МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
СУМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

В.А. Боровик, Н.В. Тыркусова

ПРОГРАММИРОВАНИЕ

ЧАСТЬ 1

УЧЕБНОЕ ПОСОБИЕ

по дисциплине "Программирование"
для студентов механико-математического факультета
всех форм обучения

СУМЫ
ИЗД-ВО СУМГУ
2004

ПРЕДИСЛОВИЕ

Язык программирования Pascal был разработан Н. Виртом в 1972 г. для целей обучения программированию. Однако он получил широкое распространение для решения инженерных и научных задач. Язык завоевал признание, поскольку:

- отражает достаточно полно идеи структурного программирования;
- удобен для разработки программ методом нисходящего проектирования;
- позволяет использовать принципы объектно-ориентированного программирования;
- предоставляет гибкие возможности в отношении используемых структур данных;
- объектно-ориентированные библиотеки, входящие в комплект, позволяют быстро разрабатывать интерфейс пользователя.

Позже появились модификации данного языка, такие, как Borland Pascal for Windows и Object Pascal, позволяющие использовать возможности программирования в системе Windows. В настоящее время большой популярностью пользуется среда визуального программирования Delphi, в основе которой лежит Object Pascal.

Данное учебное пособие отражает опыт обучения программированию на базе языка Pascal, накопленный на кафедре информатики механико-математического факультета Сумского государственного университета. В первой части пособия изложен материал первого семестра по обучению данной дисциплине.

В пособии приведено большое количество примеров, иллюстрирующих теоретический материал, позволяющих самостоятельно разрабатывать программы по той или иной теме.

ТЕМА 1. АЛГОРИТМИЗАЦИЯ И НАЧАЛА ПРОГРАММИРОВАНИЯ

1.1. Основные этапы решения задач с помощью ЭВМ

Решение задач в любой деятельности – это всегда получение определенных результатов. Сам процесс получения результатов опирается на некоторый способ действий и предполагает использование определенных средств. Одним из новейших средств решения различных задач являются ЭВМ (электронные вычислительные машины). Этапы решения задач с использованием ЭВМ представлены на рис.1.1.

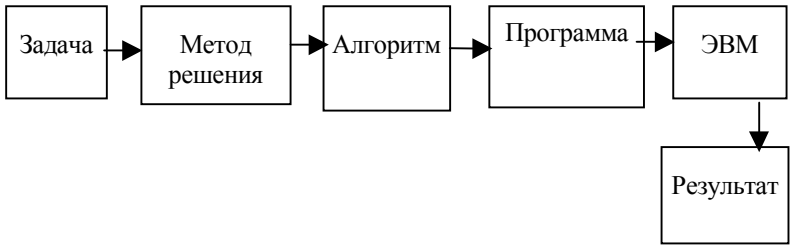


Рисунок.1.1 – Этапы решения задач с помощью ЭВМ

I Постановка задачи и построение ее математической модели.

- ✓ Необходимо четко сформулировать задачу. Определить и перечислить все входные данные и данные, которые необходимо найти, указать, как связаны входные данные и результат.
- ✓ Определить условия, из которых можно получить нужные результаты.
- ✓ Четко и однозначно определить требования к конечным результатам, указать необходимую точность вычислений.

Поскольку ЭВМ – это математическое устройство для обработки данных, постановка задачи должна выражаться в математической форме.

Описание наиболее существенных свойств объектов и явлений, которые исследуются в задаче с помощью математических формул и уравнений, называется ***построением математической модели*** этого объекта.

Точная постановка задачи – это возможность обеспечить одинаковое понимание этой задачи разными людьми. Она не должна допускать двузначного толкования.

Задача	<Содержательное формулирование >
Дано	<Перечень начальных данных >
Нужно	<Перечень данных, которые необходимо получить>
Связь	<Зависимости между нужным и начальным>
При условии	<Условия допустимости начальных данных>

II Выбор метода решения. Выбирают математический метод, который позволяет свести решение к выполнению арифметических и логических действий с оптимальной точностью.

III Разработка алгоритма. Алгоритм – это набор правил, которые указывают, какие действия и в какой последовательности нужно выполнить, чтобы за конечное число шагов получить решение задачи.

Пример.

Задача. За какое время тело, брошенное с высоты ***h*** с начальной скоростью V_0, упадет на землю?

Дано: высота, начальная скорость, ускорение свободного падения **(h, V_0, g)**.

Нужно: определить время падения **(t)**.

Связь: $\mathbf{h=V_0t+gt^2/2.}$

При условиях: $\mathbf{h>0,\ t>0,\ V_0>0.}$

Метод решения: Задача сводится к решению квадратного уравнения относительно **t:**

$$\mathbf{gt^2/2+V_0t-h=0,}$$

$$t_{1,2} = \frac{-V_0 \pm \sqrt{D}}{g}, \text{ где } D=V_0^2 +4g/2*h.$$

Алгоритм:

1 Предоставить значения h, V_0, g.

2 Вычислить дискриминант (D).

3 Проверить D>0.

4 Если D>0, вычислить t_1, t_2.

5 Выбрать время (t), которое будет удовлетворять условию задачи.

6 Выдать сообщение о полученном значении времени падения.

7 Если D<0, выдать соответствующее сообщение.

IV Написание программы. ЭВМ может выполнять алгоритмы без участия человека, автоматически. Но для этого алгоритм нужно записать на языке, “понятном” для машины, то есть на специальном языке, который называется ***языком программирования***. Алгоритм, который записан на языке программирования, называется ***программой***.

V Взаимодействие программы с ЭВМ.

1 ***Материализация программы*** – запись программы с помощью текстового редактора на магнитный носитель.

2 ***Трансляция*** – перевод команд программы с алгоритмического языка в машинные коды.

Трансляторы с языков высокого уровня (C, Basic, Pascal) переводят программы языка высокого уровня на машинный язык, кроме этого, трансляторы осуществляют синтаксический анализ программы, могут также оптимизировать программу, иметь инструменты отладки.

Ассемблеры переводят программы, написанные на машинно-ориентированных языках, которые отображают особенности того или иного процессора.

Интерпретатор- программа, которая пошагово анализирует и сразу же выполняет команды программы.

Компилятор- программа, которая полностью переводит данную программу на машинный язык. В результате компиляции получают промежуточный код программы.

3 На этапе ***компоновки*** (редактирование связей) подключаются необходимые библиотеки подпрограмм, которые содержат ранее написанные процедуры и функции. На этапе редактирования связей осуществляется формирование перекрестных ссылок между независимо составленными программными модулями при объединении их в единый абсолютный модуль. В результате получаем файл с расширением .EXE. Программа готова для запуска на выполнение.

4 ***Тестирование*** – процесс подготовки, выполнения программы и анализ результатов с целью выявления ошибок. Программу выполняют на ЭВМ с разными значениями

аргументов, которые бы обеспечивали проверку всех возможных условий, при которых может возникнуть ошибка.

5 ***Вычисление.*** Проводят необходимые вычисления.

VI Интерпретация результатов:

- анализ результатов на соответствие действительности;
- строят таблицы, графики, анализируют и объясняют полученные результаты.

Необходимо помнить, что ЭВМ не способна сама решать задачи, только выполняет указанную последовательность действий.

Использование ЭВМ не освобождает нас от необходимости осмысливать свою работу, глубоко изучать исследуемую область и разделы математики, которые используются.

1.2. Алгоритмы

Алгоритм – это организованная последовательность действий, допустимых для некоторого исполнителя. Строгого определения алгоритма нет: в информатике это понятие фундаментальное. Понятие “Алгоритм” введено в математику Аль-Херезми в IX столетии.

Свойства алгоритмов

Массовость. Алгоритм должен быть применим для любых элементов из множества входных данных (пригоден для решения задачи при всех допустимых входных данных).

Определенность. Операции не должны иметь двоякого толкования. Порядок выполнения должен быть строго определенным.

Дискретность. Процесс решения алгоритма должен состоять из отдельных завершенных операций, которые выполняются последовательно и за конечное время.

Результативность. За конечное число шагов получаем конкретный результат.

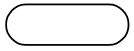
Формальность. Любой исполнитель, способный выполнять указания алгоритма (даже не понимая их содержания), действуя по алгоритму, может выполнить поставленную задачу.

Способы записи алгоритмов

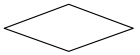
- ✓ Словесно-пошаговая форма. Базируется на обычном языке.
- ✓ Графическая – в виде блок-схемы.
- ✓ На языках программирования.

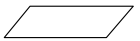
Блок-схема разрешает наглядно отобразить структуру алгоритма. Элементы блок-схемы – геометрические символы, которые обозначают отдельные операции, переходы. Символы связаны стрелками, которые определяют последовательность действий. В середине блоков записываются сами действия (например, вычисления, условия и т.д.)

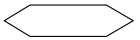
Основные символы для построения блок-схем

1  — Вход в алгоритм (начало), выход из алгоритма (конец).

2  — Описание действия. Процесс: обработка (вычисление) данных; действие – присвоение.

3  — Логический блок передачи управления. Проверка логического условия – выбор направления выполнения алгоритма в зависимости от некоторого условия.

4  — Ввод/вывод информации.

5  — Организация цикла.

6  — Выполнение прежде созданных алгоритмов. Вызов подпрограмм.

7  — Соединитель. Связь между прерванными потоками.

Пример. Стрельба в тире.

Составить блок-схему алгоритма проверки попадания точки M(a,b) в круг с центром в начале координат с радиусом R.

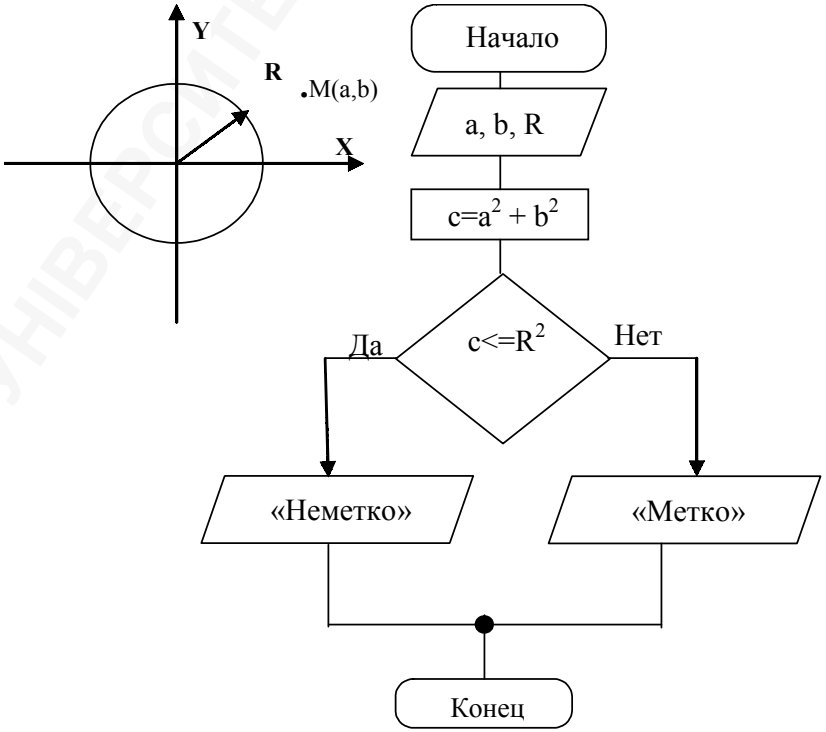
Уравнение окружности: $\mathbf{a^2 + b^2 = R^2}$.

Математическое условие попадания в мишень:

$$\mathbf{a^2 + b^2 <= R^2},$$

т.е. точка лежит внутри круга.

Блок-схема алгоритма



1.3. Методология структурного программирования (базовые структуры алгоритмов)

Структурное программирование – это методология разработки алгоритмов и программ.

Выполняется:

1 анализ задачи;

2 разделение задачи на довольно самостоятельные части;

3 программирование этих частей;

4 компоновка программы из базовых блоков.

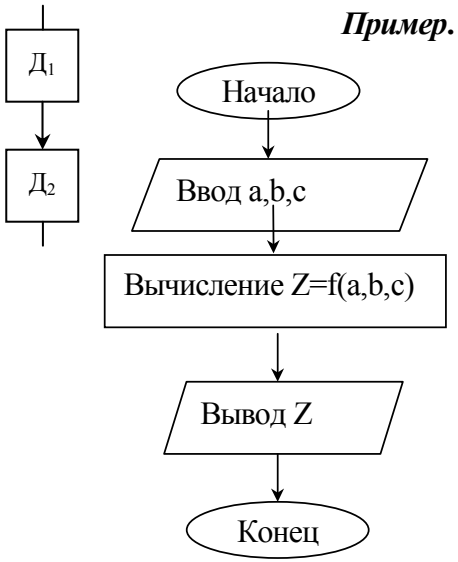
Основные типы вычислительных процессов

1 Линейный.
2 Разветвляющийся.
3 Циклический.

