организуем цикл для просмотра всех элементов исходного массива, сравнивая каждый элемент с половиной максимума. В зависимости от истинности условия записываем элемент в новый массив: для этого сначала увеличиваем значение индекса нового массива и присваиваем соответствующему элементу нового массива значение элемента исходного массива.

Так как в данной задаче придется трижды выводить на экран элементы одномерных массивов, то оформим фрагмент вывода в виде процедуры. Ввод же исходного массива (случайным образом) организуем непосредственно в тексте основной программы.

```
program mas_typ3_4;
  const n1=40;
  type mas1=array[1..n1] of integer;
  var i,n,k1,k2,max : integer;
  a,b,c : mas1;
  procedure print_mas(k:integer; d:mas1);
    var j : integer;
  begin
    for j:=1 to k do write(d[j]:3);
    writeln
  end;
Begin
  writeln('введите число элементов массива ');
  readln(n);
  {формируем массив из чисел от -20 до 30}
  for i:=1 to n do
     a[i]:=random(51)-20;
  writeln(' исходный массив');
  {вызываем процедуру вывода массива на экран}
```

```
print_mas(n,a);
{начинаем поиск максимума}
max:=a[1];
for i:=1 to n do
  if a[i]>max then max:=a[i];
{обнуляем счетчики элементов новых массивов}
k1:=0; k2:=0; {}
{начинаем цикл для их формирования}
for i:=1 to n do
  if a[i]>max/2 then {записываем элемент в массив b }
        begin
          k1:=k1+1;{}
          b[k1]:=a[i]
        end
            else {записываем элемент в массив c }
        begin
          k2:=k2+1;{}
          c[k2]:=a[i]
        end;
  {выводим новые массивы на экран}
  writeln ('первый массив');
  print_mas(k1,b);
  writeln ('второй массив');
  print_mas(k2,c);
end.
```

Пример 2. Дан двумерный массив, содержащий n строк и m столбцов. Определить, есть ли в нем столбец, состоящий только из четных элементов. Если таких столбцов несколько, то вывести номера каждого из них, если такого столбца нет, то сообщить об этом.

Рабочий блок программы будет реализован с помощью вложенных циклов, причем внешний цикл надо организовать по индексу столбца, а внутренний – по индексу строки. Во внутреннем цикле будем подсчитывать количество четных элементов в текущем столбце (его индекс изменяется во внешнем цикле), а по окончании – проверять, равно ли это количество количеству всех элементов столбца. Если да - то выводим на экран номер столбца. Для сигнализации о том, что такого столбца нет, введем переменную булевского типа –флажок, которому первоначально присвоим значение *false* (искомый столбец не найден). Если же такой столбец находится, то изменяем значение флажка на *true:*

```
program  mas2_poisk;
    var   i,j,n,m,k:integer;
          f:boolean;
          a:array[1..15,1..15] of integer;
begin
     writeln('введите число строк и столбцов не больше 15 ');
     readln(n,m);
        randomize;
        {формируем массив из чисел от 1 до 10}
        for i:=1 to n do
              for j:=1 to m do
                    a[i,j]:=random(10)+1;
       {выводим массив на экран }
        for i:=1 to n do
                 begin
                    for j:=1 to m do
                       write(a[i,j]:3);
                       writeln;
                  end;
         f:=false; {такого столбца пока нет}
         {начинаем цикл по столбцу}
         for j:=1 to m do
              begin
                 k:=0; {счетчик четных элементов}
                 for i:=1 to n do
                       if a[i,j] mod 2=0 then k:=k+1;
                {конец внутреннего цикла}
                 if k=n then {столбец найден}
                       begin f:=true;
                             writeln('столбец номер ',j);
                       end;
               end;   {конец внешнего цикла}
         if not(f) then writeln('столбца нет')
end.
```

Задачи сортировки элементов массива

Сортировкой называется упорядочение элементов массива по некоторому признаку. В основном рассматриваются следующие виды сортировок:

а) по возрастанию
a[1] < a[2] < . . .< a[n]
б) по неубыванию
a[1]≤ a[2]≤…≤a[n]
в) по убыванию
a[1] >a[2] >…>a[n]
г) по невозрастанию
a[1]≤ a[2]≤…≤a[n]

Существует множество различных методов сортировок, которые отличаются степенью эффективности. Под эффективностью понимают количество сравнений и количество обменов между элементами массива, которые необходимо выполнить для того, чтобы упорядочить массив. Рассмотрим простейшие методы сортировки на примере сортировки по возрастанию.

Метод прямого выбора.

Основные этапы алгоритма:

1. Ищется минимальный элемент в массиве и переставляется с первым элементом массива.

2.В оставшейся части (начиная со второго) также ищется минимальный элемент, и переставляется со вторым.

3.Ищется минимальный элемент, начиная с третьего, и переставляется с третьим и т.д.

Всего этих шагов будет (n-1).

На i-м шаге ищется минимальный элемент для a[i]…a[n] и этот минимальный элемент переставляется с a[i].

Пример. 3 2 9 1 8 (исходный массив).

1 шаг 3 2 9 1 8
2 шаг 1 2 9 3 8
3 шаг 1 2 9 3 8
4 шаг 1 2 3 9 8
Результат 1 2 3 8 9

Фрагмент программы для сортировки. (рабочая часть)

Открываем внешний цикл по i – число шагов.

```
For i:=1 to n-1 do begin
{Тело этого цикла – поиск минимального элемента среди a[i] … a[n] и его индекса}
min:=a[i];  k:=[i]
   for j:=i + 1   to n do
      if a[j] < min then begin
                                 min:=a[j];  k:=j
                              end;
```

```
{Перестановка минимального a[k] и a[i]}
{Возьмем промежуточный элемент x-пустой «стакан» (задача обмена)}.
      x:=a[i];
      a[i]:=a[k];
      a[k]:=x;
end; {конец  внешнего цикла}.
```

Метод прямого обмена (“пузырьковая “сортировка).

Идея метода состоит в следующем: попарно сравниваются два элемента, начиная с первого, и если a[i]>a[i+1] то они меняются местами. В результате одного прохода максимальный элемент “всплывает” к концу.

Рассмотрим пример:

3 2 9 1 8 n=5

1 проход

i=1 3 >2 9 1 8
обмен

i=2 2 3 <9 1 8
обмена нет

i=3 2 3 9>1 8
обмен

2 проход

i=1 2<3 1 8 | 9
нет обмена

i=2 2 3>1 8 | 9
обмен

i=3 2 3 1<8 | 9

3 проход

i=1 2>1 3 | 8 9

i=2 1 2<3 | 8 9
обмена нет

4 проход

i=1 1<2 | 3 8 9
обмена нет

Результат 1 2 3 8 9

Рассмотрим фрагмент программы:

Обозначим k-количество проходов, k изменяется от 1 до n-1;

i –номер сравниваемых пар, I изменяется от 1 до n-k;

внешний цикл будет по параметру k;

внутренний цикл - по i;

в теле внутреннего цикла сравниваем a[i] с a[i+1] и делаем, если надо, обмен с помощью промежуточной переменной x:

```
for k:=1 to n-1 do
{открыли цикл по количеству проходов}
   for i:=1 to n-k do
       if a[i] >a[i+1] then
begin
 {перестановка a[i]и a[i+1]}
           x:=a[i];
           a[i]:=a[i+1];
           a[i+1]:=x;
  end;
```

Рассмотренные два метода относятся к числу основных, но не очень эффективных. Поэтому существует множество улучшенных методов: сортировки: метод Хоара, сортировка методом слияний и т. д.

Алгоритмы поиска элементов в массиве

Большинство задач поиска сводится к поиску в массиве элемента с заданным значением. Простейший метод поиска, называемый линейным, состоит в том, что последовательно просматриваются все элементы массива и запоминается номер искомого элемента, если такой есть. Первоначально, перед циклом, переменной, в которой будет храниться номер искомого элемента, присваивается некоторое значение, которое не может принимать индекс и это значение является признаком того, что такого элемента нет. Соответственно фрагмент программы имеет вид:

найти в одномерном массиве a элемент, равный x (x –заданное значение).

```
{k – индекс искомого элемента;}
 k := -1 {такого элемента нет}
for i:=1 to n do
     if a[i]=x    then k:=i;
if k = -1 then writeln (‘ такого элемента нет’)
    else writeln (такой элемент есть, его индекс ‘,i);
```

Недостатками такого поиска является следующие: если элемент x не один (то есть встречается несколько раз), то будет найден индекс последнего и массив всегда просматривается до конца, то есть совершается много лишних действий.

Более совершенный способ поиска заданного значения состоит в использовании цикла “ пока”, и в качестве условия продолжения цикла будет выступать условие (i <=n) and(a[i]< >x), причем порядок этих простых условий важен, то есть их нельзя переставлять местами, иначе возможен выход за границы массива. Второе условие a[i]< > x задает (определяет) выход из цикла сразу, как только будет найден элемент, равный x. Первое условие i <=n определяет выход из цикла в том случае, когда искомого элемента в массиве нет. В этом случае после окончания цикла i=n+1:

Фрагмент программы:

```
i:=1;
while (i <=n) and (a[i] < > x do i:=i+1;
if i=n+1 then writeln (‘ такого элемента нет’)
    else writeln (такой элемент есть, его индекс ‘,i);
```

Рассмотренные две программы называют линейным поиском, можно несколько улучшить второй метод, вводя так называемый барьерный элемент a[n+1]:=x, равный искомому, тогда условие в цикле упростится.

Фрагмент программы имеет вид:

```
a[n+1]:=x;
i:=1;
while (a[i] < > x) do inc (i);
if i=n+1   then …
```

Можно предложить еще один способ реализации линейного поиска – введение специальной сигнальной переменной - флажка, который может находиться в двух состояниях (принимать два значения):

1 - такой элемент есть

2 - такого элемента нет

Обычно в качестве такой переменной выбирается переменная булевского типа:

f-true –такой элемент есть

false – такого элемента нет.