Их комбинация дает разнообразие алгоритмов.

***Теорема о структурировании*: какая бы сложная не была задача, схема алгоритма может быть представлена с использованием ограниченного числа базовых структур.

1 ***Последовательность (линейные вычисления)*** – может изображаться в виде схемы, где выполнение следующего блока идет за предыдущим.



Пример. Вычислить Z=f(a,b,c)

Оглавление программы

ReadLn(a,b,c)

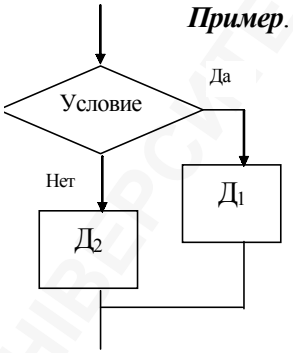
Z:=…

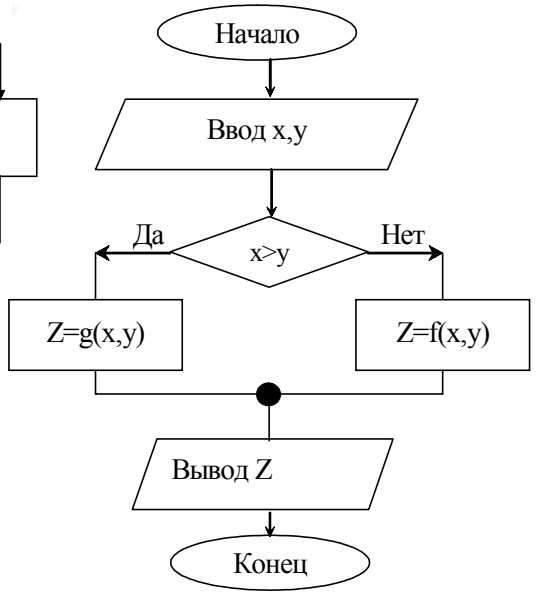
WriteLn(Z)

End.

2 ***Разветвления*** – изображается схемой, где возможно несколько путей выполнения в зависимости от выполнения определенных условий.

Пример. Вычислить функцию $Z=\begin{cases} g(x,y), x>y \\ f(x,y), x\le y \end{cases}$

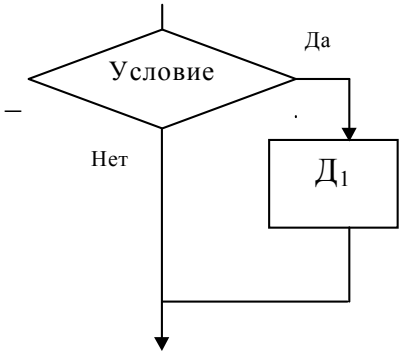




На языке Borland Pascal соответствует оператор

If <условие> **then** <Д1> **else** <Д2>;

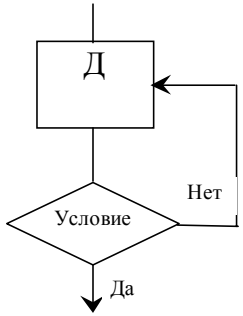
Частный случай разветвления – «обход»



If <условие> **then** <Д1>;

3 ***Циклические структуры*** (многократное выполнение некоторых действий).

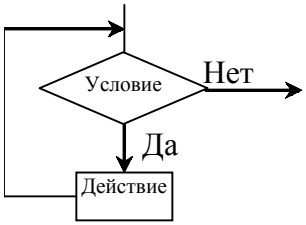
- ✓ ***Цикл с послеусловием*** – вычисления выполняются несколько раз до тех пор, пока выполнится условие выхода. Особенность: выполняется хотя бы один раз



На языке Borland Pascal соответствует оператор

Repeat
<Действие>
Until <Условие> ;

- ✓ ***Цикл с предусловием*** – пока выполняется условие – работает цикл. Если при первом проходе условие не выполняется, то цикл не выполняется ни разу.



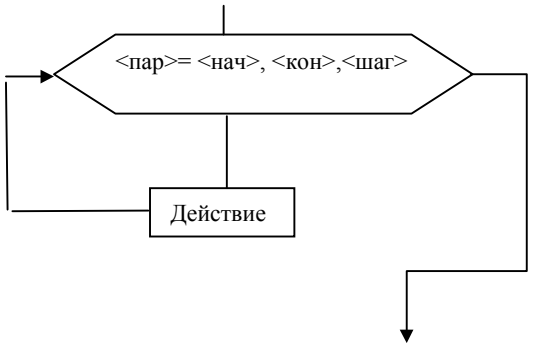
На языке Borland Pascal соответствует оператор

While <Условие> **do** <Действие>;

Если <Условие> имеет значения True, то выполняется <Действие> и повторяется проверка условия, а если False – While прекращает свою работу.

- ✓ ***Цикл с параметром*** (заранее известно число повторений цикла). Алгоритм: сначала вычисляется выражение `<нач>` и происходит присваивание `<пар> := <нач>`. После этого циклически повторяется:

1 проверка условия `<пар>` **<=** `<кон>`, если условие не выполняется, оператор завершает свою работу;
2 выполнение действия;
3 наращивание переменной `<пар>` .



На языке Borland Pascal соответствуют операторы:

for <параметр> **:=** <нач. значение> **to** <кон. значение> **do** <оператор>;

Параметр ***увеличивается*** *на* ***1***.

for < параметр > **:=** < нач. значение > **downto** < кон. значение > **do** < оператор >;

Параметр ***уменьшается*** *на* ***1***.

ТЕМА 2. ЗНАКОМСТВО С ИНТЕГРИРОВАННОЙ СРЕДОЙ РАЗРАБОТЧИКА (IDE) BPW

Компилятор Borland Pascal for Windows – первый компилятор языка Pascal, который полностью работает в Windows–окружении, позволяет создавать программы с использованием технологии объектно-ориентированного программирования.

Для запуска IDE (интегрированной среды разработчика) для Windows необходимо дважды щелкнуть "мышью" по пиктограмме BPW на рабочем столе или выбрать ее с помощью клавиатуры и нажать клавишу Enter.

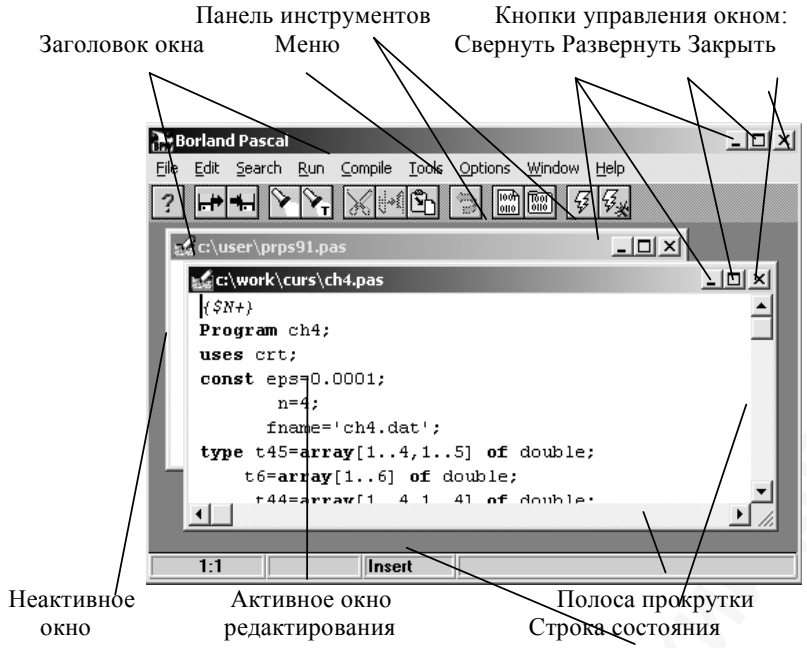


Рисунок 2.1 – Элементы интерфейса

2.1 Интерфейс программы

Внешний вид и принципы работы среды ничем не отличаются от работы любой Windows-программы с интерфейсом MDI (многодокументный интерфейс).

2.2 Пишем первую программу

Опишем все шаги, необходимые для создания первой программы “Hello World!”.

1 Откройте IDE, щелкнув по иконке или запустив файл *BPW.EXE*.

2 В меню *File* выберите пункт *New*. Таким образом вы создадите новое окно для редактирования программы.

3 На панели инструментов нажмите кнопку  или в меню *File* – пункт *Save as*. В верхней строчке появившегося диалогового окна напишите полный путь и имя сохраняемого файла (например: C:\USER\HELLO.PAS). Помните, что все программы нужно сохранять в своем каталоге! Во избежание проблем имя файла должно состоять только из латинских букв и иметь длину не более восьми символов.

4 Наберите текст программы:
Program *hello;*
Uses *WinCRT;*

Begin

Writeln*(’Hello, WORLD!’);*
End.

5 Нажмите комбинацию клавиш Ctrl+F9 для компиляции программы и запуска ее на выполнение. Если вы все сделали правильно, то должно появиться окно, как на рис. 2.2.

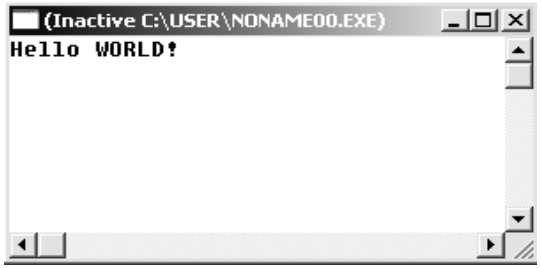


Рисунок 2.2 – Результат работы программы

Вполне возможно, что вы ошиблись в наборе или пропустили один символ. В таком случае после компиляции будет подсвечена первая строка, содержащая ошибку, а в строке состояния красным цветом будет указан код ошибки и ее описание (рис. 2.3).

Рисунок 2.3 – Сообщение об ошибке

Исправьте ошибку и повторите компиляцию (п.5).

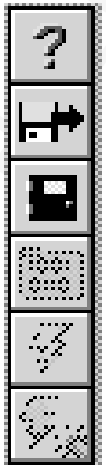
Сохраните результаты своей работы. Для этого нажмите кнопку  на панели инструментов или выберите в меню *File* пункт *Save*.

2.3. Работа в среде
Использование панели задач

С помощью панели задач SpeedBar и "мыши" вы можете быстро выбирать команды и другие действия. Например, если вы щелкните "мышью" на командной кнопке Open a File (Открыть файл), то реакция IDE будет такой же, как при выборе команды Open меню File.

Оперативная полоса контекстно-зависима. То, какие командные кнопки в ней выводятся, зависит от того, какое окно активно - окно рабочей области или окно редактирования.

Панель задач окна рабочей области выводится, когда в IDE не открыты окна редактирования. На панели задач рабочей области выводятся следующие командные кнопки:



Help (Контекстные экраны справочной системы)

Open a File (Открыть файл)

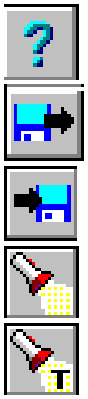
Exit the IDE (Выход из IDE)

Make (Формирование)

Make and Run (Формирование и запуск)

Make and Run under Debugger
(Формирование и запуск с отладчиком)

Следующие командные кнопки выводятся на панели инструментов окна редактирования:



Help (Справка по редактору)

Open a File (Открыть файл)

Save a file (Сохранить файл)

Search for text (Поиск текста)

Search again (Повторный поиск)


Cut to Clipboard (Вырезание в буфер)


Copy to Clipboard (Копирование в буфер)


Paste from Clipboard (Вставка из буфера)


Undo (Отмена)


Compile (Компиляция)


Build (Формирование)


Make and Run (Формирование и запуск)


Make and Run under Debugger (Формирование и запуск с отладчиком)

Иногда определенные командные кнопки панели инструментов выводятся тусклыми. Это означает, что команда, представляемая данной кнопкой, в текущем контексте для вас недоступна. Например, если окно редактирования открыто, и буфер вырезанного изображения пуст, то кнопка Paste Text from Clipboard будет тусклой.

2.4. Использование справочной системы

Справочная система Help дает вам возможность легкого доступа к подробной информации о языке Borland Pascal, интегрированной среде, библиотеке динамической компоновки, ObjectWindows, интерфейсе прикладных программ Windows (API) и дополнительных утилитах, предусмотренных в Borland Pascal. Вы можете просматривать все эти темы в справочном окне Help или получать контекстно-зависимую справочную информацию об IDE или терминах, набираемых вами в окне редактирования.

Справочную систему Borland Pascal for Windows вы можете использовать аналогично справочной системе Windows. Чтобы узнать о работе Help Windows, выберите команду Help|Using Help. Вы узнаете об общих средствах Help Windows (таких, как аннотирование, использование меток текста, просмотр и печать), о которых не упоминается в данном руководстве.

Вызов справки

Получить доступ к справочной информации Help вы можете следующими способами:

- Щелкните "мышью" на команде Help полосы меню или нажмите клавиши Alt+H. IDE выводит меню Help. В этом меню вы можете выбрать экран Contents (Оглавление) системы Help, получить справку по использованию справочной системы Help, вывести информацию по теме, на которой позиционирован курсор в окне редактирования, или вывести такую специфическую для Borland Pascal информацию, как справка по языку, сообщениям об ошибках Borland Pascal, примерах программ и т.д.

- Для вывода экрана оглавления справочной системы Borland Pascal нажмите клавиши Shift+F1.

- Выберите в диалоговом окне командную кнопку Help.

Выводится экран с кратким пояснением по всем командам, доступным в данном диалоговом окне. Если вы щелкните "мышью" на подчеркнутой теме или выберите ее с помощью клавиатуры и нажмете Enter, то увидите более подробную информацию о выбранной команде.

- Поместите курсор на термин в окне редактирования и выберите Topic Search. Используйте любой из следующих методов:

- нажмите клавиши Ctrl+F1;

- выберите команду Help|Topic Search;

- выберите команду Topic Search в локальном меню окна редактирования.

Выводится справочный экран с информацией о ключевом слове, на котором находится курсор в активном окне редактирования.

- В справочном окне Help выберите командную кнопку Search (Поиск).

Выводится диалоговое окно поиска Search. В его верхнем блоке списка вы можете прокручивать каждую тему в справочной системе Borland Pascal. Если вы знаете, какую тему вы ищете, начните набирать эту тему в блоке ввода и вы увидите данную тему в блоке списка. Выделите тему и выберите Show Topic.

Примечание. Если вы знаете, что нужно найти, то командная кнопка Search дает вам скорейший способ для вывода нужного экрана Help.

Если ваша тема имеет более детальное разбиение, вы увидите в нижнем блоке списка другие темы. Выделите нужные темы и выберите Go To. Выводится справочный экран по нужной теме.

Копирование примеров кода

Справочная система Help содержит пример кода для каждой процедуры и функции. Вы можете скопировать эти примеры из справочной системы в свое окно редактирования. Чтобы скопировать пример, сделайте следующее:

1 Выведите экран Help по нужной процедуре или функции.

Вы увидите имя и пример исходного кода в нижней части окна Help.

2 Для вывода примера щелкните кнопкой "мыши" на имени примера кода.

3 Выберите команду Edit|Copy.

Выводится диалоговое окно с примером кода. Вы можете выделить часть кода для копирования в буфер вырезанного изображения. Если вы этого не сделаете, пример будет скопирован целиком.

4 Выберите команду Copy (Копирование).

5 Вернитесь в окно редактирования и выберите команду Edit|Paste, либо нажмите клавиши Shift+Ins или щелкните "мышью" на командной кнопке Paste. Справочная система Help дает вам возможность легкого доступа к необходимой информации.

2.5. Работа в редакторе

Поскольку редактор IDE ведет себя аналогично всем другим редакторам Windows, вероятно вы уже знаете, как редактировать текст. В редакторе соблюдается стандарт общего доступа пользователя CUA (Common User Access), который используется в большинстве программ Windows. Те же команды редактирования, которые вы использовали в других приложениях Windows, работают также и в редакторе IDE.

2.6. Операции с файлами

При программировании в IDE вы можете создавать новые файлы, открывать существующие файлы и сохранять их. Основные команды работы с файлами перечислены в таблице 1.

Таблица 1

Основные команды работы с файлами

Команда	Описание
File|New	Открывает новое окно редактирования и присваивает ему временное имя
File|Open	Выводит диалоговое окно, с помощью которого можно открыть файл
File|Save	Сохраняет файл в активном окне редактора на диске
File|Save As	Сохраняет файл в активном окне редактора под другим именем
File|Save All	Сохраняет все модифицированные файлы
File|Print	Выводит содержимое активного окна редактирования на принтер

Помните, что все программы необходимо сохранять в своем каталоге! (обычно это c:\user\ваша_группа или c:\user\ваше_имя). Во избежание проблем имя файла должно состоять только из латинских букв и иметь длину не более восьми символов.

ТЕМА 3. ЯЗЫК BORLAND PASCAL FOR WINDOWS

3.1. Алфавит языка

- Буквы латинского алфавита (a..z, A..Z, _).
- Арабские цифры (0..9).
- Символы (+ - * / = , ' . : ; < > [ ] ( ) { } @ $ # и их пары <> <= >= := (* *) (. .)).

Русские и украинские буквы можно использовать только в строковых переменных и комментариях.

- Комбинация символов образует ***составные символы***:
  := — присваивание;
  <> — не равняется;
  <= — меньше или равняется;
  >= — больше или равняется;
  . . — диапазон значений;
  (. .) — [];

3.2. Слова и разделители

Программа на Borland Pascal состоит из слов и разделителей.

***Разделители*:** пробел, точка с запятой (;), фигурные скобки({ }).

Для удобства программирования на языке Borland Pascal предусмотрена вставка комментариев. В роли комментариев могут выступать любые символы, взятые в фигурные скобки – {…} или в скобки типа „скобка-звёздочка" - (*…*). При обработке компилятор не воспринимает символы, следующие за фигурными скобками. Исключением являются директивы компилятора.

Слова делятся на:

- ***зарезервированные*** (ключевые) слова (**Program, Begin, Var, End,** …). Они имеют фиксированное название и раз и навсегда определенное содержание. *Идентификаторы пользователя не могут совпадать с зарезервированным словом;*
- ***стандартные***_слова — служат для обозначения заранее определенных типов данных, констант, процедур и функций: **sin, abs, cos, ln,** … Стандартный идентификатор (в отличие от зарезервированного слова) может переопределяться, но это часто приводит к ошибкам;
- ***идентификаторы*** пользователя — служат для обозначения меток, констант, переменных, процедур, функций. Они должны быть понятны. (Например. Data лучше, чем D).

Общие правила написания идентификаторов:

- ✓ идентификатор должен начинаться с буквы или знака подчеркивания (_) (исключение: метки могут начинаться и с цифры);
- ✓ идентификатор может состоять из букв, цифр (пробелы, точки и др. специальные символы в них недопустимы);
- ✓ между двумя идентификаторами должен быть хотя бы один пробел (неверно: **BeginWriteln**). Заглавные и прописные буквы в программе трактуются одинаково.

Пример. Правильные идентификаторы

```
Metka12
Block_56
MyProgramIsBestProgram
_beta
```

Пример. Неправильные идентификаторы

```
Number.Data
{содержит специальный символ – точку}
1Graph
{начинается  с цифры}
Omd 1
{содержит пробел}
Block*
{содержит *}
mod
{зарезервированное слово}
```

3.3. Общая структура программы на языке Pascal

Правила записи программы:

Программа начинается с заглавия, которое задает имя программы.

Program <Name>;

Программа на языке Borland Pascal состоит из 2 частей:

I Описание используемых данных, процедур, функций.

Блок состоит из 5 разделов, расположенных в таком порядке:

1 Раздел описания меток (**Label**).
2 Раздел описания констант (**Const**).
3 Раздел описания типов (**Type**).
4 Раздел описания переменных (**Var**).
5 Раздел описания процедур и функций (**Procedure, Function**).

II Указание действий (операторов), которые должны быть выполнены над данными. Блок выполняемых операторов расположен между **begin …. End.**

Каждая конструкция на Borland Pascal должна заканчиваться знаком „точка с запятой”(;). В конце программы обязательно ставится точка.

Любой раздел может быть пустым. Разделы 1-4 - разделы описаний, в них определяются данные, обрабатываемые в программе. Раздел 5 и блок выполняемых операторов содержат алгоритмы обработки данных.

```
Program <имя>;
{раздел описаний}
begin {начало самой программы}

{раздел операторов – тело программы}

end.      {конец программы}
```

Пример.

Program My_First_Program; *{название программы не обязательно}*
{точка с запятой – отмечает завершение оператора или описания}

uses wincrt; {подключение стандартной библиотеки}

var text: string; *{описание переменных}*
{название и тип}
begin
Text:= 'Я программирую на Borland Pascal';
{переменной типа строка присваивается строка символов}
Writeln(Text);
{выводит сообщение на экран компьютера}
end.

3.4. Константы, переменные и выражения

Программа обрабатывает данные: константы, переменные.

3.4.1. Типы данных

Рассмотрим простые типы данных

Целые типы		
Название типа	Границы	Размер, байт
Byte	0..+255	1
Shortint	-128..+127	1
Word	0..+65535	2
Integer	-32768..+32767	2
Longint	2147438648..+2147438647	4

Действительные типы			
Название типа	Количество значащих цифр	Диапазон десятичного порядка	Размер, байт
Real	11..12	-39..+38	6
Double	15..16	-324..+308	8
Extended	19..20	-4951..+4932	10
Comp	19..20	$2*10^{63}$..+$2*10^{63}$-1	8

Символьные – char (1 байт).

Логические – boolean (true/false) (1 байт).

3.4.2. Константы и переменные

Константы – элемент данных, значения которых известны заранее и в процессе выполнения программы не изменяются. Запись в программе:

const <имя> = <значение>;

Пример.

```
Const
    f=5; {целое число 5}
    A=3.5; {действительное число 3.5}
    z='z'; {символ z}
    P=3.14E5; {3.14 помножить на 10 в степени 5}
    str='Строка символов';
```

Переменные в отличие от констант могут изменять свои значения в процессе выполнения программы. Каждая переменная и константа принадлежат к определенному типу данных. Тип констант автоматически определяется компилятором без предшествующего описания.

Запись в программе:

Var < имя > : тип;

Пример.

```
var: f2, w: integer; {переменные целого типа}
   x1, x2: real;   {переменные действительного типа}
```

В Borland Pascal необходимо **обязательно** описывать **все** объекты, которые используются в процессе выполнения программы.

3.4.3. Выражения

Выражение – определяет порядок выполнения действий над элементами данных и состоит из:

1 операндов (константы, переменные, обращение к функциям);
2 круглых скобок;
3 знаков операций.

Записывается линейно, в строку.

Действия с целыми числами:

+ - * (сложение, вычитание, умножение)
div – целая часть от деления (5 div 2 = 2)
mod – остаток от деления (5 mod 2 = 1)

Действия с действительными числами:

+ - * / (сложение, вычитание, умножение, деление)

Пример.

(a+b)*c
(a+b)/(c-d)
sin(t)+a*t

Встроенные (стандартные) функции			
Обозначение	Тип параметра	Тип результата	Объяснение
Abs(x)	Д, Ц	тип аргумента	Модуль аргумента $\|x\|$
Arctan(x)	Д	Д	Арктангенс (в радианах), *arctgX*
Cos(x)	Д	Д	Косинус (угол в радианах), *cosX*
Dec(x)	Ц	Ц	Уменьшает значение x на i (или на 1, если i отсутствует)
Exp(x)	Д	Д	Экспонента, e^x (например x^y=exp (y * ln (x))
frac(x)	Д	Д	дробная часть числа
Inc(x,[i])	Д	Д	Увеличивает значение x на i (или на 1, если i отсутствует)
Int(x)	Д, Ц	Ц	Целая часть числа
ln(x)	Д, Ц	Д	Логарифм натуральный, *lnX*
Odd(i)	Д	Л	True, если i нечетное, False, если i чётное
Pi	-	Д	π=3.1415926...
Random	-	Д	случайное число, равномерно распределенное в диапазоне [0;1]
Random(x)	Ц	Ц	Случайное целое число, равномерно распределенное в диапазоне 0..X
Randomize	-	-	Инициация генератора случайных чисел
Round(x)	Д, Ц	Ц	Округление к целому
Sin(x)	Д	Д	Синус (угол в радианах), *sinX*
Sqr(x)	Д	Д	Квадрат аргумента, X^2
sqrt(x)	Д	Д	Квадратный корень аргумента, $\sqrt{x}$
Trunc(x)	Д, Ц	Ц	Отсекание дробной части

Д – действительные, Ц – целые, Л – логические.

Пример. Записать выражения:

- x^y exp(y*ln(x));
- tg(x) sin(x)/cos(x);
- $\ln(\sqrt{\ell^{x-y}} + x^{|y|} + \sqrt[7]{z})$ ln(sqrt(exp(x-y))
  +exp(abs(y)* ln(x))+exp(1/7*ln(z)));
- $\frac{\sqrt[5]{|\sin^3 \beta x| + c} + \ell^{arctgx}}{\sqrt{|cx + tgx|}}$ (exp(1/5*ln(abs(exp(3*
  sin(b*x)))
  +c))+exp(arctan(x)))/sqrt(abs(c*x+sin(x)/cos(x)));

Логическое выражение

В логических выражениях допустимы следующие операции:

- **Not** - отрицание - **not** *L*
- **And** – и - *L1* **and** *L2 – логическое умножение*
- **Or** – или - *L1* **or** *L2 – логическое сложение*
- Операции отношения:

< , >, =, <=, >=, <>

< > = ≤ ≥ ≠

Таблица истинности логических выражений (обозначение " + " – true, " - " – false):

A	B	A **and** B	A **or** B	**not** A
+	+	+	+	–
+	–	–	+	–
–	+	–	+	+
–	–	–	–	+

Приоритет без скобок:

1) NOT, @ - унарные операции;
2) * / div mod and – бинарные операции типа умножения;
3) + - or, xor - бинарные операции типа сложения;
4) < >, <=, >, >=, <, = , in - бинарные операции типа отношения.