Фрагмент программы:

```
i:=1; f:=false;{то есть считаем , что элемента нет}
while (i <=n) and (not(f)) do
          begin
              if a[i]=x then f:=true;
                  i:=i+1
          end;
if f then writeln (‘такой элемент есть, его индекс,i-1’);
     else writeln (‘такого элемента нет’);
```

Поиск элементов в массиве можно сделать более эффективным, если элементы массива упорядочены. В этом случае используется бинарный поиск, который также называют поиском методом деления пополам. Идея метода состоит в следующем: находится средний элемент массива и определяется, равен ли он искомому значению x. Если да, то поиск заканчивается, если нет, то отбрасывается та часть массива, в которой в силу упорядоченности массива известно, что элемента нет.

Например, если массив упорядочен по возрастанию

a[1] < a[2] < . . .<a[n] и мы нашли средний элемент a[m], то индекс этого элемента m:=(1 +n) div 2. Если x=a[m] –то конец программы, если не равен, то сравнить x <a[m], и если условие истинно, то отбросить правую часть, если ложно, то отбрасывается левая часть. Процесс продолжается до тех пор, пока, либо не выполнится условие x=a[m], либо та часть массива, которую мы делим пополам, не сведется к одному элементу.

Соответствующий фрагмент программы:

```
    begin
       lg :=1;{ левая граница рассматриваемой части массива
                на первом шаге весь массив}
        rg:=n; { правая граница }
        f:=false;{признак того, что элемент = x не найден }
{открываем цикл; условия продолжения цикла:
                                              (lg <=rg) and (not (f))}
         while (lg <=rg) and (not f) do
                begin
                   m:=(lg+rg) div 2;
      {проверяем условие}
              if a[m]=x then  f:=true
                         else
                  if x < a[m] then rg:=m-1
       {отбрасываем правую часть}
                                 else lg:=m+1
        { отбрасываем левую часть}
                end
          end.
```

В отличии от линейного поиска бинарный поиск дает существенную экономию времени работы программы за счет значительного сокращения операций сравнения.

Контрольные вопросы и задания

Вопросы:

1) Дайте определение массива. Какие ключевые слова можно в нем выделить?
2) Какие основные характеристики имеют элементы массива? ?
3) Что такое индекс? Каким требованиям он должен удовлетворять?
4) Каков формат описания массива, что в нем указывается?
5) Общие и отличительные черты одномерных, двумерных и n-мерных массивов.
6) Каким образом в Паскале задается обращение к элементу массива?
7) Какие типовые задачи обработки массивов можно выделить?
8) Какие существуют способы ввода элементов массива в память?
9) Приведите примеры формулировок задач каждого типа.

Задания:

Одномерные массивы:

1) Дан массив a[1]..,a[20] . Вычислить среднее арифметическое значение элементов и отклонение от среднего для каждого элемента.
2) Дан массив чисел a1,...,an. Выяснить, имеются ли в данном массиве 2 идущих подряд положительных элемента. Подсчитать количество таких пар.
3) Дан одномерный массив из n элементов. Требуется найти максимальный элемент и отклонение от максимального для каждого из элементов.
4) Дан одномерный массив из n элементов . Вычислить сумму положительных и произведение четных членов данного массива.
5) Если в данном массиве действительных чисел есть хотя бы один элемент, меньший чем -2,то все отрицательные элементы заменить их квадратами.
6) Сформировать в программе массив из целых чисел от 2 до N. Подсчитать сумму квадратов четных и сумму квадратов нечетных элементов.
7) Дан одномерный массив целых чисел из n элементов. Выяснить имеется ли в массиве хотя бы одно нечетное отрицательное число и определить его местонахождение первого такого числа в массиве.
8) Дан одномерный массив целых чисел. Найти количество и
9) сумму тех членов массива, которые делятся на 5 и не делятся на 7.

10) Дан одномерный массив целых чисел. Все члены массива, предшествующие наименьшему, умножить на это значение.

11) Дан одномерный массив целых чисел. Найти максимальный элемент массива и поменять его местами с первым элементом.

12) Дан массив целых чисел a1,...,an. Найти минимальный и максимальный элементы массива и вычислить сумму элементов, расположенных между ними (считать, что они не повторяются).

13) Дан одномерный массив. Определить максимальный элемент массива и элемент, являющийся максимальным без учета этого элемента.

14) Дан одномерный массив из четного числа элементов. Поменять местами его половины следующим способом: первый элемент поменять с последним, второй – с предпоследним и т.д.

15) Дан одномерный массив из 15 различных элементов. Переставить в обратном порядке элементы, расположенные между максимальным и минимальным элементами, включая их.

16) Известно, что в массиве имеются элементы, равные 5. Определить номер последнего из них. (Условный оператор не использовать).

17) Найти число элементов одномерного массива, которые больше своих "соседей",т.е. предшествующего и последующего.

18) Дан одномерный массив. Переписать его элементы в другой массив такого же размера следующим образом: сначала должны идти все отрицательные элементы, а затем все остальные. Испльзовать только один проход по исходному массиву.

19) Заполнить одномерный массив 20-ю первыми натуральными числами, делящимися нацело на 13 или на 17 и находящимися в интервале, левая граница которого равна 300.

20) Дан одномерный массив из 15 элементов. Найти три наибольших элемента.

21) Даны два массива, состоящие из 10 элементов. Получить третий массив, состоящий из 20 элементов по правилу: сначала должны идти все отрицательные элементы 1-го и 2-го массивов, а затем все остальные элементы исходных массивов.

Лабораторная работа № 5

Тема: Одномерные массивы

Цель: 1) Получение навыков в организации ввода-вывода элементов массива; закрепление навыков в использовании процедур

2) Изучение типовых алгоритмов обработки массивов: работа с элементами, поиск максимального и минимального элемента, сдвиг и

перестановка элементов, вставка и удаление элементов, формирование нового(ых) массива(ов) из элементов данного(ых) массива(ов), упорядочение элементов.

Содержание отчета:

1. Постановка задачи.
2. Текст программы с комментариями алгоритмической структуры отдельных ее частей и описанием смысла и назначения используемых переменных.
3. Протокол работы программы, анализ результатов.

Задание 1. Даны два одномерных массива целых чисел (массив A, состоящий из n элементов, массив B – из m элементов), заполненных случайным образом числами из промежутка [L,P]. Сформировать из элементов этих массивов два новых массива (C,D) по правилу, описанному в Вашем варианте. (Ввод и вывод массивов оформить в процедурах).

1. n= 15, m= 20, L= 10, P=99;

Массив C состоит из тех элементов исходных массивов, в которых обе цифры четные, а массив D – обе цифры нечетные.

2. n= 10, m= 25, L= 100, P=500;

Массив C состоит из тех элементов исходных массивов, в которых средняя цифра четная, а массив D – средняя цифра нечетная.

3. n= 20, m= 15, L= -50, P=50;

Массив C состоит из элементов исходных массивов, меньших -5, а массив D – больших 10.

4. n= 25, m= 12, L= 10, P=2000;

Массив C состоит из тех элементов исходных массивов, в которых последняя цифра равна 6, а массив D – последняя цифра равна 9.

5. n= 17, m= 18, L= -30, P=60;

Массив C состоит из положительных элементов исходных массивов, а массив D – отрицательных.

6. n= 12, m= 28, L= -60, P=90;

Массив C состоит из четных элементов исходных массивов, а массив D – нечетных.

7. n= 20, m= 30, L= 10, P=50;

Массив C состоит из элементов исходных массивов, меньших 30, а массив D – больших 30.

8. n= 30, m= 10, L= 10, P=150;

Массив C состоит из элементов исходных массивов, кратных 5, а массив D – кратных 3.

9. n= 22, m= 16, L= -50, P=30;
Массив C состоит из элементов исходных массивов, меньших первого элемента массива A, а массив D – больших второго элемента массива B.

10. n= 10, m= 20 L= 1000, P=5000;
Массив C состоит из элементов исходных массивов, первая цифра которых меньше последней, а массив D – первая цифра больше последней.

11. n= 15, m= 30, L= -100, P=100;
Массив C состоит из отрицательных элементов исходных массивов с четными индексами, а массив D – положительных элементов с нечетными индексами.

12. n= 30, m= 10, L= -200, P=150;
Массив C состоит из четных отрицательных элементов исходных массивов , а массив D – нечетных положительных элементов.

Задание 2.

1. Задан массив чисел. Замените каждое число суммой предыдущих, включая заменяемое.
2. Даны действительные числа a1,a2,...,a16. Найдите минимальное из произведений a1a9,a2a10,...,a8a16.
3. Даны действительные a1,a2,...,a16 . Найдите максимальное из сумм a1+a16,a2+a15,...,a8+a9.
4. Даны целые a1,a2,..,an.Все члены последовательности, предшествующие первому по порядку наименьшему члену, умножить на этот наименьший член.
5. Даны действительные a1,a2,..,an . Требуется найти b, равное среднему арифметическому чисел a1,a2,..,an и наибольшее отклонение от среднего т.е. max(|a1-b|,|a2-b|,...,|an-b|).
6. Ввести массив a[1],a[2],...,a[n] целого типа. Произвести сдвиг элементов массива на 2 позиции влево так, что на место a[1] станет a[3], на место a[2] станет a[4],на место a[n-1] станет a[1], на место a[n] станет a[2].
7. Ввести массив a[1],a[2],..,a[n] целого типа и число k<n. Произвести сдвиг элементов массива на k позиций вправо так, что на место a[n] станет a[n-k], на место a[1] станет a[n-k+1], на место a[2] станет a[n-k+2] и т.д..
8. Найти три максимальных значения max1>=max2>=max3 среди элементов таблицы из n вещественных чисел.
9. Дан массив из n целых элементов. Исключить последний минимальный элемент, то есть сдвинуть все значения таблицы, начиная с последнего минимального, влево на 1 позицию. Значение последнего элемента таблицы не определено.

10. Ввести массив a[1],a[2],...,a[n] целого типа и заменить все его элементы, стоящие до максимального, нулями.

11. Ввести массив a[1],a[2],...,a[n] целого типа и заменить все его элементы, стоящие после минимального, нулями.

12. Дан массив из n целых элементов. Исключить первый максимальный элемент, то есть сдвинуть все значения таблицы, начиная с первого максимального, влево на 1 позицию. Значение последнего элемента таблицы не определено.

Задание 3. Дан одномерный массив целых чисел, содержащий n элементов (n вводится с клавиатуры, массив создается с помощью случайных чисел). Упорядочить массив:

Варианты 1,3,5,7,9,11 – по возрастанию

Варианты 2,4,6,8,10,12 – по убыванию.

Двумерные массивы:

1) Дана квадратная матрица A(N.N). Составить программу вычисления суммы элементов, расположенных ниже главной диагонали.

2) Дана квадратная матрица A(N,N). Составить программу замены отрицательных элементов, расположенных выше главной диагонали, на 0. Исходную и скорректированную матрицы напечатать.

3) Дан двумерный массив с одинаковым количеством строк и столбцов. Определить сумму элементов побочной диагонали массива.

4) Дан двумерный массив. Определить, сумму элементов 4-го столбца массива, меньших 100.

5) Дан двумерный массив. Определить количество элементов 4-й строки массива, меньших 5.

6) Дана вещественная матрица A(N,M). Составить программу нахождения первого максимального элемента матрицы и его индексов.

7) Дана вещественная матрица A(N.M). Составить программу замены всех отрицательных элементов матрицы на элемент, имеющий максимальное значение.

8) Дана целочисленная матрица. Составить программу подсчета среднего арифметического значения матрицы. Найти отклонение от среднего у элементов первой строки.

9) Составить программу нахождения числа строк двумерного массива, минимальный элемент которых меньше заданного числа P.

10) Составить программу нахождения числа строк двумерного массива, сумма элементов у которых отрицательна.

11) Составить программу нахождения максимального элемента в каждом столбце двумерного массива.

12) Дан двумерный массив целых чисел. Столбец, содержащий первый максимальный элемент, поменять местами со столбцом, содержащим последний минимальный элемент.

13) Дан двумерный массив целых чисел. Сформировать одномерный массив, каждый элемент которого равен первому четному элементу соответствующего столбца двумерного массива (если такого элемента в столбце нет, то равен нулю).

14) Дан двумерный массив целых чисел. Сформировать два одномерных массива. Каждый элемент первого равен количеству нечетных отрицательных элементов соответствующей строки двумерного массива. Кажый элемент второго равен сумме положительных элементов в соответствующем столбце двумерного массива, кратных 4 или 5.

15) Дан двумерный массив целых чисел. Ко всем положительным элементам массива прибавить последний элемент соответствующей строки, а к остальным - первый элемент такой же строки.

16) Дан двумерный массив из четного числа столбцов. Столбцы левой половины поменять местами со столбцами правой половины.

17) Дан двумерный массив. Удалить из него все столбцы, сумма элементов которых больше заданного числа.

18) Дан двумерный массив. Вставить в него строку из чисел 100 после строки с номером s.

19) Дан двумерный массив из 24 столбцов. Перенести первые s столбцов в конец массива, соблюдая порядок их следования.

20) Дан двумерный массив. Определить количество различных элементов в нем.

21) Дан двумерный массив целых чисел. В каждой его строке найти первый отрицательный и последний четный элементы (принять, что отрицательные и четные элементы есть в каждой строке).

Лабораторная работа № 6

Тема: Двумерные массивы

Цель: 1) Получение навыков в организации ввода-вывода элементов двумерного массива.

2) Знакомство с типовыми алгоритмами обработки массивов.

Содержание отчета:

Постановка задачи.

Текст программы с комментариями алгоритмической структуры отдельных ее частей и описанием смысла и назначения используемых переменных.

Протокол работы программы, анализ результатов.

Задание 1. Дан двумерный массив целых чисел, состоящий из n строк и m столбцов, заполненный случайным образом числами из промежутка [-100,100]. Сформировать из элементов этого массива одномерный массив, каждый элемент которого равен (по вариантам):

1. Максимальному значению элементов соответствующей строки
2. Минимальному значению элементов соответствующей строки
3. Максимальному значению элементов соответствующего столбца
4. Минимальному значению элементов соответствующего столбца
5. Сумме положительных элементов соответствующей строки
6. Сумме отрицательных элементов соответствующей строки
7. Сумме положительных элементов соответствующего столбца
8. Сумме отрицательных элементов соответствующего столбца
9. Сумме четных элементов соответствующей строки
10. Сумме нечетных элементов соответствующей строки
11. Сумме четных элементов соответствующего столбца
12. Сумме нечетных элементов соответствующего столбца

Задание 2. Задан двумерный массив целых чисел размером n*m

1. Найти номера столбцов, все элементы которых нули.
2. Найти номера столбцов, элементы в каждом из которых одинаковы.
3. Вычислить количество нулей в четных строках и количество единиц в нечетных строках.
4. Найти количество строк, содержащих все равные компоненты.
5. Найти количество строк, содержащих хотя бы одну нулевую компоненту.
6. Подсчитать количество строк, в которых отличны от нуля только три элемента.
7. Найти номер строки, содержащей наибольшее количество нулей.
8. Поменять местами в каждой строке первый элемент с первым максимальным
9. Поменять местами в каждом столбце первый элемент с первым максимальным
10. Найти номер столбца, содержащего наибольшее количество нулей.
11. Найти количество столбцов, содержащих хотя бы один отрицательный элемент.
12. Найти количество строк, все элементы которых отрицательны

Задание 3.

1. В двумерном массиве произвести сдвиг элементов таким образом, что первая строка становится второй, вторая строка становится третьей и т.д., а последняя строка -первой.

2. В двумерном массиве произвести сдвиг элементов таким образом, что второй столбец становится первым, третий столбец - вторым и т.д., а первый становится последним.

3. В таблице X(m,m) подвинуть диагональные элементы так, чтобы первый диагональный элемент стал вторым, второй - третьим, m-й - первым).

4. Дана таблица a[1:7,1:7]. Заменить наименьший элемент каждой строки, начиная со второй, наибольшим элементом предыдущей строки.

5. Заданы две матрицы a и b размером nxn. Сформировать из них прямоугольную матрицу p размером nx2n, включая в первые n столбцов матрицу a, в следующие - матрицу b.

6. Задан массив X размером n. Сформировать из него матрицу A, содержащую по l элементов в строке. Недостающие элементы в последней строке заполнить нулями. Напечатать матрицу по строкам.

7. Задана квадратная матрица. Переставить строку с максимальным элементом на главной диагонали со строкой с заданным номером.

8. Задана квадратная матрица. Исключить из нее строку и столбец, на пересечении которых расположен максимальный элемент главной диагонали.

9. Заданы матрица размером (NxN) и число k (1<=k<=N).Столбец с максимальным по модулю элементом в k-й строке переставить с k-м столбцом.

10. Заданы матрица размером (NxN) и число k (1<=k<=N). Строку с максимальным по модулю элементом в k-й строке переставить с k-й строкой.

11. Даны матрица размерностью не более чем 15x14 и произвольное число. Построить массив, каждый элемент которого представляет собой разность между этим числом и средним арифметическим для соответствующей строки матрицы. Определить, сколько элементов предшествует первому минимальному в полученном массиве.

12. Дан двумерный целочисленный массив размерностью nxn. Сформировать результирующий одномерный массив, элементами которого являются суммы элементов по строкам для тех строк, которые начинаются с k положительных чисел подряд.

Раздел 6. СТРОКОВЫЙ ТИП ДАННЫХ. МНОЖЕСТВА. ЗАПИСИ

Строковый тип данных.

Строкой называется последовательность символов кодовой таблицы ПК (цепочка символов).

Примечание. Напомним, что каждый символ имеет свой код в зависимости от используемой кодировки ASCII, КОИ-7, КОИ-8, Unicode и т.д.. В кодировке ASCII для кодирования одного символа используется 8 бит, поэтому всего можно закодировать 256 символов. То есть коды символов – это числа от 0 до 255. Первые коды символов в интервале от 0 до 32 – управляющие, они не имеют символьного представления на экране, а только выполняют определенные действия. Остальные коды отображаются на экране (например латинские заглавные буквы: “A” имеет код 65, “B” – 66 и т.д.)

При использовании в выражениях, строковая константа заключается в знаки апостроф, а для описания строковой переменной в Паскале используется ключевое слово ***string,*** после которого в квадратных скобках ([ ]) указывается максимально возможное количество символов строки. Если количество символов не указано, то по умолчанию подразумевается самое большое возможное количество символов – 256.

Таким образом, описание строкового типа в Паскале имеет следующий формат (можно описывать в разделе type, можно в var):

type <*имя типа*> = string [<*максимальное количество символов*>];
var <*имя переменной*>: string [<*максимальное количество символов*>];

Например,

```
type s1 = string [30];
var s, c : s1;  t : string; t : string [5];
```

Тип string в Паскале во многом похож на одномерный массив, элементами которого являются символы, т.е. например, описание типа string [20] аналогично описанию array [1 .. 20] of char. Потому к любому символу строки можно обращаться точно также как и к элементу одномерного массива – просто указав его индекс, т.е если

```
var s : string [15];
```

… … … …
s := ‘лекция’; то s[3] = ‘к’

Для хранения строки из n символов отводится n+1 байт памяти и в нулевом байте хранится число, равное количеству символов в строке.

15	л	е	к	ц	и	я		

Над строковыми величинами определены некоторые операции, процедуры и функции, позволяющие склеивать строки, выделять часть символов (подстроку) из строки, удалять подстроку из строки и т.д.

Рассмотрим их формат.

1. **Операция конкатенации** (склеивания, сцепления).

Склеивает несколько строк в одну и обозначается знаком “+”. Например, чтобы получить слово «паровоз»

```
s1 := ‘пар’;
s2 := ‘воз’;
s3 := s1 + ‘о’ + s2;
```

Замечание Кроме операции конкатенации существует аналогичная функция concat, соответственно предыдущая запись с использованием функции аналогична s3 := concat (s1, ‘о’, s2);

2. **Операция сравнения** (>, <, =, <>).

Строки сравниваются посимвольно – последовательно, по одному символу слева направо, причем символы сравниваются по их кодам (больше тот символ, код которого больше). Например, ‘A’ < ‘a’.

Строки считаются равными, если они имеют равное количество символов и соответствующие коды символов совпадают.