Пример. Записать логические выражения:

- *-3 ≤ X ≤ 5* (x>=-3)**and** (x<=5)
- *X>0 и Y>25* (x>0) **and** (y>25)
- *A>X или B<Y* (a>x) **or** (b<y)

Пример. Определить значение логического выражения.

Значение переменных	Выражение	Результат
X=1	X>3	False
A=true, C=false	A **or** C	True
A=true, C=false	A<>C	True
X=1, Y=3	Sqr(x)+sqr(y)<4	False

ТЕМА 4. СТРУКТУРЫ УПРАВЛЕНИЯ ПЕРЕМЕННЫМИ В BORLAND PASCAL

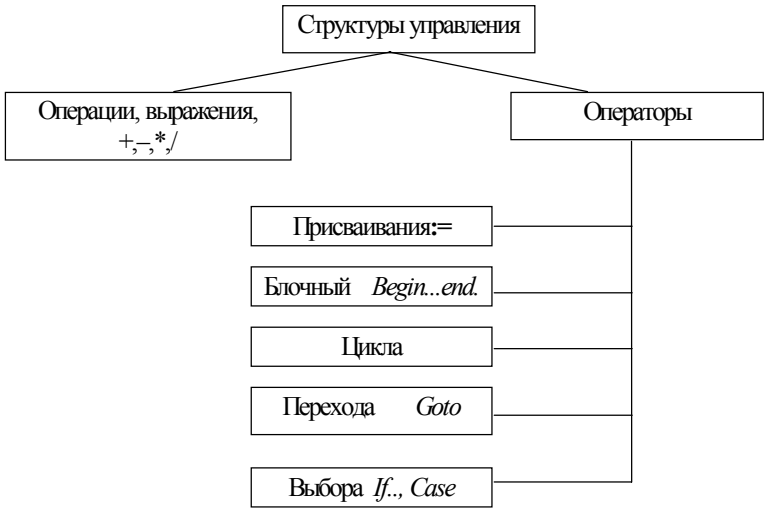


Рисунок 4.1– Структуры управления в Borland Pascal

Для записи простейших программ нужны:

- ✓ процедуры ввода/вывода;
- ✓ оператор присваивания;
- ✓ блочный оператор;
- ✓ операторы выбора;
- ✓ циклические операторы.

4.1. Процедуры ввода/вывода

Для ввода информации используют встроенные процедуры **read** и **readln**.

Read(<список переменных>);

Readln(<список переменных>);

Процедуры read и readln "считывают" значения переменных с клавиатуры и присваивают их тем переменным, которые записаны в них. Основное отличие обнаруживается в момент окончания работы оператора - местоположение курсора: **read** - курсор в очередной позиции текущей строки, а **readln** - курсор сначала очередной строки.

Пример.

Readln(a,b);

При выполнении программы, встретив оператор readln, компьютер приостановит работу в ожидании ввода информации. После того как мы введем с клавиатуры два значения через пробел, например 16 5 и нажмем клавишу enter, компьютер присвоит переменной *a* значение 16, т.е. отправит его в ячейку памяти с именем *a*, а переменной *b* – значение 5, т.е. отправит его в ячейку памяти с именем *b*. Этот процесс называют "считыванием" значения в переменную. Затем будет выполняться следующий оператор программы.

Для вывода информации используют встроенные процедуры **write и writeln**.

Процедура **write** выводит информацию в одну строку, курсор в очередной позиции текущей строки, а **writeln** после вывода информации осуществляет переход к новой строке**.**

Обе эти процедуры выводят информацию на экран, если эта информация содержится в виде значений переменных, тогда достаточно в скобках в операторах write или writeln записать имена этих переменных, например,

write(a); **writeln**(f);

Если таких переменных несколько, то их записывают через запятую, например:

write(a, b, c, d); **writeln**(e,f, g, h);

Если информацией являются слова, предложения, части слов или символы, тогда она заключается между знаками " ' " - *апостроф*, например,

write('Введите длину пути');

writeln('Значение скорости равно');

Возможен одновременный вывод и символьной информации и значений переменных, тогда они разделяются запятыми, например,

write('Значение температуры равно ', t);

writeln('Скорость равна ', v, ' при времени движения ', t);

Заметьте, в конце слов, перед апострофом оставлен пробел. *Для чего это сделано*? Конечно, чтобы следующая числовая информация была разделена со словами пробелом.

4.2. Оператор присваивания

<имя переменной >:= <выражение >;

На языке Pascal команда присваивания обозначается **:=** (двоеточие и знак равно). Команда присваивания вычисляет значение выражения, которое находится справа от оператора, "стирает" предыдущее значение переменной и "придает" ей новое значение. Типы переменных по левую и правую стороны должны совпадать.

Пример. Вычислить значение функции

$$y=x^3+5|a-\sin x|+e^{ax}$$

```
uses wincrt;
 var   x, y,a: real; {описываем данные}
begin
   writeln(‘введи значения a, x ’);
   Readln(a,x); {ввод значений переменных a и x}
   y:= sqr(x)*x+5*abs(a-sin(x))+exp(a*x); {вычисляем
   значение y}
   writeln(‘y=’,y); {вывод на экран значения y в экс-
   поненциальной форме, например,
        y = -3.1084952830E+01 }
   writeln(‘y=’,y:6:2); {вывод на экран форматиро-
   ванного значения y, например,
        y = -31.08 }
 end.
```

Если вы выполните программу, то столкнетесь с неприятным явлением - результат, т.е. числовое значение функции, будет выдаваться на экран в экспоненциальной форме, например,

y = -3.1084952830E+01

Во-первых, такой результат неудобно читать и запоминать, во-вторых, нам не всегда нужна высокая точность вычисления и, в-третьих, даже целые значения будут выдаваться тоже в экспоненциальной форме, что уже совсем неудобно.

Чтобы избежать всех этих неприятностей, можно использовать форматированный вывод информации. В общем случае формат имеет следующий вид:

r:f1:f2

Здесь r - имя переменной, значение которой выводится (в данном случае - y), формат f1 указывает, сколько позиций нужно для всего числа, включая знак числа, целую часть, точку и дробную часть числа; f2 - число позиций дробной части числа (после точки).

Надо вообще заметить, что вещественное число в формате с фиксированной точкой печатается так: несколько пробелов; знак минус (-) или пробел; последовательность цифр (целая часть); точка; последовательность цифр (дробная часть).

В Pascal **не допускается** смешивать типы данных.

Переменным некоторых типов можно присваивать значения других типов. Такие типы являются подмножествами тех типов переменных, которым присваивается значение. Перечислим некоторые:

- ✓ переменным типа **STRING** можно присваивать значения типа **CHAR**;
- ✓ переменным типа **REAL** можно присваивать значения типа **INTEGER**;
- ✓ переменным типа **BYTE** можно присваивать значения типа **INTEGER** в допустимом диапазоне для типа **BYTE** и наоборот;
- ✓ переменным типа **WORD** можно присваивать значения типа **BYTE** и т.д.

Пример.

```
uses wincrt;
var  i, j: integer;
           x, y: real;
           a, b: boolean;
           c:    char;
           s:    string;
begin
……
x:=i*y;
i:=j/2; {Неверно: целому типу присваивается действительный}
j:=i*y; {Неверно: несоответствие типов}
i:=j div 2;
x:=y mod 2; {Неверно: mod лишь для целых типов}
j:=i mod j;
x:=y and j; {Неверно: x – действительный тип, ему нельзя присвоить логический}
c:=c+s; {Неверно: символу нельзя присвоить строку}
s:=s+c;
a:=not(a and b);
b:=((x-y)<2)=a;
end.
```

4.3. Блочный оператор

Блочный оператор – составной оператор. Составной оператор – это последовательность произвольных указаний программы, отделенных точкой с запятой, взятых в операторные скобки **Begin..End.** Воспринимается как *одна* инструкция. Причем “;” перед **End** не обязательна.

```
Begin
…….
Begin
 …{группа операторов}
End;
…….
End.
```

4.4. Операторы выбора

4.4.1. Условный оператор

If <логическое выражение > **then** <оператор1>
else <оператор2 >;

или

If <логическое выражение > **then** <оператор>;

Условие – это выражение логического типа, может быть простым или составным.

Пример.

{Простые условия}
a<5 {a<5}
x<=t {x≤t}

{Составное условие}
(a>5) and (a<10) {5<a<10}
(a=2) or (b=5) {a=2 или b=5}
a>-5 or a>5 {неверно}

Алгоритм: сначала вычисляется <логическое выражение >. Если результат TRUE, то выполняется <оператор1>, а <оператор2> пропускается; если результат FALSE, наоборот, <оператор1> пропускается, а выполняется <оператор2>.

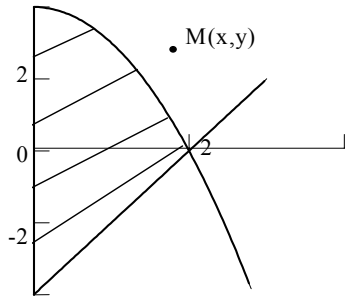
Если необходимо выполнить более одного оператора после **then** или **else,** нужно использовать операторные скобки **begin… End.**

Следует также помнить, что перед **else** точка с запятой (;) *не ставится***.**

Так как не каждый из условных операторов имеет часть **else** (случай обхода), то нужно пользоваться правилом: часть, которая встретилась перед **else,** отвечает ближайшей сверху части **then**.

Пример.

Задается область на плоскости и т. М (x, y). Проанализировать, принадлежит ли точка заданной области, и выдать соответствующее сообщение**.** Вычислить расстояние данной точки от начала координат.



Область ограничена параболой $y=2-x^2$, прямой y=x-2 и осью OX.

```
Program Tochka;
uses wincrt;
var x, y: real;
{переменные: x, y – координаты точки}
   r: real;
{расстояние ее от начала координат}

begin          {начало раздела операторов}
writeln('введите координаты искомой точки');
{вывод сообщения на экран}
readln(x,y);                              {ввод  x, y}
  if (x>0) and  (y>x-2) and (y<2-x*x) then
{проверка условия попадания точки в заданную область}
    writeln('входит')
{сообщение о принадлежности точки области}
  else writeln('не входит');
{сообщение о непринадлежности точки области}
r:=sqrt(x*x+y*y);
{вычисление расстояния от начала координат}
writeln('r= ', r:4:1);      {вывод результата}
  end.                      {завершение программы}
```

Пример. Вычислить значение функции

$$Y = \ln x + \frac{1}{\sin x}.$$

```
uses wincrt;
var x, y: real;
begin
      writeln(‘введите значения x=’);
                              {вывод сообщения на экран}
      readln(x);      {ввод значения  x}
         {проверка условия существования функции}
      if (x>0) and (sin(x)<>0) then
            begin
             y:=ln(x)+1/sin(x);
                        {вычисление значения функции}
             writeln(‘y=’,y:4:1);  {вывод значения y}
            end
         else
              writeln(‘функция не существует’);
            {сообщение о несуществовании  функции}
end.                  {завершение программы}
```

4.4.2. Оператор выбора Case

Оператор выбора Case – разрешает выбирать одно из нескольких возможных продолжений программы.

Case <ключ > ***of***
<список выбора >
[***else*** < оператор >]
***end*;**

<ключ > - выражение порядкового типа: integer, boolean, char (кроме real и string).

<список выбора > - одна или несколько конструкций вида < константа >:< оператор >;.

Алгоритм: вычисляется значение выражения <ключ>, потом в последовательности операторов <список выбора > отыскивается такой, которому предшествует константа, равная значению ключа. Этот оператор выполняется, после чего оператор выбора завершает работу. Если в списке нет такого значения, то выполняется оператор после **else** (если он существует) или оператор сразу завершает работу.

Пример. Программа вводит 2 числа и знак +, -, *, / и выводит на экран результат соответствующего арифметического действия.

```
uses wincrt;
var operation: char;
   x, y, result: real;
begin
 write('введите ненулевые числа x,y=');
 readln (x,y);
 write(‘операция: ’);
 readln(operation);
 case operation of
‘+’: result:=x+y;
‘-’: result:=x-y;
‘*’: result:=x*y;
‘/’: result:=x/y;
end;
writeln(‘результат= ’, result);
end.
```

Пример. По номеру месяца вывести название времени года.

```
uses wincrt;
var  month: byte;
begin
writeln('введите номер месяца 1-12');
read(month);
case month of
12, 1, 2: writeln('зима');
3..5:     writeln('весна');
6..8:     writeln('лето');
9..11:    writeln('осень')
else
          writeln('вы ввели неверный месяц');
end;
end.
```

4.5. Оператор перехода

Оператор перехода **GoTo** – передает управление соответствующему отмеченному оператору.