Например,

‘мама’ < ‘папа’

‘пар’ < ‘паром’

3. **Процедура удаления подстроки из строки.**

Формат:

delete (<*строка из которой удаляем*>, <*номер начальной позиции*>, <*количество удаляемых символов*>);

Например,

```
s := ‘соколок’;
delete (s, 3, 4);          {s = ‘сок’}
```

4. **Процедура вставки подстроки в строку.**

Формат:

insert (<*подстрока которую вставляем*>, <*строка, в которую вставляем*>, <*номер позиции вставки*>);

Например,

```
s := ‘пар’;
insert (‘ож’, s, 2);                {s = ‘пожар’}
```

5. **Функция копирования части строки в подстроку.**

Формат:

copy (<*строка*>, <*позиция, начиная с которой копируем*>, <*количество символов*>);

Например,

```
s := ‘паровоз’;
s1 := copy (s, 1, 3);                {s1 = ‘пар’}
```

6. **Функция поиска подстроки в строке.**

Формат:

pos (<*подстрока, которую ищем*>, <*строка в которой ищем*>);

Результат работы функции – целое число, равное номеру позиции, с которого начинается первое вхождение подстроки в строку. Если подстроки в строке нет, то результат равен 0.

Например,

```
var s1, s2 : string; n : byte;
… … … … …
s1 := ‘ко’;
s2 := ‘колокольчик’;
n := pos (s1, s2);                          {n=1}
```

7. **Функция определения длины строки.**

Длина строки – это количество символов в ней.

Формат:

length (<*строка*>);

8. **Процедуры, позволяющие преобразовать строку в число и наоборот.**

а) число $\rightarrow$ в строку. Формат:

str (<*число*>,<*строковая переменная*>);

б) строку → в число. Формат:

val (<*строка*>, n, k);

где n – полученное число-результат, а k – это число, определяющее возможность преобразования строки в число (k=0, если преобразование окончилось успешно и k=<*номеру позиции*>, с которым процедура «не справилась» - он не является цифрой).

Например,

val (‘456’, n, k); {n = 456; k = 0}

val (‘456a8’, n, k); {k = 4}

Пример 1. Составить программу, которая подсчитывает сколько раз подстрока s1 входит в строку s.

s := ‘мама пошла в магазин’

s1 := ‘ма’;

Идея: организуем цикл, в котором перебираем все символы, начиная с позиции i=1 до позиции n-k+1 (n – длина строки, k – длина подстроки) и сравниваем вырезку с i-ой позиции длиной k символов с подстрокой s1. В случае совпадения увеличиваем счетчик на единицу.

```
program chet;
var s: string; s1, s2 : string [10]; m, n, k, i : integer;
begin
    writeln (‘введите строку’);
    readln (s);
    writeln (‘введите подстроку’);
    readln (s1);
    m:= 0;
    n:= length (s);   k:= length (s1);
    for  i := 1  to  n-k+1  do
    begin
       s2:= copy(s, i, k);
       if  s1 = s2  then  m:=m+1;
    end;
    writeln (‘m = ‘, m);
end.
```

Второй способ: использовать функцию pos и процедуру delete. Для этого организовать цикл while или repeat (пока позиция вхождения не равна 0): в теле цикла ищем позицию вхождения подстроки в строку и удаляем эту часть.

```
program chet2;
var s,s1 : string; m,n: integer;
begin
     writeln ('введите строку');
     readln (s);
     writeln (‘введите подстроку’);
     readln (s1);
     m:=0;
     repeat
        n:=pos(s1,s);
        if n<>0 then
                 begin m:=m+1;
                        delete(s,n,2);
                end;
     until n=0;
     writeln (‘m = ‘, m);
end.
```

Пример 2. Дан текст, состоящий из слов, разделенных пробелами (одним или несколькими). Составить программу, записывающую все слова этого текста в строковый массив.

Например, ‘мама пошла в магазин’

a1= мама, a2 = пошла, …

1 способ решения. Необходимо организовать вложенные циклы: внешний – для просмотра всей строки, и два цикла внутри него: в первом выделяется слово (символы не являющиеся пробелами), а во втором пропускаются все пробелы до следующего слова. Каждое слово собирается в переменной строкового типа s1, до начала первого внутреннего цикла она очищается (s1:=’ ’), а по окончании ее значение заносится в массив.

```
program slova_stroki;
var s, s1 : string;   a: array [1 .. 20] of string [15];   i, n, k : integer;
begin
     writeln (‘введите строку’);
     readln (s);
     i := 1;    k := 0;     n := length (s);
     while i <= n do                              {внешний цикл}
     begin
          s1:=’’; {пустая строка - слово}
```

```
        while (s[i]<>'  ') and (i <= n) do
        begin                                   {цикл для выделения одного
слова}
           s1:=s1+s[i];       i := i + 1;
        end;
        k := k + 1;       a[k]:=s1;             {заносим слово в массив}
        while  s[i] = '  '  do
           i := i + 1;                          {цикл для пропуска пробелов}
    end;
    writeln ('количество слов =', k);
    for i := 1 to k do                          {вывод результата}
       writeln (a[i]);
end.
```

2 способ решения. Организуем цикл «для», в котором параметр изменяется от 1 до n (длины слова). В теле цикла мы выделяем один символ и с помощью полного ветвления анализируем его: если этот символ не пробел, то добавляем его к слову, а иначе очищаем строку, в которой собираем слово, предварительно добавив полученное слово в массив.

```
program slova_stroki_2;
var s, s1 : string;   a: array [1 .. 20] of string [15];   i, n, k : integer;
begin
    writeln ('введите строку');
    readln (s);
    s1:=''; {пустая строка -слово}    k := 0;     n := length (s);
    for i := 1  to  n  do
        if s[i]<>'  ' then {символ не пробел, добавляем к слову }
                        s1:=s1+s[i]
                   else begin {пробел, первый ли после слова? }
                        if s1<>'' then begin
         {первый, добавляем слово в массив}
                                              k := k+1;
                                             a[k] := s1;
                                          end;
                                s1:=''; {очищаем строку-слово}
                             end;
     {последнее слово добавляем в массив}
                        k := k+1;
                        a[k] := s1;
```

```
    writeln (‘количество слов =’, k);
    for i := 1 to k do
        writeln (a[i]);
end.
```

Множественный тип данных

В языке Pascal множественный тип данных (множество) – это набор данных одного типа, логически связанных между собой. При этом логическая связь отслеживается только программистом, а для компилятора Pascal она отсутствует.

В качестве типа элементов множества может выступать любой ограниченный скалярный тип, который называется базовым типом, т.е. базовым не может быть вещественный и неограниченный целый типы, а в основном – тип char (лучше его диапазон) и диапазон из типа integer. Количество элементов в базовом типе не должно быть больше 256.

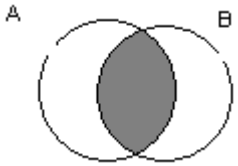
Описание множественного типа задается при помощи ключевого слова set, после которого указывается базовый тип. Можно описывать либо в разделе type, либо в разделе var:

```
type <имя типа> = set of  <базовый тип>;
var <имя переменных> : set of  <базовый тип>;
```

Например,

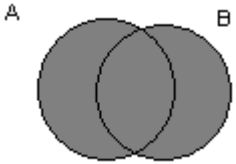
```
type  m1 =  set of  char;
      m2 =  set of  10 .. 99;
var p1, p2 : set of  m1;   p3 : set of  ‘1’ .. ‘9’;
```

Над множествами в языке Pascal определены операции объединения, пересечения и разности, обозначаемые соответственно знаками +(⋃), *(⋂), –(/).

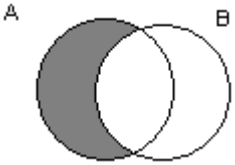


A * B

Пересечением множества A и B называется совокупность элементов, принадлежащих одновременно и множеству A и множеству B.



A + B
Объединением множеств A и B называется совокупность элементов, принадлежащих либо множеству A, либо B



A – B
Разностью множеств A и B называется совокупность ‘элементов множества A, не принадлежащих множеству B

Над множествами определены операции отношения =, <>, >=, <=, <, >. Множество A равно множеству B, если равны элементы этих множеств (количество и величины), а порядок следования элементов при этом не важен. В языке Pascal множество обозначается квадратными скобками (элементы множества заключаются в квадратные скобки).

Например,
A = [7,5,4,1] и B = [4,1,7,5] $\Rightarrow$ A = B.

Множество A меньше множества B, если любой элемент множества A является элементом множества B.

Для определения принадлежности элемента некоторому множеству в Pascal используется операция in. Результат этой операции имеет булевский тип и, соответственно, равен true, если элемент принадлежит множеству и false, если не принадлежит.

Например,
7 in A – true
8 in A – false

Необходимо помнить, что элементы множественного типа нельзя вводить и выводить в программах. Поэтому, обычно, значения элементов множества задаются с помощью команды присваивания, а для вывода организуется специальный цикл с параметром, который пробегает все значения базового типа. В теле цикла определяется принадлежность параметра множеству: если принадлежит, то его значение выводится на экран. При этом значения элементов множества выводятся не в том порядке, в котором они записаны во множестве, а в порядке возрастания,

причем если какое-то значение элемента встречается во множестве несколько раз, то оно выводится один раз.

Эти ограничения по вводу и выводу приводят к тому, что множественный тип используется в основном внутри программ как вспомогательный, для решения каких-либо задач. Чаще всего – это проверка принадлежности значения какой-либо переменной определенному множеству, которое заменяет составное условие, или определяет такое условие, которое в терминах алгебры логики практически невозможно записать.

Например, условие

```
if  (a=2) or (a=5) or (a=7) or (a=8)  then …
```

можно заменить условием

```
if  a  in  [2,5,7,8]  then …
```

Пример 1. Из чисел, заключенных в интервале от 1 до 100, сформировать множества чисел, кратных 2, кратных 3, кратных 6 (одновременно 2 и 3), кратных 2 или 3.

Для формирования множеств D2 и D3 необходимо организовать цикл, в котором параметр будет изменяться от 1 до 100, а в теле цикла, выделять числа, которые делятся на 2 и 3 и добавлять их к соответствующему множеству с помощью операции сложения. Перед циклом D2 и D3 присваивать значение пустого множества [ ].

Множество D6 - пересечение множеств D2 и D3, а множество D23- объединение D2 и D3. На экран надо будет выводить элементы четырех множеств, поэтому будем использовать процедуру.

```
program pr_set;
      const n=100;
      type mn =set of 1 . . n;
      var D2,D3,D6,D23: mn;
            i:byte;
        procedure print_mn (D:mn);
          var j:byte;
        begin
            for j:=1 to n do
              if j in D then write (j:4);
              writeln;
        end;
begin
        D2:=[ ];
        D3:=[ ];
      for i:=1 to n do begin
          if i mod 2 =0 then D2:=D2+[i];
```

```
        if i mod 3 =0 then D3:=D3+[i]
                        end;
    D6:=D2*D3;
    D23:=D2+D3;
    writeln('множество чисел, делящихся на 2');
          print_ mn(D2);
    writeln('множество чисел, делящихся на 3');
          print_mn (D3);
    writeln('множество чисел, делящихся на 2 и на 3');
          print_mn (D6);
   writeln('множество чисел, делящихся на 2 или на 3');
          print_mn (D23);
    end.
```

Пример 2. Определить количество гласных и согласных букв в предложении на русском языке.

Вспомним, как кодируются русские буквы:

Заглавные: А -128, . . .,.Я = 159

Строчные: а – 160,. . ., п - 175, потом разрыв кодовой таблицы и р – 224, …, я -.239.

Из приведенной кодировки видно, что множество всех русских букв можно задать из двух интервалов - 'А'..'п' , 'р'..'я'. Множество всех гласных букв задаем перечислением, а множество согласных находим как разность множества всех русских букв и множества гласных букв.

Исходный текст будем вводить в переменную строкового типа. В начале программы сформируем множества гласных и согласных букв. Для подсчета гласных и согласных букв в тексте организуем цикл с параметром i, изменяющимся от 1 до длины текста. В теле цикла будем проверять условие принадлежности i-го символа текста соответствующему множеству и увеличивать счетчик гласных или согласных:

```
program  podschet_bukv;
     type m =set of char;
     var    m1, m2, m3 : m;
          t : string;  i,k1,k2 : byte;
 begin
  m1 := ['А'..'п' , 'р'..'я']  {все русские буквы}
 {перечисляем множество  гласных букв}
  m2:=['А','Е','И','О','У','Э','Ю','Я','а','е','и','о','у','э','ю','я']
  m3 := m1-m2;  {согласные русские буквы}
     wtiteln ('Введите предложение');
      readln (t);
      k1:=0;  k2:=0;
```

```
    for i:=1 to length (t) do begin
        it t[i] in m2  then k1:=k1+1;
        it t[i] in m3  then k2:=k2+1;
                                end;
    writeln (k1,k2);
end.
```

Контрольные вопросы и задания

Вопросы:

1. Дайте определение строки.
2. основные операции для обработки строк и их смысл
3. основные функции для обработки строк и их смысл
4. основные процедуры для обработки строк и их смысл
5. Какие основные операции определены над множествами? ?
6. Каков формат описания множества, что в нем указывается?
7. Каким образом в Паскале выводятся элементы множеств
8. Какие типовые задачи обработки строковых величин можно выделить?
9. В каких задач обработки строковых величин целесообразно использовать множества?

Задания:

1. Дана строка символов. Выделить подстроку между первой и второй точкой.
2. Дана строка символов. Выделить подстроку между первой и последней точкой.
3. Из строки удалить среднюю букву, если длина строки нечетная, иначе – удалить две средние буквы.
4. Заменить все вхождения строки str1 на подстроку str2 (str1 и str2 вводятся с клавиатуры).
5. Дана строка символов. После каждого символа ch вставить строку str1.
6. Дана строка символов. Удалить из нее все знаки препинания.
7. Дана строка символов. Заменить в ней все 'A' на 'AAA'.
8. Дана последовательность слов, разделенных запятыми. Напечатать все слова, отличные от слова hello.
9. Составить программу вывода самой большой цифры в строковой записи данного числа.
10. Дан непустой текст из заглавных русских букв, за которым следует точка. Определить, упорядочены ли эти буквы по алфавиту.

11. Дана строка символов до точки. Определить, является ли она правильным скобочным выражением (число открывающихся скобок равно числу закрывающихся). Рассмотреть только круглые скобки.

12. Дана строка символов. Определить, является ли она записью целого десятичного числа, кратного двум

Лабораторная работа № 7

Тема: Строковый и множественный типы данных

Цель: 1) Знакомство с операциями, процедурами и функциями обработки строковых величин

2) Получение навыков в задании переменных множественного типа и выполнении операций над ними

3) Знакомство с задачами, в которых целесообразно использовать переменные множественных и строковых типов

Содержание отчета:

1. Постановка задачи.
2. Текст программы с комментариями алгоритмической структуры отдельных ее частей и описанием смысла и назначения используемых переменных.
3. Протокол работы программы, анализ результатов.

Задание 1.

Дана непустая строка символов. Требуется построить и напечатать множество, элементами которого являются встречающиеся в строке:

1) буквы от ‘а ‘ до ‘ж’ и цифры от ‘4’ до ‘9’
2) буквы от ‘ф ‘ до ‘ю’ и цифры от ‘1’ до ‘7’
3) буквы от ‘з ‘ до ‘ф’ и цифры от ‘2’ до ‘9’
4) буквы от ‘г ‘ до ‘л’ и знаки препинания
5) знаки препинания и цифры от ‘0’ до ‘6’
6) знаки арифметических операций и цифры
7) знаки арифметических операций и буквы от ‘Ф ‘ до ‘Я’
8) знаки препинания и буквы от ‘З’ до ‘Ш’
9) знаки препинания и буквы от ‘D’ до ‘L’
10) знаки операций отношения и цифры
11) знаки операций отношения и буквы от ‘f ‘ до ‘q’
12) знаки арифметических операций и цифры от ‘0 ‘ до ‘8’

Задание 2.

Дан текст, состоящий из слов: между словами произвольное число пробелов, после последнего слова точка.

1) Найти количество слов, содержащих ровно две буквы ‘о’
2) Вывести на экран слова, состоящие из 5 букв
3) Найти количество слов, у которых первая и последняя буква одинаковые
4) Найти количество слов, содержащих сочетание ‘ан’
5) Вывести на экран слова, содержащие одновременно буквы ‘а’ и ’о’
6) Напечатать те слова, в которых первая буква встречается еще раз
7) Вывести на экран слова с максимальной длиной
8) Вывести на экран слова с минимальной длиной
9) Напечатать все слова в алфавитном порядке
10) Напечатать все слова, перенеся в них первую букву в конец слова
11) Напечатать все слова, удалив из них первую букву
12) Напечатать все слова, удалив из них две последних буквы

Задание 3.

Дан текст, состоящий из слов, содержащих только строчные русские буквы; между соседними словами – запятая, после последнего слова – точка.

Напечатать в алфавитном порядке:

1) все гласные буквы, которые входят в каждое слово
2) все согласные буквы, которые входят в каждое слово
3) все гласные буквы, которые не входят ни в одно слово
4) все согласные буквы, которые не входят ни в одно слово
5) все звонкие согласные буквы, которые входят хотя бы в одно слово
6) все глухие согласные буквы, которые не входят хотя бы в одно слово
7) все согласные буквы, которые входят только в одно слово
8) все гласные буквы, которые входят хотя бы в одно слово
9) все звонкие согласные буквы, которые не входят только в одно слово
10) все гласные буквы, которые не входят более чем в одно слово
11) все глухие согласные буквы, которые не входят только в одно слово

12) все глухие согласные буквы, которые входят хотя бы в одно слово

(*Примечание:* гласные буквы – а,е,и,о,у,ы,э,ю,я; согласные – все остальные, кроме й,ь,ъ; звонкие согласные – б,в,г,д,ж,з,л,м,н,р; глухие согласные – к,п,с,т,ф,х,ц,ч,ш,щ)

Комбинированный тип данных (записи)

При обработке большого количества данных с помощью массивов основное ограничение состоит в том, что элементы массива должны иметь одинаковый тип. На практике же часто возникают задачи, в которых требуется обработать большое количество некоторых объектов, каждый из которых имеет несколько характеристик и они имеют разный тип данных.

Например, пусть требуется обработать информацию об успеваемости студентов какой-то группы. С каждым студентом связаны следующие характеристики:

фамилия и инициалы, оценки по 4 предметам, средний балл студента. Для их описания в программе будут использоваться переменные следующих типов:

```
fio : string [20]
01, 02, 03, 04 : 2 . . 5
 sr : real
```

Для того, чтобы обрабатывать данные об успеваемости группы с помощью массива, необходимо соответственно ввести 3 массива, и при обработке данных об успеваемости соответствие между массивами проследживать по индексам. Это не совсем удобно и для таких случаев лучше использовать специальный тип данных языка Паскаль – ***записи***.

Запись характеризует объект, который состоит из нескольких признаков – частей, называемых полями. Каждое поле может иметь свой тип. Тип поля может быть любым, в том числе и типа запись.

Формат описания записей в разделе TYPE :

```
< имя типа>  = record
                  <имя поля 1> : <тип 1>;
                   <имя поля 2> : <тип2>;
                    . . . . . .. . .
                   < имя поля n> : < тип n>;
                 end;
```

Соответственно для рассмотренного выше примера данных об успеваемости группы студентов каждого студента можно охарактеризовать типом запись следующего вида:

```
TYPE
    Student = record
                fio:string [20];
                01,02,03,04 :2. . 5;
                sr_ball:real:
              end;
```

Затем можно рассмотреть группу студентов как массив, элементами которого являются записи, то есть ввести следующий тип:

```
gruppa = array [1. .25] of student;
```

тогда i-й студент группы будет являться элементом массива. Введем переменные соответствующих типов.

```
var
    st: student;
    gr: gruppa;
```

Для характеристики одного студента при использовании записей поля добавляются к имени записей и в качестве разделителя используется знак “.” (точка). То есть можно выполнить следующие операторы присваивания:

```
st.fio:=’Иванов И.И.’
st.01 : =3; st.02 := 4; . . . . . .
st.sr_ball := (01+02+03+04)/4
```

Для того, чтобы группе студентов присвоить соответствующие характеристики, мы должны выполнить эти присваивания для каждого студента группы, используя, соответственно, вместо операторов присваивания фамилии и оценок, операторы ввода, и организовав цикл по индексу массива, то есть соответствующий фрагмент программы будет иметь вид:

```
for i:=1 to 25 do
    begin
        readln (gr[i].fio);
        readln (gr[i].01, gr[i].02, gr[i].03, gr[i].04)
    end;
```

Если какое-либо поле записи в свою очередь имеет тип запись, то для конкретного поля конструируется более длинное имя, в котором к имени записи добавляются имена двух, трех и так далее полей.