Goto <метка>;

Метка – это произвольный идентификатор, который разрешает именовать некоторый оператор программы и таким образом ссылаться на него. Перед появлением в программе метка должна быть описана:

```
Label l1, l2, l3;
begin  ...
goto l2;  ...
l1: ...
...
l2: l3: ...
goto l3;  ...
...
end.
```

4.6. Операторы цикла

Цикл в программировании - это многократно выполняемая группа команд, часть программы.

4.6.1. Цикл с заданным числом повторений (оператор For)

for <параметр> **:=** <нач. значение> **to** <кон. значение> **do** <оператор>;

`< параметр >` - параметр цикла – переменная порядкового типа;

`< нач. значение >` - начальное значение – выражение того же типа;

`< кон. значение >` - конечное значение – выражение того же типа;

`< оператор >` - произвольный оператор Borland Pascal. Если операция содержит не один оператор, а несколько, то их объединяют с помощью ОПЕРАТОРНЫХ СКОБОК **begin** и **end**.

Алгоритм: сначала вычисляется выражение `< нач. значение >` и происходит присваивание `<параметр > := < нач. значение >`. После этого циклически повторяется:

a) проверка условия `<параметр > <= < кон. значение >`, если условие не выполняется, оператор завершает свою работу;
b) выполняется <оператор>;
c) наращивание переменной `< параметр >` на **1**.

Существует другая форма записи оператора:

for < параметр > **:=** < нач. значение > **downto** < кон. значение > **do** < оператор >

В таком случае шаг наращивания переменной < параметр> равняется **-1**.

Пример. Вычисление факториала (N!= 1·2·3·..·N).

```
uses wincrt;
var   fact: real;
    n,i: integer;

 begin
          fact:=1;
         write('введите число n=');
         readln (n);
            for i:=1 to n do
           fact:=fact*i;
         writeln('результат= ', fact:2:1);

end.
```

4.6.2. Цикл с предусловием (оператор WHILE…DO)

While <условие> **do** <оператор>

<условие> - выражение логического типа;
<оператор> - произвольный оператор Pascal.

Если операция содержит не один оператор, а несколько, то их объединяют с помощью ОПЕРАТОРНЫХ СКОБОК **begin** и **end**.

Алгоритм: если выражение <условие> имеет значение TRUE, то выполняется <оператор>, после чего проверка выражения <условие> повторяется. Если <условие> имеет значение FALSE, оператор прекращает свою работу.

Пример. Вычислить значения функции $y=f(x)=\left(x^2+1\right)\sqrt{x^8+9}$, если x изменяется в диапазоне от $x_{нач}$=a до $x_{кон}$=b с шагом Δx=h).

```
uses wincrt;
var  x, y, a, b, h: real;

 begin
      write('Введите значения a, b, h=');
      readln(a, b, h);
      x:=a;
         while x<=b do
         begin
            y:=(sqr(x)+1)*sqrt(exp(8*ln(x))+9);
            writeln('При x=', x:2:1, '  y=', y:2:1);
             x:=x+h;
         end
end.
```

4.6.3. Цикл с послеусловием (оператор REPEAT…UNTIL)

```
Repeat
< тело цикла >
until < условие >;
```

< тело цикла > - произвольная последовательность операторов;
< условие > - выражение логического типа.

Алгоритм: операторы < `тело цикла` > выполняются хотя бы 1 раз, после чего вычисляется выражение < `условие >:` если его значение FALSE, операторы < `тело цикла` > повторяются, в противном случае оператор **`Repeat ... until`** завершает работу.

Пример. Ввод алфавитно-цифрового символа и распечатка его кода. Условие прекращения работы – нажатие клавиши ENTER.

```
uses wincrt;
Var c:char;
begin
  repeat
          writeln(‘Нажмите клавишу’);
           c:=readkey;
           write(‘Код ’, ord(c));
            writeln;
 until c=chr(13);
end.
```

Цикл будет работать до нажатия клавиши ENTER.

4.7. Характерные приемы программирования

Пример. Увеличить меньшее из двух целых чисел вдвое.

```
uses WinCrt;
  var
    a, b, c : integer;
  begin
    write('Введите первое целое число '); readln(a);
    write('Введите второе целое число '); readln(b);
    if a < b  then c := 2*a
              else  c := 2*b;
    writeln('Меньшее число ', c div 2);
    writeln('Оно же увеличенное вдвое ', c)
  end.
```

Пример. Составить программу решения квадратного уравнения

$$ax^2 + bx + c = 0.$$

```
uses WinCrt;
  var
    a, b, c, d, x1, x2 : real;

  begin
    write('Введите коэффициенты уравнения ');
    readln(a, b, c);
     d := b*b - 4*a*c; {вычисление дискриминанта}
     if d < 0 then
                writeln('Уравнение не имеет корней')
            else
               if d=0 then
                                  begin
                    x1 := (-b - sqrt(d))/(2*a);
                    x2:=x1;
                     writeln('Уравнение имеет 2
                    одинаковых корня x1=x2=
                    ',x1)
                    end
                else
                    begin
                      x1 := (-b - sqrt(d))/(2*a);
                      x2 := (-b + sqrt(d))/(2*a);
```

```
    write('Уравнение имеет два различных корня ');
    writeln('x1 = ',  x1, '  x2 = ', x2);
    end;
end.
```

Пример. Вычисление суммы и произведения.

$$S = \sum_{n=1}^{10} \frac{1}{n^2},\ P = \prod_{i=2}^{15} \frac{(i-1)}{(i^2+1)}$$

```
uses wincrt;
var   s,p: real;
     n,i: integer;
begin
{Вычисление суммы}
s:=0;{Зануляем значение переменной, которая соответствует сумме}
for n:=1 to 10 do
              s:=s+1/(n*n); {вычисление суммы}
writeln(s:2:1); {печать результата}
{Вычисление произведения}
 p:=1; {Переменной, которая соответствует произведению, присваиваем 1}
for i:=2 to 15 do
               p:=p*(i-1)/(i*i+1); {вычисление произведения}
writeln('Произведение=',p); {печать результата}

 end.
```

Пример. Итерационные циклы: вычислить приближенно e^x, членами ряда <0,001 пренебречь.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!} + ...$$

Рекурсивная формула $U_n = U_{n-1} * \frac{x}{n}$

```
uses wincrt;
var u, s,x: real;
   n: integer;
begin
   u:=1; s:=0; n:=1; {Задаем начальные значения}
write('x='); readln(x);    {вводим значение x}
  while abs(u)>=0.001 do {проверка условия: член
        ряда больше заданной точности}
    begin
         s:=s+u; {вычисление суммы}
         u:=u*x/n;{вычисление следующего члена
              ряда}
         n:=n+1; {увеличиваем счетчик на 1}
    end;
writeln('S=',s:3:3); {печать результата}
write('exp(x)=',exp(x):3:3); {печать точного значения}
end.
```

Пример. Определить, является введенный символ буквой или цифрой.

```
uses wincrt;
    var k:char;
      t: boolean;
    begin
    t:=false;
    repeat
          readln(k);
```

```
            case k of
                    'a'..'z',’A’..’Z’:  writeln ('буква');
                    '0'..'9':  writeln ('цифра');
            else t:=true;
            end;
    until t;
  end.
```

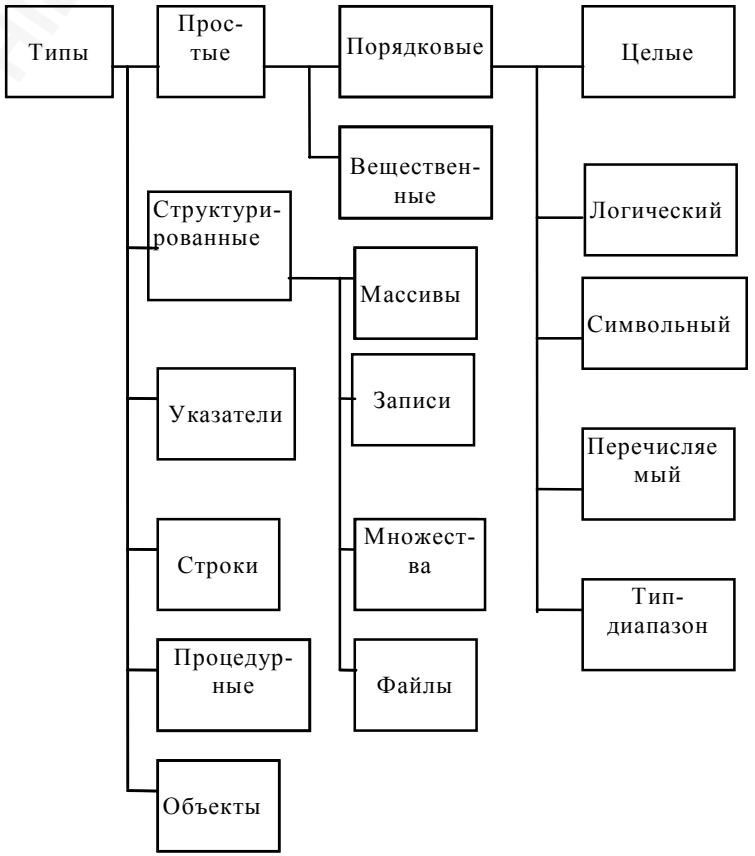
Пример. Вложенные циклы.

Найти z= f (x,y) для xmin ≤ x ≤ xmax с шагом Δx
ymin ≤ y ≤ ymax с шагом Δy

```
uses WinCrt;
Var x,y,z,xmin,xmax,ymin,ymax, dy,dx:real;

begin
writeln(‘введи Хнач Хкон dX');
readln(xmin,xmax,dx);
writeln(' введи Yнач Yкон dY ');
readln(ymin,ymax,dy);
x:=xmin;
        while x<=xmax do
          begin
            y:=ymin;
            while y<=ymax do
              begin
              z:=< функция f (x,y)>;
              writeln (‘x=’,x:3:3,’  y=’,y:3:3,’  z=’,z:3:3);
              y:=y+dy;
              end;
            x:=x+dx
        end;      end.
```

ТЕМА 5. СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ. СТРОКИ

Рассмотренные выше простые типы данных позволяют использовать в программах одиночные объекты – числа, символы и т.п. Однако, Паскаль характеризуется разветвленной структурой типов данных.
Рисунок 5.1 – Типы данных в Pascal



В Паскале могут использоваться также объекты, содержащие множество однотипных элементов. Для описания таких данных используют ***структурированные типы***. К структурированным типам относятся:
тип-массив (**array**);
тип-множество(**set**);
тип запись (**record**);
файловый тип (**file**).

5.1. Массивы

Массивы - формальное объединение нескольких однотипных объектов (чисел, символов, строк и т.п.), рассматриваемое как единое целое. К необходимости применения массивов мы приходим, когда требуется связать и использовать целый ряд родственных величин.
Массивы (Array) — упорядоченная структура однотипных данных, которой дано одно общее имя, причем количество компонентов фиксированное. Ее размер не должен превышать 64 Кб (размер сегмента памяти). Тип компонента может быть структурный.
Массивы могут быть как одномерными, так и многомерными.

Описание массива:

1-й способ:

Type <имя типа> **= Array** [диапазон1,…,диапазон n] **of** <тип компонентов>;
…..
Var <имя переменной>: <имя типа>;

2-й способ:
Var <имя переменной> **: Array** [диапазон1, …,диапазон n] **of** <тип компонентов>;

Диапазон может быть любым перечисляемым типом, но он должен помещаться в тип Word.

5.1.1. Одномерные массивы

Одномерный массив состоит из n элементов – это $a_1,a_2,\ldots,a_n$ – представляет собой строку или столбец. Элементы одномерного массива обозначаются a[i], где a - имя массива, i – номер элемента в массиве.

Пример. Географ передал набор показаний температуры, которые снимались в течение июня месяца. Необходимо вычислить:
а) среднюю температуру;
б) число дней, в которых температура была выше 23 градусов.

Набор температур образует массив. Дадим этому массиву имя t. Тогда массив t содержит 30 элементов, при этом каждый элемент представляет собой температуру одного из дней июня.

Элементы массива t можно записать в таком виде: t[1], t[2], t[3], t[4], ..., t[29], t[30].

Ниже показано, как назначаются элементы массива для 30 чисел, которые представляют собой июньские температуры.