Например,

```
type
    student = record
```

```
                fio : record
                        f : string [15];
                        im : string [10];
                        ot :s tring [15]
                      end;
                 d_r : record
                      d : 1 . .31;
                      m : 1. .12;
                      g : 1978. .1982
                        end;
                 end;
var
    st: student;
    gr: gruppa;
```

Тогда для присваивания значений одному студенту используются конструкции:

```
st.fio.f := 'Иванов'
st.fio.im := 'Иван'
st.fio.ot : 'Иванович'
```

и аналогично для даты. То есть получаются довольно таки длинные имена. В этих случаях для сокращения записи можно использовать специальный оператор присоединения with.

Формат оператора:

```
with  < имя записи[.поле]>  do
        begin
              <операторы>
        end;
```

Для соответствующего примера:

```
with st.fio     do
        begin   f := 'Иванов'
                im := 'Иван'
                ot: = 'Иванович'
            end;
```

Пример:

Пусть имеются данные об успеваемости группы студентов: Ф.И.О. и оценки по четырем предметам каждого студента. Обработать информацию об успеваемости: подсчитать средний балл каждого студента, средний балл группы и вывести на экран фамилии студентов, средний балл которых выше, чем средний балл группы.

Данные о каждом студенте оформим в виде записи, группа студентов - массив записей.

```
program uspevaemost;
        type pupil = record
                          fio:string 20;
                          0:array [1. .4] of 2. .5;
                          sr:real
                     end;
             mas = array [1. .30] of pupil;
        var
             gruppa : mas;
             j,n,i : integer;
             sr_b_g : real;
 begin
      write (‘Введите число студентов в  группе   ‘);
      readln (n);
{Организуем цикл для ввода исходных данных о студентах}
      writeln (‘Введите данные о студентах.);
      for i:=1 to n do
         begin
           with gruppa [i] do
               begin
                 write (‘фамилия и инициалы’)
                readln (fio);
                write (‘введите 4 оценки через пробел  ’);
                       for j:=1 to 4 do read (0[j]);
                       readln;{для перевода строки}
               end;
          end;
{подсчет среднего балла студентов и группы в целом}
 sr_b_g:=0; {начальное присваивание - сред балл группы}
 for i:=1 to n do
     begin
        gruppa [i].sr:=(0[1]+0[2]+0[3]+0[4])/4;
        sr_b_g := sr_b_g +gruppa[i].sr]
     end;
  sr_b_g:=sr_b_g/n;
{вывод на экран фамилий студентов,  у которых большой средний балл}
   for i:=1 to n do
        with gruppa[i] do
```

```
            if sr > sr_b_g  then writeln (fio);
    end;
```

Контрольные вопросы и задания

Вопросы:

1. Почему запись называют комбинированным типом данных?
2. Как определяется тип записи? Что называется полем записи?
3. Какие требования предъявляются к идентификаторам поля в записи?
4. Что такое составное имя поля записи? Из каких частей оно состоит и как записывается?
5. Для чего при обращении к полю записи используется ключевое слово with?

Лабораторная работа № 8

Тема: Комбинированный тип данных

Цель: 1) Получение навыков в задании переменных комбинированного типа и выполнении операций над ними

2) Знакомство с задачами, в которых целесообразно использовать переменные типа запись

Содержание отчета:

1. Постановка задачи.
2. Текст программы с комментариями алгоритмической структуры отдельных ее частей и описанием смысла и назначения используемых переменных.
3. Протокол работы программы, анализ результатов.

Задание 1. Составить программу, использующую массив записей (в соответствии с номером варианта). Организовать ввод информации с клавиатуры, вывод всей информации на экран в форме таблицы и результата работы в соответствии с вариантом:

1) В записной книжке указаны фамилии и номера телефонов 30 человек. Составить программу, которая определяет, есть ли в записной книжке телефон некоторого человека, и, если есть, печатающую номер его телефона.

2) В записной книжке указаны фамилии и номера телефонов 30 человек. Составить программу, которая определяет, есть ли в записной книжке информация о человеке с заданным номером телефона, и, если есть, печатающую фамилию этого человека.

3) Известны фамилии, адреса и телефоны 25 человек. Найти фамилии и адреса людей, чей телефон начинается с цифры "3". Телефон задан в виде 7-ми значного числа.

4) Известны фамилии, адреса и телефоны 25 человек. Найти фамилии и адреса людей, чей телефон начинается с цифры "3". Телефон задан в виде, аналогичном следующему: 268-50-59.

5) Известны данные о 20 сотрудниках фирмы (фамилия, зарплата и пол). Определить фамилию мужчины, имеющего самую большую зарплату (считать, что такой есть и он единственный).

6) Известны данные о 20 сотрудниках фирмы (фамилия, зарплата и пол). Определитьфамилии мужчины и женщины, имеющих самую маленькую зарплату (считать, что такие есть и они единственные в своей группе сотрудников).

7) Известны данные о 16 сотрудниках фирмы: фамилия, вораст, отношение к воинской службе (военнообязанный или нет). Определить фамилию самого младшего по возрасту человека среди военнообязанных (считать, что такой есть и он единственный).

8) Известны данные о 16 сотрудниках фирмы: фамилия, возраст, отношение к воинской службе (военнообязанный или нет). Определить фамилии самых старших по возрасту людей среди военнообязанных и среди невоеннообязанных (считать, что такие есть и они единственные в своей группе).

9) Известны фамилии 25 человек, их семейное положение: женат (замужем) или нет, и сведения о наличии детей (есть или нет). Определить фамилии женатых (замужних) людей, имеющих детей.

10) Известны данные о тридцати учениках: фамилия, класс и оценка по информатике. Определить фамилии учеников в девятых классах, имеющих оценку "5".

11) Известна информация о 28 учениках нескольких школ, занимающихся в районном доме творчества учащихся (фамилия, имя, адрес, номер школы и класс). Вывести фамилию, имя, адрес тех учеников, которые учатся в данной школе в старших классах (10 - 11-х), записать в отдельный массив.

12) Известны данные о 20 учениках класса: фамилии, имена, отчества, даты рождения (год, номер месяца и число). Определить, есть ли в классе ученики, у которых сегодня день рождения, и если да, то напечатать имя и фамилию каждого.

Раздел 7. ФАЙЛОВЫЙ ТИП ДАННЫХ

Файловый тип данных. Процедуры и функции для работы с файлами

В рассмотренных ранее программах исходные данные вводились с клавиатуры, а результаты выводились на экран, т.е. при каждом запуске программы надо было заново набирать на клавиатуре данные, а результаты переписывать в тетрадь (на бумагу). При большом количестве исходных данных и результатов это неудобно. В практических задачах, в основном, данные хранятся во внешней памяти в виде файлов. Во всех языках программирования соответственно предусмотрены средства для работы с файлами на дисках.

Примечание. Напомним, что файл – это область информации во внешней памяти, имеющая имя. В операционной системе MS-DOS имя файла состоит из латинских букв, причем длина имени не превышает 8 символов, а расширение – не более 3 символов.

В Паскале различают 3 типа файлов, используемых в программах (файловых переменных):

1. типизированные файлы
2. текстовые файлы
3. нетипизированные файлы

Рассмотрим формат описания этих типов.

1. Формат описания типизированных файлов:

type <*имя типа*> = file of < *тип элементов файла* >;

var <*имя файловой переменной*> : file of < *тип элементов файла*>;

Например,

```
type f1 = file of real;
var f, g : f1; f2, h : file of char;
```

В качестве типа элементов файла может выступать любой тип, кроме файлового.

2. Формат описания текстовых файлов:

var <*имя файловой переменной*> : text;

Например, var f1, f2 : text;

3. Формат описания нетипизированны файлов:

var <*имя файловой переменной*> : file;

Переменные файловых типов в Паскалеl можно представить в виде ленты, разделенной на отдельные элементы, которые нумеруются с нуля. В каждый момент времени из программы доступен какой-либо один элемент из файла, на который указывает текущий указатель – маркер.

Если в программе проводится операция считывания элемента из файла на диске или операция записи на диск, то явно не указывается номер элемента, к которому мы обращаемся, т.к. всегда обрабатывается текущий элемент. После обработки текущего элемента, указатель автоматически перемещается на следующий элемент, т.е. на 1 позицию вправо. Обычно, вначале обработки файла текущий указатель (маркер) указывает на нулевой элемент, а по мере обработки элементов последовательно смещается вправо.

Различают файлы прямого и последовательного доступа.

В файлах последовательного доступа для обращения к конкретному элементу файла необходимо сначала последовательно считать все элементы файла, по очереди от начала до нужного элемента.

В файлах прямого доступа можно с помощью соответствующей процедуры сразу установить указатель на элемент с нужным номером, т.е. сразу обрабатывать конкретный элемент.

Рассмотрим основные процедуры и функции для работы с файловыми переменными.

1. Первоначально в программе нужно установить связь между физически существующим (создающимся) файлом на диске и соответствующей ему файловой переменной в программе. Для этого существует процедура

assign (<*имя файловой переменной в программе*>, <*имя файла на диске*>)

Имя файла на диске может быть как строковой константой, так и строковой переменной. Необходимо помнить, что когда файл находится не в текущей папке, в имени файла необходимо указывать полный путь к нему.

Например, assign (f, ‘D\gr1\cod\dan’);

Для того чтобы программа могла обрабатывать данные из разных файлов, перед вызовом этой процедуры следует организовать ввод имени файла с клавиатуры.

2. Процедура reset (<*имя файловой переменной*>) открывает файл для чтения. При этом текущий указатель (маркер) устанавливается на начало файла (нулевой компонент).

3. Процедура rewrite (<*имя файловой переменной*>) открывает файл для записи в него элементов. При этом если файла на диске, соответствующего имени файловой переменной не было, то он создается и указатель устанавливается в начале. Если на диске существовал файл с указанным именем, то файл создается заново, то есть все данные в этом файле будут потеряны.

Замечание. В случае типизированных файлов, при открытии файла для чтения процедурой reset он становится доступным не только для чтения, но и для записи.

4. Процедура close (<*имя файловой переменной*>) закрывает открытый в программе файл. Эту процедуру нужно не забывать использовать в конце программы обработки файлов и, кроме того, она необходима в том случае, если в программе надо несколько раз открывать для чтения один и тот же файл: повторно открывать нельзя, предварительно надо обязательно его закрыть.

5. Рассмотренные процедуры не выполняют каких-то действий с отдельными элементами файла, а просто целиком подготавливают файл к работе. Основная же часть программы при работе с файлами состоит в извлечении (считывании) отдельного элемента из файла (внешней памяти) в переменную программы (оперативную память), или наоборот – запись значения переменной из программы в файл. Для этих целей используются те же операции (процедуры) ввода и вывода элементов, что и с клавиатуры (read, write).

В случае чтения из файла или записи в файл эти процедуры имеют соответствующий формат:

а) ввод (чтение) из файла
read (<*имя файловой переменной*>, <*список имен переменных*>)

б) вывод в файл
write (<*имя файловой переменной*>, <*список вывода*>)

6. Логическая функция eof(<*имя файловой переменной*>) определяет, достигнут ли конец файла и принимает значение true, если конец достигнут и false – если не достигнут.

Рассмотренные процедуры и функции 1 – 6 позволяют работать с файлами всех типов и выполнять основные операции.

Пример 1. Составить программу, формирующую на диске файл из целых чисел, сгенерированных случайным образом в интервале [-20, 30]. Количество чисел n вводится с клавиатуры.

Так как количество чисел известно, то в основной программе следует организовать цикл типа «для», в теле цикла получать одно случайное число и записывать его в файл. До начала цикла следует подготовить файл для записи и не забыть закрыть его по окончании.

```
program primer_zapisi_faila;
var n, i, x : integer;   f: file of integer;
begin
    writeln ('введите количество элементов');
    readln (n);
    assign (f, 'chisla.dat');
    rewrite (f);
    randomize;
    for i := 1 to  n  do  begin
          x := random(51) – 20;
          write (f, x);   end;
    close (f);
end.
```

Пример 2. Составить программу, формирующую на диске файл из вещественных чисел, вводимых с клавиатуры. Признак конца ввода: число -99999.

От предыдущей программа отличается тем, что вместо цикла «для» следует использовать цикл «пока» или «до», а очередной элемент файла вводить с клавиатуры:

```
program primer_zapisi_faila_2;
var  x : real;   f: file of real;
begin
    assign (f, 'f1.dat');
    rewrite (f);
    write ('введите элемент');
    readln (x);
    while x <> -99999  do  begin
          write (f, x);
          write ('введите следующий элемент или -99999 для окон-
                      чания работы программы');
          readln (x);       end;
    close (f);
end.
```

Пример 3. Составить программу, считывающую с диска файл целых чисел и выводящую на экран элементы этого файла.

```
program vivod_iz_faila;
var  x : integer;   f: file of integer;
begin
     assign (f, 'chisla.dat');
     reset (f);
     while  not (eof(f))  do
     begin
        read (f, x);
        write (x : 6);
     end;
     close (f);
end.
```

Процедуры и функции для работы с файлами прямого доступа и файлами на диске

В типизированных файлах можно организовать прямой доступ к элементам, используя для этого следующие процедуры и функции:

1) Процедура

seek (<имя файловой переменной>,<номер позиции>)

– позволяет установить текущий указатель файла на элемент с заданным номером (не забывать, что элементы нумеруются с 0).

2) Функция

filesize (<имя файловой переменной>)

– позволяет определить длину файла, то есть количество элементов в нем.

Комбинируя 1) и 2), можно установить текущий указатель на конец файла, то есть подготовить файл для добавления в него элементов:

seek (<имя файловой переменной>, **filesize** (<имя файловой переменной>)).

3) Функция

filepos (<имя файловой переменной>)

– позволяет определить номер текущей позиции в файле, то есть определить, на какой элемент указывает маркер в данный момент.

Для всех типов файлов в языке Паскаль определен ряд процедур, позволяющих работать с файлами на диске, то есть непосредственно из программы выполнять действия, которые обычно выполняются в среде

операционной системы (копирование, переименование, удаление файлов и т.д.).

Рассмотрим две из них:

1) erase (<имя файловой переменной>) – удаляет файл на диске.

2) rename (<имя файловой переменной>,<имя файла на диске>) – переименовывает файл.

Перед выполнением этих процедур необходимо, чтобы файлы были закрыты.

Эти две процедуры используют в программах в том случае, когда необходимо добавить данные к существующему файлу (не используя прямой доступ) или каким – то образом изменить содержимое файла (удалить часть элементов, добавить элементы в середине файла и т.д.). Для этого поступают следующим образом: открывают исходный файл для чтения и создают некоторый промежуточный файл, открыв его для записи. В этот промежуточный файл переписывают элементы из исходного файла, изменяя их при необходимости, добавляя новые и т.д., то есть, в промежуточном файле получается требуемый результат. Затем исходный файл удаляется, а промежуточный – переименовывается в имя исходного файла.

Пример: Дан файл целых чисел, содержащий не меньше десяти элементов. Добавить в этот файл новые элементы (количество вводится с клавиатуры), вставив их после пятого элемента исходного файла.

```
Program pr;
    Var   f1, f2: file of integer;
            i, x, n: integer;
begin
  assign(f1, 'ish.dat');
  assign(f2, 'prom.dat');
  reset(f1);
  rewrite(f2);
{переписываем первые 5 элементов в промежуточный файл}
  for i:=1 to 5 do
     begin
          read(f1,x);
         write(f2,x);
    end;
{вводим с клавиатуры и добавляем в файл новые элементы}
   write('введите количество элементов ');
   readln(n);
   for i:=1 to n do
       begin
```

```
          write('введите значение элемента ');
          readln(x);
          write(f2,x);
       end;
{дописываем оставшиеся элементы в промежуточный файл}
  while not (eof(f1)) do
       begin
          read(f1,x);
          write(f2,x);
       end;
  close(f1);
  close(f2);
  erase(f1);
  rename(f2,'ish.dat');
end.
```

Задача Самостоятельно составьте программу для решения этой же задачи, используя процедуры и функции прямого доступа к файлу.

Текстовые файлы

Текстовые файлы – это последовательность символьных строк переменной длины (типа string) , каждая из которых заканчивается управляющим символом конца строки - последовательностью ASCII кодов 13+10. Для определения этого символа, то есть того, достигнут или нет конец строки, в Паскале используется *функция*

eoln*(< имя файловой переменной>)*

Результат ее работы имеет булевский тип. *true* - если достигнут конец, *false* - если не достигнут.

Для текстовых файлов дополнительно к процедурам и функциям, рассмотренным выше, применимы следующие дополнительные процедуры:

1) append *(< имя файловой переменной>)*– открывает файл для добавления в него элементов.

2) Осуществлять запись в текстовые файлы конкретных элементов и чтение из него можно с помощью рассмотренных ранее процедур *write* и *read,* но очевидно, что с их помощью мы можем записать в файл (считать из файла) только один символ. Для считывания и записи целиком строки файла используются соответственно процедуры:

readln (<имя файловой переменной>,<имя переменной строкового типа>);

writeln (<имя файловой переменной>,<имя переменной строкового типа>).

Пример: Составить программу, которая считывает с диска текстовый файл, выводит его содержимое на экран, подсчитывает количество букв ‘а’ в каждой строке файла, а также общее количество строк в файле. Эту задачу можно решить двумя способами: I способ – с помощью построчного считывания файлов, II способ – с помощью посимвольного считывания файлов.

I способ – с помощью построчного считывания

```
Program pr1;
   Var f: text;
         s: string;
          k0,ks,i: integer;
begin
   assign (f,'f1.txt');
   reset(f);
   k0:=0; {обнулили счетчик общего количества строк}
{открываем цикл для считывания строк файла до конца}
   while not(eof (f)) do
   begin
     readln (f, s); {считана одна строка}
     writeln (s); {выводим ее на экран}
     k0:=k0+1; {увеличили значение счетчика количества строк}
      ks:=0;       {обнулили счетчик букв а в строке}
     for i:=1 to length(s) do {открываем цикл по строке}
     if s[i]='a' then ks:= ks+1; {увеличиваем счетчик количества
букв а в строке}
      writeln ('в строке ', k0,'число букв а =',ks); {выводим на
экран}
    end;
  writeln('общее количество строк в файле ', k0);
close(f);
end.
```

II способ: посимвольное считывание из файла.

Для этого необходимо организовать вложенные циклы: внешний – пока не конец файла, а внутренний – пока не конец строки. При этом во внутреннем цикле будет считываться по одному символу при каждом повторении тела цикла и по его окончании (выходе из него) необходимо пропустить символ конца строки, то есть перейти на следующую строку. Он считывается при помощи процедуры readln (f) (опускается второй параметр). Соответственно, не забываем обнулять счетчики (сам файл выводить на экран не будем).

```
Program pr1_sposob2;
  Var f: text;
       c: char;
       k0, ks: integer;
begin
   assign (f,'f1.txt');
   reset(f);
   k0:=0; {обнулили счетчик общего количества строк}
   while not (eof(f)) do
   begin
      ks:=0; {обнулили счетчик букв а в  строке}
      while not(eoln(f)) do {открываем внутренний цикл по  строке}
      begin
         read(f,c); {читаем из файла один символ}
         if c=a then ks :=ks+1;
      end; {конец внутреннего цикла}
      readln (f); {перевод строки в файле}
      k0:=k0+1; {увеличили значение счетчика количества строк}
      writeln ('в строке ', k0,'  число букв а равно  ', ks);
    end;  {конец внешнего цикла}
    writeln ('количество строк в файле - ', k0);
    close(f);
end.
```

Пример: Составить программу, которая создает на диске текстовый файл из строк, введенных с клавиатуры. Количество строк, записываемых в файл, заранее не известно, признаком их конца служит ввод пустой строки, то есть просто нажатие клавиши Enter.