Дата	Температура	Элемент
1 ИЮНЯ	15	t[1]
2 ИЮНЯ	18	t[2]
3 ИЮНЯ	23	t[3]
4 ИЮНЯ	25	t[4]
5 ИЮНЯ	24	t[5]
. . . . .	. . . . .	. . . . .
29 ИЮНЯ	20	t[29]
30 ИЮНЯ	21	t[30]

Например, значение элемента t[3]=23, а значению элемента t[29]=20. Важно помнить, что элементы t[1], t[2], t[3], ..., t[30] представляют собой просто отдельные числа.

```
Program Temperature;
  uses WinCrt;
                          var
  t : array[1..30] of integer; {описание массива}
    i, k : integer;{i- счетчик, k - количество дней для которых t>23}
    s    : real; {s- средняя температура}
  begin
```

```
          {ввод массива температур}
      for i := 1 to 30 do
begin
write('Введите температуру в ',i,' - день '); readln(t[i])
end;
      s := 0; k := 0;
      for i := 1 to 30 do
begin
s := s + t[i]; {вычисляем суммарную температуру}
if t[i] > 23 then k := k + 1
end;
s:=s/30; {вычисляем среднюю температуру}
      writeln('Средняя температура в июне ', s:4:2);
      writeln('Число дней с температурой больше 23 град. ', k)
   end.
```

Пример. Необходимо ввести одномерный массив и проверить, принадлежат ли его элементы отрезку [a,b] и, если принадлежат, запомнить их номера и вывести на экран. Отрезок ввести с клавиатуры.

```
uses wincrt;
var x: array[1..20] of real; {описание исходного массива}
   nom: array[1..20] of integer; {описание массива номеров  элементов,
принадлежащих заданному интервалу}
   a, b: real;
   i, k, n: integer;

  begin
k:=0; {количество искомых элементов}
writeln(‘введите отрезок [a,b], a < b:');
write('a='); readln(a);
write('b='); readln(b);
{ввод исходного массива}
write('размер массива n='); readln(n); {n<=20}
writeln('ввод массива');
for i:=1 to  n do begin
read(x[i]); {ввод N чисел в строку через пробел}
     if (x[i]>=a) and (x[i]<=b) then
 begin
k:=k+1; {вычисляем количество}
nom[k]:=i; {заполняем массив}
 end;
            end;
write('количество таких элементов: k=',k);
writeln;
write('их номера: ');
for i:=1 to k do write('  ',nom[i]); {печать нового массива}
end.
```

Пример. Составить программу перестановки m - го и k-го элементов одномерного массива. Рассмотрим решение задачи на частном примере.

Пусть задан произвольный массив чисел:

3	-12	45	16	-23	4	-5	76	-34
a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

Пользователь потребовал, чтобы были переставлены 5-й и 8-й элементы, т. е. -23 и 76. Если выполнить сразу команду присваивания a[5] := a[8], то значение элемента a[5] будет "*стерто*" и безвозвратно утеряно. Чтобы этого не произошло, нужна еще одна команда, которая бы "*запоминала*" значение одного из переставляемых элементов. Пусть эта переменная с именем p, тогда картина перестановки будет такой:

p := a[5] - "*запоминается*" 5 - й элемент,
a[5] := a[8] - на место пятого элемента "*ставится*" восьмой,
a[8] := p - на место восьмого элемента "*ставится*" пятый.

```
uses wincrt;
var a: array[1..9] of real; {описание исходного массива}
   p: real;
   i, k, m,n: integer;
  begin
writeln(‘введите m , k:');
write('m='); readln(m);
write('k='); readln(k);
{ввод исходного массива}
write('размер массива n='); readln(n);
writeln('ввод массива');
for i:=1 to  n do
read(a[i]); {ввод N чисел в строку через пробел}
```

```
        p := a[k];
a[k] := a[m];
a[m] := p;
for i:=1 to n do write('  ',a[i]); {печать нового массива}
writeln;
end.
```

Пример. Найти максимальный элемент числового массива A(20) и его номер.

Алгоритм: Пусть задан некоторый числовой массив (снова для простоты рассуждений зададим массив чисел):

A	45	25	93	35	…
i	1	2	3	4	….

В начале в качестве наибольшего элемента будем считать первый элемент и затем последовательно сравниваем его с остальными элементами массива, как только встретится элемент, больший максимального, то его (встретившийся больший элемент) считаем наибольшим и запоминаем его номер. Продолжаем дальнейшее сравнение до конца массива.

Для нашего примера этот процесс будет выглядеть так: Принимаем в качестве наибольшего –A(1) (max=45, im=1) и сравниваем его с последующими элементами A(2)> max (25>45–?) (нет), идем дальше; A(3)>max (93>45–?) (да) - в качестве наибольшего принимаем A(3) (max=93) и запоминаем его номер (im=3) продолжаем сравнение; A(4)> max ((35>93–?)) (нет) и т.д. В результате переменная max будет равна максимальному элементу, im– будет содержать его номер.

```
uses wincrt;
var a: array[1..9] of real; {описание исходного массива}
   max: real;
   i, im,n: integer;

  begin
{ввод исходного массива}
        write('размер массива n='); readln(n);
        writeln('ввод массива');
        for i:=1 to  n do
        read(a[i]); {ввод N чисел в строку через пробел}
        max := a[1];
         for i := 2 to n do
                if a[i]> max then
                        begin max := a[i] ; im:=i;
                        end;
        writeln('max=',max,' im=',im);
end.
```

5.1.2. Методы сортировки массивов

При решении задач сортировки обычно выдвигается требование минимального использования дополнительной памяти, из которого вытекает необходимость применения дополнительных массивов.
Для оценки быстродействия алгоритмов различных методов сортировки, как правило, используют два показателя:
количество присваиваний;
количество сравнений.

Все методы сортировки можно разделить на две большие группы:
прямые методы сортировки;
улучшенные методы сортировки.

Прямые методы сортировки по принципу, лежащему в основе метода, в свою очередь, разделяются на три подгруппы:
сортировка вставкой (включением);
сортировка выбором (выделением);
сортировка обменом (“пузырьковая” сортировка).

Улучшенные методы сортировки основываются на тех же принципах, что и прямые, но используют некоторые оригинальные идеи для ускорения процесса сортировки. Прямые методы на практике используются довольно редко, так как имеют относительно низкое быстродействие. Однако они хорошо показывают суть основанных на них улучшенных методов. Кроме того, в некоторых случаях (как правило, при небольшой длине массива и/или особом исходном расположении элементов массива) некоторые из

прямых методов могут даже превзойти улучшенные методы.

Сортировка вставкой

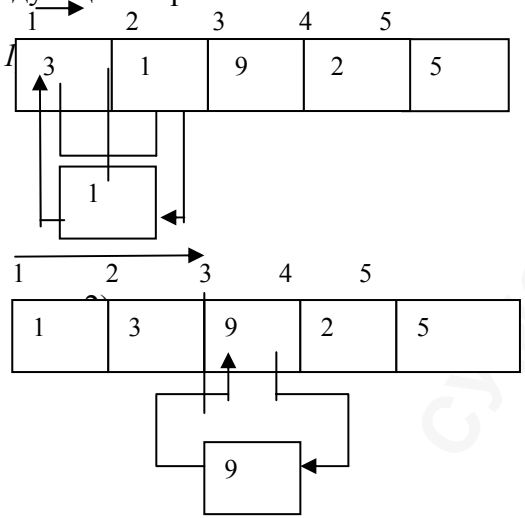
Сортировка массивов методом вставки(включением) осуществляется следующим образом. Массив разделяется на две части: отсортированную и неотсортированную. Элементы из неотсортированной части поочерёдно выбираются и вставляются в отсортированную часть так, чтобы не нарушить в ней упорядоченность элементов. В начале работы алгоритма в качестве отсортированной части массива принимают только один первый элемент, а в качестве неотсортированной части – все остальные элементы.

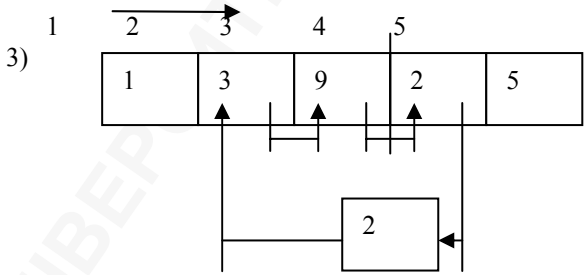
Таким образом, алгоритм будет состоять из *n-1*-го прохода (*n* – размерность массива), каждый из которых будет включать четыре действия:

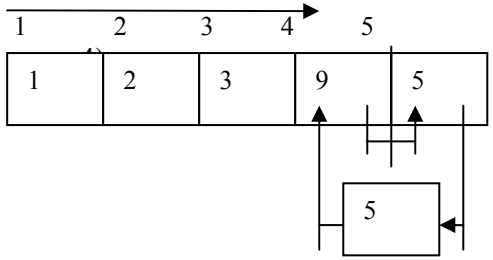
- взятие очередного *i*-го неотсортированного элемента и сохранение его в дополнительной переменной;
- поиск позиции *j* в отсортированной части массива, в которой присутствие взятого элемента не нарушит упорядоченности элементов;
- сдвиг элементов массива от *i-1*-го до *j*-го вправо, чтобы освободить найденную позицию вставки;
- вставка взятого элемента в найденную *j*-ю позицию.

Для реализации данного метода можно предложить несколько алгоритмов, которые будут отличаться способом поиска позиции вставки. Рассмотрим схему реализации одного из возможных алгоритмов.

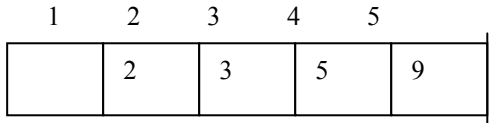
Схематично описанные действия можно представить следующим образом:







Результат:



Сортировка выбором

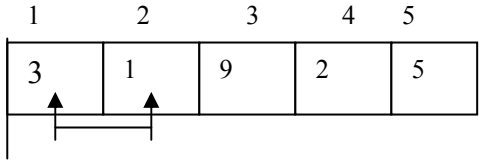
Сортировка массивов методом выбора (выделения) осуществляется таким образом.

Находим (выбираем) в массиве элемент с минимальным значением на интервале от *1*- го элемента до *n*-го (последнего) элемента и меняем его местами с первым элементом. На втором шаге находим элемент с минимальным значением на интервале от *2*-го до *n*-го элемента и меняем его местами со вторым элементом.

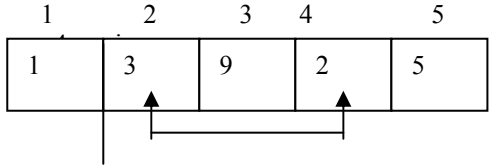
И так далее для всех элементов до *n-1*-го.

Рассмотрим схему алгоритма прямого выбора.

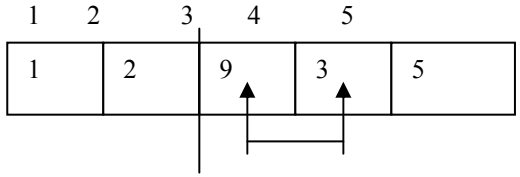
1)



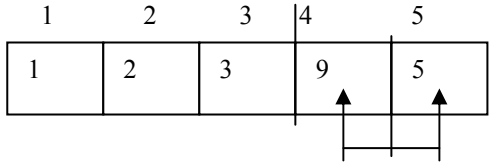
2)



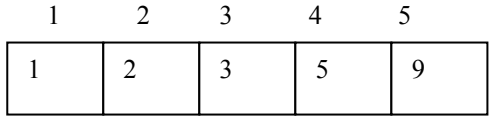
3)



4)



Результат:



Сортировка обменом (“пузырьковая”сортировка)

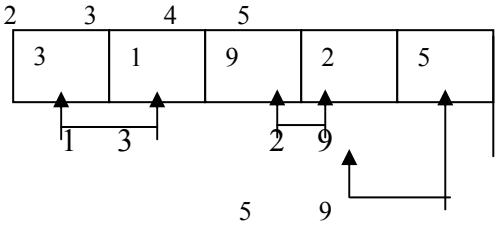
Сортировка массивов методом выбора (выделения) осуществляется таким образом.

Слева направо поочерёдно сравниваются два соседних элемента, и если их взаиморасположение не соответствует заданному условию упорядоченности, то они меняются местами. Далее берутся два следующих соседних элемента и так далее до конца массива.

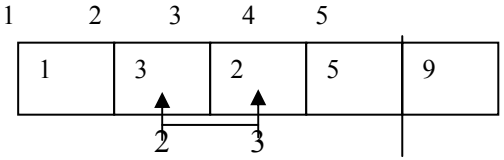
После одного такого прохода на последней *n*-ой позиции массива будет стоять максимальный элемент (“всплыл” первый “пузырёк”). Поскольку максимальный элемент уже стоит на своей последней позиции, то второй проход обменов выполняется до *n-1*-го элемента. И так далее. Всего требуется *n-1* проход.

Рассмотрим схему алгоритма сортировки методом прямого обмена по неубыванию.

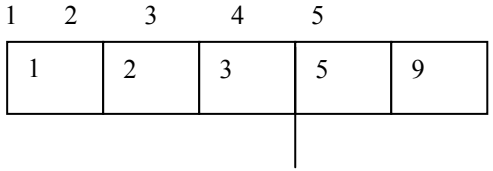
1)



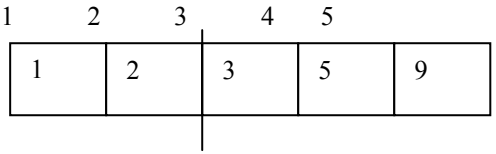
2)



3)



4)



Результат:

1	2	3	4	5
1	2	3	5	9

Ниже приведены фрагменты программ для реализации этих методов сортировки

```
uses wincrt;
   type mas = array[1..1000] of real;
        var n: integer; {размер массива}
        x:mas;
        i,j,c,k: integer;
        b: real;

                                  begin
        ..{ввод размера массива N
        и значений элементов массива}

        .   .    .    .    .
        writeln('..:::Метод «пузырька»::::.. ');
        c:=n-1;
                                  repeat
                for i:=1 to c do
                                          begin
                        if x[i]>x[i+1] then
                                                  begin
                                b:=x[i];
                                x[i]:=x[i+1];
                                x[i+1]:=b;
                                end;
                end;
                c:=c-1;
        until c<1;
{--------------------}
        writeln('...:::Метод выбора:::.');
        for i:=2 to n do
                                          begin
                b:=x[i];
                j:=1;
                while (b>x[j]) do j:=j+1;
                        for k:=i-1 downto j do
                        begin
                                x[k+1]:=x[k];
                        end;
                        x[j]:=b;
                end;
{--------------------}
        writeln(':::Метод вставки:::. ');
        for i:=2 to n do
                begin b:=x[i];
                        j:=1;
                        while (b>x[j]) do
                                j:=j+1;
                        for k:=i-1 downto j do
                        begin
                                x[k+1]:=x[k];
                        end;
                        x[j]:=b;
                end;
end.
```

Сравнение прямых методов сортировки

Теоретические и практические исследования алгоритмов прямых методов сортировки показали, что как по числу сравнений, так и по числу присваиваний они имеют квадратическую зависимость от длины массива *n*. Исключение составляет метод выбора, который имеет число присваиваний порядка *n*ln(n)*. Это свойство алгоритма выбора полезно использовать в задачах сортировки в сложных структурах данных, когда на одно сравнение выполняется большее число присваиваний. В таких задачах метод выбора успешно конкурирует с самыми быстрыми улучшенными методами сортировки.

Если сравнить рассмотренные прямые методы между собой, то в среднем методы вставки и выбора оказываются приблизительно эквивалентными и в несколько раз (в зависимости от длины массива) лучше, чем метод обмена (“пузырька”).

5.1.3. Двумерные массивы

Двумерный массив представляет собой прямоугольную матрицу, состоящую из *m* – строк и *n* – столбцов. Для определения элементов в двумерном массиве надо указывать *два* индекса, *номер строки* и *номер столбца*. Элементы *двумерного* массива обозначаются a[i, j], где a - имя массива, i - номер строки, j - номер столбца.
В разделе описаний массив надо описать так:

Первый способ

type <название типа> = **array**[1..m, 1..n] **of** <тип>;
var <имя переменной> : < название типа >;

Второй способ

var <имя переменной> : **array**[1..m, 1..n] **of** <тип>;

Например:

♦ Type mas : array[1..10,1..15] of real; {описываем новый тип данных: двумерный массив из 10 строк и 15 столбцов действительных чисел }
Var x:mas;

♦ Var x : array [1..10,1..15] of real;

Пример. Сформировать одномерный массив C, каждый элемент которого равен произведению элементов столбцов матрицы X(10x10), умноженному на максимальный элемент этого столбца.

```
uses wincrt;
var x: array[1..10, 1..10] of real;
   i, j, m, n: integer;
   p, xmax: real;
   c: array[1..10] of real;
begin
{Ввод матрицы}
  writeln ('_ВВОД МАТРИЦЫ_');
  write (‘количество строк m=’);
  readln(m);
  write (‘количество столбцов n=’);
  readln(n);
  for i:=1 to m do
  for j:=1 to  n  do read(x[i,j]);
```

{ Цикл по i задает номера строк массива. После этого начинает работать цикл по j - по количеству элементов в строке, т. е. по количеству столбцов. После ввода элементов одной строки (элементы вводятся через пробел, в конце строки нажать ENTER), цикл по i повторяется, вводятся элементы следующей строки и так далее. }

```
for j:=1 to  n  do
 begin
  p:=1;
  xmax:=x[1,j];
for i:=1 to m do
begin
if xmax<x[i,j] then xmax :=x[i,j];{Находим максимальный элемент
столбца}
  p:=p*x[i,j]; {Вычисляем произведение элементов столбца}
end;
 c[j]:= p*xmax;
end;
{Печать полученного массива }
writeln(‘масив C:’);
for i:=1 to n do write(‘ ’, c[i]:2:1);
end.
```

Пример. Транспонирование квадратной матрицы.

```
Program Transp;
uses wincrt;
var i, j, m, n: integer;
```

```
   x: real;
    Matr: array [1..10,1..10] of real;
begin
    writeln ('___ВВОД МАТРИЦЫ___');
write (‘количество строк m=n= ’); readln(m);
n:=m;
for i:=1 to m do
for j:=1 to n do  read(Matr[i,j]);

{Транспонирование матрицы}
for i:=1 to m do
for j:=i+1 to n do
  begin
       x:=Matr [i,j];
Matr [i,j]:=Matr [j,i];
Matr [j,i]:=x;
  end;

{Вывод транспонированной  матрицы на экран
Для вывода элементов массива на экран организованы два цикла, но по
завершению внутреннего цикла по j включен пустой оператор writeln,
чтобы следующие элементы выводились с начала новой строки.}

    for i:=1 to m do   begin
    for j:=1 to n do
    write ('  ',Matr[i,j]:2:1);
    writeln;
    end;
End.
```

5.2 Сроковый тип – STRING

Строковый тип – аналогичен типу одномерного массива, но его компоненты имеют тип CHAR**.** Длина переменной данного типа по умолчанию равняется 255 символов, может быть также ограничена в разделе описаний. Компонента с нулевым номером содержит длину строки. Строковые переменные описываются следующим образом:

Var <Имя типа> : **STRING;**
<Имя типа> : **STRING[n];**

В отличие от массивов количество символов может изменяться от 0 до 255. Если максимальный размер строки не указан, то он берется равным 255 символам.

Пример.

```
Var a1: string [2]; a2: string;
Begin
….
A1:=’ДА’;
A2:=’Может быть’;
…
end.
```

К любому символу можно обращаться посредством индекса.

Пример.

```
A2:=’Осень’;
Writeln(a2[3]);   {На печать будет выведена буква е}
Writeln(ord(a2[0]); { ord(a2[0] )дает длину строки, }
A2:=A2+’ пришла’; {Знак +  «склеивает» строки}
Writeln(a2); {На печать будет выведено сообщение “ Осень пришла”}
```

Для данного типа имеется библиотека дополнительных функций:
Function Concat (<переменная1>,…,<переменная n>) **–** слияние строчных переменных.
Function Copy (<имя строки>, <позиция>, <количество символов>) – возвращает подстроку из заданной строки, начиная с <позиции> длиной <количество символов>.

```
A:=’baby_12_toy’;
b:= Copy(a,6,3); {  из строки a, начиная с 6-й позиции, копируется 3
символа. Результат: b= 12_}
```

Procedure Delete (<имя строки>, <позиция>, <количество символов>) – удаление.
Procedure Insert (<имя вставки>,< имя строки>, <позиция>) – вставка.
Function Upcase (<переменная>) **–** возвращает для строчных латинских букв соответствующие большие буквы.
Function Pos (<имя подстроки >, <имя строки>) **–** поиск 1-го вхождения фрагмента (подстроки) в данную строку.

Function Length (<имя строки >) – возвращает длину строки, результат типа integer.
Procedure Val(<имя строки>, <переменная целого или действительного типа>, <параметр>) – превращает строку символов в значении целого или действительного типа в зависимости от типа переменной. Если преобразование удачное, <параметр> = 0, если нет – он содержит номер позиции, где произошла ошибка преобразования.

Пример. Подсчитать количество цифр в строке.

```
uses wincrt;
var  lng, i, h: word;
     St: string;
begin
   writeln ('введите строку');
   readln (st);
   h:=0;
   lng:=ord (st [0]);
 for i:=1 to lng do
    if (st[i]>='0') and (st[i]<='9') then
        h:=h+1;
write ('количество цифр=', h);
end.
```

5.3. Множества

Множества – наборы однотипных объектов. Множество может содержать до 255 элементов.

Type <имя типа> **= Set of <**тип элемента>

Где <тип элемента> - базовый тип элементов множества, в качестве которого может использоваться любой порядковый тип, кроме WORD, INTEGER, LONGINT.

Пример.

```
uses wincrt;
Type     dc = Set of '0'…'9';
         dt=Set of 0…9;
var      s1,s2,s3:dc;
         s4,s5,s6:dt;
begin
                  ……
s1:=['1','2','3'];
s2:=['3','2','1'];
s3:=['2','3'];
s4:=[0…3,6];
s5:=[4,5];
s6:=[3…9];
……
end.
```

Основные операции над множествами:

+ объединение; результат содержит элементы первого множества, дополненные недостающими элементами со второго:
s4+s5 →[0,1,2,3,4,5,6];
s5+s6 → [3,4,5,6,7,8,9];
***** пересечение; результат содержит элементы, общие для обоих множеств:
s4*s6 → [3,6].
- разность; результат содержит элементы из первого множества, которые не принадлежат второму:
s6-s5 → [3,6,7,8,9];
= проверка эквивалентности; возвращает значение TRUE, если оба множества эквивалентны.
<> проверка неэквивалентности; возвращает значение TRUE, если оба множества неэквивалентны.
<= проверка включения; возвращает значение TRUE, если первое множество включается во второе.
>= проверка включения; возвращает значение TRUE, если второе множество включается в первое.
IN проверка принадлежности; в этой операции первый элемент – выражение, а второй – множество. Возвращает TRUE, если выражение имеет значение, которое принадлежит множеству:
3 in s6 возвращает TRUE;
2*2 in s1 возвращает FALSE;

5.4 Записи

Записи **–** структурный тип данных, состоящий из фиксированного числа компонентов разного типа. Компоненты записи называются ***полями записи***, характеризуются типами.

Type <имя записи> **= record**
<список полей>
end;
Список полей – перечень имен полей с объявлениями их типа (через ;):
<имя>:<тип>;

Пример. Описать запись, содержащую информацию о группе студентов: фамилию, средний балл, номер группы.

```
Type  stud = record
Fam: string;
oc: real;
gr: string;
end;
var b: stud;
```

К каждому из компонентов записи можно получить доступ, если указать имя переменной-точка-имя поля:
b.fam: = ‘Иванов’;
b.oc:= 4;
Поля записи могут быть также структурного типа. Для упрощения доступа к полям вводится оператор **WITH:**

With <имя записи> **do** <действия над полями>

Работают с полями как с обычными переменными, без использования составленных имен.

With b **do** Fam: = ‘Иванов’;

Пример. Имеем список студентов из 10 человек. В списке даются фамилия и оценка. Обработать массив с целью выбора студентов с неудовлетворительной оценкой.

```
uses wincrt;
Type   stud=record
      Fam:string[20];
      Mh:byte;
end;
var mas: array [1..10] of stud;
   i: integer;
begin
   for i:=1 to 10 do
           begin
           writeln(‘ Фамилия – ‘);  readln(mas [i].fam);
           writeln(‘ Оценка – ‘);  readln ( mas [i].mh);
                                                  end;
                                                writeln;
   for i:=1 to 10 do
        with mas [i] do
         if mh=2 then writeln(fam);
end.
```

ТЕМА 6. ПРОЦЕДУРЫ И ФУНКЦИИ

6.1. Блочная структура программы

Часто при решении задач необходимо повторить группу операторов – блок – в нескольких местах программы.

Блоки – инструмент разбиения программы на ряд почти независимых частей.

Решая какую-либо задачу, программист должен видеть в целом программу, а потом разбить ее на отдельные части, составить на выбранном языке программирования эти части программы, объединить их в единое целое и получить программу.

Итак, весь творческий процесс можно разбить (разумеется, чисто условно) на следующие этапы:

1) основная идея решения задачи;
2) общая конструкция программы;
3) выделение отдельных, элементарных частей программы;

4) практическая реализация на языке программирования этих частей программы;

5) объединение их в единую программу.

Такой процесс программирования называют структурным или нисходящим. Для облегчения такой работы и созданы *подпрограммы*.

Подпрограммой называется группа операторов, к которой обращаются из основной программы несколько раз.