Для решения этой задачи организуется цикл типа «до», в теле которого будем вводить с клавиатуры одну строку и записывать ее в текстовый файл. Условие окончания цикла – пустая строка.

```
Program pr2;
     Var f: text;
          s: string;
begin
       assign(f,'f2.txt');
       rewrite(f);
       repeat
            readln (s); {ввод одной строки с клавиатуры}
            writeln(f,s); {запись строки в файл}
       until s='';
      close(f);
end.
```

Файлы с записями

Как мы отмечали, элементы файла могут иметь любой тип, кроме файлового. Особый интерес в практических задачах представляют файлы, элементы которого имеют тип запись. Фактически такой файл представляет собой таблицу, каждая строка которой содержит сведения о каком–либо элементе – объекте. Такие файлы образуют базу данных во внешней памяти компьютера. Существуют специальные программы – система управления базами данных, которые предназначены для организации таких баз данных. В то же время простейшие базы данных можно создавать непосредственно используя процедуры и функции для работы с файлами в языке Паскаль.

Рассмотрим **пример:**

Требуется обрабатывать информацию об участниках группы «здоровья». Каждый член группы характеризуется следующими параметрами: фамилия, имя, возраст, вес. Составить три программы для управления базой данных, позволющих:

1. создавать на диске файл, содержащий сведения об участниках группы;
2. выводить информацию об участниках в виде таблицы на экран;
3. выводить информацию о тех участниках, которые имеют избыточный вес.

Программа 1

```
Program pr1;
  Type gr=record
                fam: string [20];
                name: string [20];
                vozr: byte;
                rost: 140..230;
               ves: real;
   var  f: file of gr;
        uch: gr;
begin
     assign (f, 'zdorov.dat');
     rewrite(f);
     repeat
      write('введите фамилию участника, или *** - конец');
      readln (uch.fam);
        if uch.fam <> '***' then
          begin
```

```
            write('имя ');
            readln(uch.name);
           write('возраст, рост (см), вес(кг) через пробел  ');
           readln(uch.vozr, uch.rost, uch.ves);
           write(f,uch);
        end;
    until uch.fam = '***';
  close(f);
end.
```

Замечание: Обратите внимание, что с клавиатуры надо вводить каждое поле записи по отдельности, а в файл записывается вся запись целиком!

Программа 2

Выводит на экран информацию об участниках группы здоровья в виде таблицы (считываем одну запись целиком и выводим на экран каждое поле). Раздел описания такой же, как в предыдущем примере:

```
Begin
    assign (f,'zdorov.dat');
    reset(f);
    writeln('фамилия':15,'имя ':20,'возраст':10,'вес':10);
    while not (eof(f)) do
     begin
        read(f,uch);
        with uch do
           writeln(fame:20, name:20,vozr:10, rost:10, ves:8);
     end;
     close(f);
 End.
```

Программа 3 – составить самостоятельно

Выводит информацию о тех участниках, которые имеют избыточный вес (норму, определяющую избыточный вес, вводить с клавиатуры).

Контрольные вопросы и задания

Вопросы:

1. Что такое файл?
2. Какие типы файлов применяются в ТР?
3. Основные процедуры и функции для работы с файлами и их формат.
4. Опишите стандартную последовательность вызова процедур при обработке файлов в программах.

5. Назовите общие и отличительные черты типизированного и текстового файла.
6. Для чего используется специальная файловая переменная? Как устанавливается соответствие файловой переменной файлу во внешней памяти?
7. Что общего у процедур Reset и Rewrite и чем они отличаются?
8. Какие отличия существуют в использовании процедуры Reset при открытии различных типов файлов (текстовых, типизированных)?

Задания:

1. Составьте программу, которая создает файл, состоящий из 10 значений типа integer, введенных с клавиатуры. Прочитайте файл и вычислите сумму его элементов.

2. Составьте программу, которая создает файл из элементов типа Char с помощью цикла **while**. Признак выхода из цикла — символ **'#'.**

3. Дан файл, компоненты которого являются действительными числами. Найти наибольшее из значений модулей компонент с нечетными номерами..

4. Дан файл, компоненты которого являются действительными числами. Найти сумму наибольшего и наименьшего из значений компонент.

5. Дан файл, компоненты которого являются действительными числами. Найти порядковый номер минимального числа в файле. Если таких чисел несколько, найти номер первого из них.

6. Дан файл f, компоненты которого являются целыми числами. Найти сумму k1-го и k2-го чисел файла.

7. Дан файл f, компоненты которого являются целыми числами. Найти среднее арифметическое положительных чисел файла.

8. Дан файл f, компоненты которого являются целыми числами. Найти первое число, большее числа B. Если такого числа нет, то сообщить об этом.

9. Дан символьный файл f. Подсчитать число вхождений в файл сочетаний ab.

10. Имеется текстовый файл. Все четные строки этого файла записать во второй файл, а нечетные в третий файл. Порядок следования строк сохраняется.

11. Имеется два текстовых файла с одинаковым количеством строк. Переписать с сохранением порядка следования строки первого файла во второй, а строки второго файла - в первый. Использовать вспомогательный файл.

12. Дан текстовый файл f. Исключить пробелы, стоящие в концах его строк. Результат поместить в файл f1.

13. Дан текстовый файл f. Переписать компоненты файла f в файл g, вставляя в начало каждой строки по одному пробелу. Порядок компонент должен быть сохранен.

14. Дан текстовый файл. Дописать в его конце следующие данные: количество строк, количество символов в каждой строке.

15. Имеется текстовый файл. Удалить из него третью строку. Результат записать в отдельный файл.

Лабораторная работа № 9

Тема: Файловый тип данных

Цель: 1) Получение навыков в использовании переменных файлового типа и выполнении операций над ними.

2) Знакомство с задачами, в которых целесообразно использовать определенные типы файловых переменных.

Содержание отчета:

1) Постановка задачи.
2) Текст программы с комментариями алгоритмической структуры отдельных ее частей и описанием смысла и назначения используемых переменных.
3) Протокол работы программы, содержимое используемых файлов и анализ результатов.

Задание 1 (типизированные файлы).

В зависимости от варианта следует оформлять отдельные программы для создания файла, его обработки и вывода содержимого на экран (в некоторых случаях обосновать целесообразность объединения отдельных программ):

1) Сформировать файл, элементами которого являются 12 первых членов последовательности Фибоначчи (последовательность в которой первые два члена равны 1, а каждый следующий равен сумме двух предыдущих).

2) Дано натуральное число n. Записать в файл g целые числа b1, b2, ... ,bn, определенные так, как указано в задании: bi = i! Операцию возведения в степень не использовать.

3) Дано натуральное число n. Записать в файл g целые числа b1, b2, ... ,bn , определенные так, как указано в задании: $bi = 2^{(i+1)}$ Операцию возведения в степень не использовать.

4) Дано натуральное число n. Записать в файл g целые числа $b_1, b_2, ... ,b_n$, определенные так, как указано в задании**:** $\mathbf{b_i = 2^i + 3^{(i+1)}}$ (Операцию возведения в степень не использовать).

5) Дан файл, компоненты которого являются действительными числами. Найти: наименьшее из значений компонент с четными номерами. Размер файла не известен.

6) Дан файл, компоненты которого являются действительными числами. Найти: наибольшее из значений модулей компонент с нечетными номерами. Размер файла не известен.

7) Дан файл, компоненты которого являются действительными числами. Найти:сумму наибольшего и наименьшего из значений компонент. Размер файла не известен.

8) Имеется файл, элементами которого являются отдельные символы. Удалить из него первую из букв "o" (принять, что буквы "o" в файле имеются). Результат записать в другой файл.

9) Дан символьный файл f. Записать в файл g с сохранением пордка следования те символы файла f, которым в этом файле предшествует буква "a".

10) Дан символьный файл f. Записать в файл g с сохранением порядка следования те символы файла f вслед за которыми в этом файле идет буква "a".

11) Дан символьный файл f. Записать в файл g с сохранением пордка следования те символы файла f, которые заключены между первой о последней буквами "a".

12) В существующем файле, элементами которого являются числа изменить на заданное число первый элемент. Новые значения вводятся с клавиатуры.

Задание 2 (текстовые файлы).

Исходный текстовый файл может быть создан либо с помощью текстового редактора, либо с помощью отдельной программы. Предусмотреть вывод на экран всего содержимого текстового файла (можно с помощью отдельной программы):

1) Дан текстовый файл, содержащий строки. Вывести на экран самые короткие строки.

2) Имеется текстовый файл. Найти длину самой длинной строки.

3) Имеется текстовый файл. Найти номер самой длинной строки. Если таких строк несколько, то напечатать первую из них.

4) Имеется текстовый файл. Напечатать k-ый символ n-ой строки.

5) Имеется текстовый файл, в каждой строке которого первые два символа являются буквами. Получить слово, образованное первыми буквами каждой строки.

6) Имеется текстовый файл, в каждой строке которого первые два символа являются буквами. Получить слово, образованное вторыми буквами каждой строки.

7) Имеется текстовый файл, в каждой строке которого первые два символа являются буквами. Получить последовательность символов, образованную s-ми символами каждой строки.

8) Имеется текстовый файл, содержащий 20 строк. Переписать каждую из его строк в массив в том же порядке.

9) Дан текстовый файл f, содержащий программу на языке Паскаль. Проверить эту программу на несоответствие числа открывающих и закрывающих круглых скобок. Считать, что каждый оператор программы занимает не более одной строки файла f.

10) Даны текстовый файл f, строка s. Получить все строки файла f, содержащие в качестве фрагмента строку s.

11) Дан текстовый файл, содержащий строки. Найти количество строк, начинающихся и заканчивающихся одинаковыми символами.

12) Имеется текстовый файл. Переписать его строки в другой файл. Порядок строк во втором файле должен быть обратным по отношению к порядку строк в заданном файле.

Задание 3 (файлы записей).

В соответствии с задачей своего варианта из лабораторной работы №8 составить три программы: первая должна формировать на диске файл записей с исходными данными, вторая – выводить его содержимое на экран, третья – выводить результаты (вторую и третью программы можно объединить – в зависимости от варианта).

Учебное издание

Волкова Татьяна Ивановна

ПРОГРАММИРОВАНИЕ В СРЕДЕ PASCAL ABC

Учебное пособие

Подписано в печать 16.11.2013 г.
Гарнитура «Times». Печать на ризографе с оригинала.
Формат 60x84 1/16. Усл.-печ.л. 8,18. Уч.-изд.л. 7,62.
Бумага писчая. Тираж 100 экз. Заказ № 67.
Цена договорная.

452452, Республика Башкортостан, г. Бирск, ул. Интернациональная 10.
Бирский филиал Башкирского государственного университета.
Отдел множительной техники БФ БашГУ