I.Использование подпрограмм позволяет:

1) сделать основную программу более наглядной и компактной;

2) уменьшить объем используемой памяти ЭВМ;

3) сократить время отладки программы.

На языке Паскаль *подпрограммы* бывают двух видов - это *процедуры* и *функции*.

Функция **отличается** от процедуры тем, что вычисляет только одно значение, в то время как процедура может вычислять несколько значений либо ни одного. Поэтому функции можно использовать в выражениях рядом с константами и переменными. Процедуры, хотя и могут возвращать результат, но передают его переменной. Вызов процедур и функций осуществляется указанием их **имени** и **параметров** в программе. Каждая процедура или функция должна быть описана в разделе описаний программы.

Пример структурного программирования

```
Program main;
uses wincrt;
Var V1,…{Переменные основной программы}
Procedure A;
  Var V2,…        {Переменные процедуры}
begin
              Procedure A1;
            Var V3,; {Переменные A1}          A1
            Begin … End;
  End;
Function B1(…..):<тип>;
Var W1,…:…; {Переменные B1}     B1
            Begin … End;

Begin
{основная программа}
{вызов процедур и функций}
end.
```

A

6.2. Подпрограмма-функция

Подпрограмма-функция предназначена для вычисления одного результата. Описание подпрограммы-функции начинается с ключевого слова **Function,** затем указывается имя функции и в скобках список формальных параметров и их типов, после «:» указывается тип результата.

Function <имя> (<параметр1>:<тип1>;…;<параметр n >:<тип n>): <тип результата>;
<раздел описаний>; (**var, label** и т.д.)
Begin
<тело функции>;
{последовательность операторов}
end;

В теле функции хотя бы один раз необходимо присвоить значение переменной, имя которой совпадает с именем функции:

<имя функции>:=…;

Вызов функции в основной программе осуществляется указанием ее имени и списка фактических параметров, которые должны совпадать с формальными по количеству, типу и порядку следования.

Пример. Вычислить $C_n^m = \frac{n!}{(n-m)!\cdot m!}$ (вычисление факториала оформим в виде функции).

```
Program Soch;
uses wincrt;
var c: real;
   m, n: integer;
{описание функции}
function fact(n: integer): longint;
      {формальные ↑параметры}
    var f: longint;
       i: byte;
 begin
          if n=0 then fact:=1
          else begin
```

```
                    f:=1;
                    for i:=1 to n do f:=f*i;
                    fact:=f;
                end;
  end;
{основная программа}
begin
{ввод значений m и n}
  writeln('m,n-?');
  readln(m,n);
{обращение к функции с тремя разными параметрами}
  c:=fact(N)/(fact(n-m)*fact(m));
  {       ↑          ↑          ↑   }
  {  фактические параметры  }

  write('c= ', c);
end.
```

6.3. Подпрограмма-процедура

Процедуры помещаются в разделе описаний и начинаются зарезервированным (служебным) словом **Procedure.**

Процедуре **обязательно** дается имя, которое должно удовлетворять тем же требованиям, что и имена переменных. После имени в скобках записываются формальные параметры и их тип: ***входные*** параметры, значения которых используются для вычисления в качестве аргументов, и ***выходные*** параметры - это те переменные, в которых получается результат выполнения процедуры.

Входные и выходные параметры процедуры называются формальными параметрами.

Фактические, конкретные значения формальные параметры должны получить в основной программе после обращения к ней (а пока в процедуре они являются не чем иным, как "**пустышками**").

Надо заметить, что процедура может быть такой, что в ней не будет вообще параметров, достаточно тех, которые будут введены из программы.

Описание процедуры имеет вид

```
 Procedure <имя> (<входные параметры> : <их тип>;
     var  <выходные параметры> : <их тип>);
{раздел описаний var, label и т.д.}
    begin
      (раздел операторов)
    end;
```

Вызов процедуры в основной программе осуществляется указанием ее имени и списка фактических параметров, которые должны совпадать с формальными по количеству, типу и порядку следования.

Пример. Вычислить значения корней уравнений:

$$x^2 + 4x - b = 0,$$

$$ay^2 + cy + 5 = 0.$$

```
Program Kvu;
uses wincrt;
var a, b, c, x1, x2, y1, y2: real;
{Описание процедуры нахождения корней квадратного уравнения}
procedure Kv( a, b, c: real;  var z1, z2: real);
                  ↑                    ↑
{параметры значения} {параметры переменные}
  var d: real;

          begin
          d:=sqr(b)-4*a*c;
          if d<0 then begin
                  writeln('D<0');
                  halt;
                  end;
          z1:=(-b+sqrt(d))/(2*a);
          z2:=(-b-sqrt(d))/(2*a);
          end;

{основная программа}
  begin
          writeln('a,b,c:');
          readln(a,b,c);
          Kv(1,4,-b,x1,x2);
           {       ↑       }
          {фактические параметры}
```

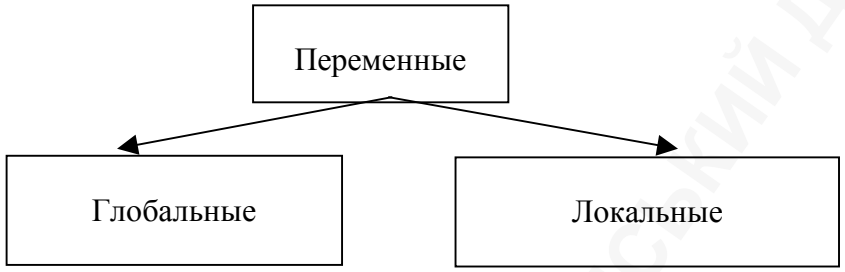
```
        writeln('x1= ',x1);
        writeln('x2= ',x2);
        readln;
        Kv(a,c,5,y1,y2);
          {      ↑       }
        {фактические параметры}

        writeln('y1= ',y1);
        writeln('y2= ',y2);
end.
```

6.4. Локализация имен

Все переменные, описанные в середине блока (процедуры или функции), называются ***локальными,*** и они недоступны извне.

Всякий блок перед его использованием должен быть описан. Поэтому блоку доступны те объекты верхнего уровня, которые были описаны до появления данного блока. Эти объекты называются ***глобальными***. Имена переменных в теле процедур или функций должны быть уникальными и не совпадать с именами глобальных переменных. При совпадении имени глобальной переменной с локальной значение глобальной переменной внутри блока теряется.

Любая переменная основного тела программы может использоваться в процедурах и функциях. Переменные, описанные в процедурах и функциях, теряют свое значение после завершения процедуры или функции.



Процедуры и функции могут иметь вложенный характер, но описание процедур и функций, которые используются внутри других процедур и функций, должны стоять в теле программы раньше, чем данная процедура или функция.

Пример.

```
Program pr;
uses wincrt;
Var i: integer;
        Procedure wp;
          Var i: integer;
           Begin
                  Writeln(i);
           End;
Begin
  i :=  1;
  wp;
End.
```

Программа напечатает что-либо (неинициализированное i, так как локализованная переменная "перекроет" глобальную). Если удалить описание переменной i из процедуры, то будет напечатана ***1***.

Пример. Сложение двух величин.

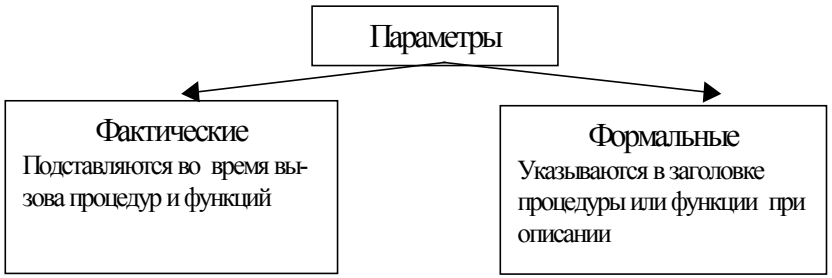
Если использовать процедуру, то программа будет выглядеть следующим образом:

```
Program sp;
uses wincrt;
  Var a: integer;
          Procedure sum (b,c: integer; var k:integer);
                  begin
                   k:= b + c;
                  end;
begin    {основная программа}
  Sum (12, 3, a);
  Writeln('Сумма = ', a);
End.
```

С использованием функции программа примет следующий вид:

```
Program sf;
uses wincrt;
  Var a: integer;
        Function sum (b,c: integer): integer;
                  begin
                  Sum:= b + c;
                  end;
  begin   {основная программа}
          A := Sum (12, 3);
          Writeln('Сумма = ', a);
  End.
```

6.5. Параметры. Их взаимодействие



Параметры:

Формальные

(при описании процедур и функций):

1) параметры-значения;

2) параметры-переменные.

Фактические

(при вызове процедур и функций;

их количество, порядок, а значит, и тип должны соответствовать формальным).

Пример.

```
Procedure prim1 (a,b:char; var c,d:integer);
```

Формальные параметры могут быть или значением, или переменной (b,c – значение, k – переменная).

Переменной при вызове блока должен отвечать фактический параметр – переменная того же типа.

Параметр-значение: ему в качестве фактического может отвечать произвольное значение. Фактическое значение передается внутрь блока (его копия), и изменение параметрического значения внутри блока не влияет на вызывающую программу.

Параметр-переменная: при вызове блока передается сама переменная (а не копия!), и любые изменения этого параметра внутри блока приведут к изменению фактических параметров в вызывающей программе.

Поэтому параметр-переменная используется для связи блока с вызывающей программой, а именно – для передачи результатов своей работы.

Пример. Удвоить числа 5 и 3 в процедуре.

```
Program pr1;
uses wincrt;

  Var x, y: integer;
        Procedure inc2 (var a: integer; b: integer);
                  Begin
                  a := a + a; b := b + b;
                  writeln ('Удвоенные ',a,'  ', b);
        End;
Begin
        Writeln ('Исходные данные');
        Readln (x, y);
        Inc2 (x, y);
        Writeln ('Результат ',x, '  ',y);
End.
```

Результат: Исходные данные

5 3

Удвоенные 10 6

Результат 10 3

6.6. Использование массивов и строк в качестве параметров

Формальные параметры могут быть стандартного или ранее объявленного типа. Поэтому, если надо передавать как параметр массив или строку, они должны быть заранее описаны. То есть нельзя описать параметры массива в блоке так:

Procedure A (B: array [1..10] of real);

так как в списке формальных параметров фактически объявляется тип-диапазон, указывающий границы индексов массива.

Если мы хотим передать какой-то элемент массива, то проблем не возникает, но если в подпрограмму передается весь массив, то следует первоначально описать его тип, а затем описывать процедуру.

Например.

Type mas= array [1..10] of real;
Procedure A (B: mas);

Пример. Вычислить $\alpha = \frac{\sqrt{\left|\sum_{i=1}^{10} x_i\right|}}{(\sum_{i=1}^{6} y_i)^2}$. Ввод массивов организовать с помощью процедуры, вычисление суммы – используя функцию.

Для написания процедуры ввода массива, назовем ее VVOD, проанализируем, какие параметры она должна содержать. Входные параметры отсутствуют. В результате работы процедуры мы должны получить размер массива (N) и сам заполненный массив (Z). Схематически это можно изобразить следующим образом:

↓ входные параметры отсутствуют

VVOD

↓ Выходные параметры – N, Z:
N –размер массива, Z – массив

Напомним, что выходные параметры должны быть описаны как параметры переменные, т.е. перед ними ставится ключевое слово Var.

Для функции вычисления суммы элементов массива (назовем ее SUM) входными параметрами будут являться размер массива (N) и сам массив (Z)). Схематически это можно изобразить следующим образом:

↓ Входные параметры – N, Z:
N –размер массива; Z – массив

SUM

↓ Выходные параметры: SUM – сумма элементов массива

```
Program pr2_F;
uses wincrt;
Type mas=array[1..10] of real;
Var x,y: mas;
   Nx,Ny: byte;
   alfa: real;
   {Описание процедуры ввода массива}
  Procedure Vvod(var n:byte; var z:mas);
    Var i:byte;
      Begin
            Writeln('Введи размер массива'); Readln(N);
            Writeln('Введи ', N,' чисел');
            For i:=1 to N do
                  Read(Z[i]);
    End;
```

```
{Описание функции вычисления суммы элементов массива}
 Function Sum ( n:byte; z:mas):real;
   var j:byte;s:real;
     begin
              s:=0;
              for j:=1 to N do s:=s+z[j];
               sum:=s;
     end;
{основная программа}
begin
{ввод массивов x,y}
  Vvod(Nx,X);
  Vvod(Ny,Y);
    alfa := sqrt(abs(sum(Nx,X)))/sqr (sum(Ny,Y));
                          ↑                    ↑
              {обращение к функции SUM}
  writeln ('alfa=',alfa:6:3);
end.
```

Вычисление суммы элементов массива можно также оформить и в виде процедуры, тогда программа примет вид:

```
Program pr2_P;
uses wincrt;
Type mas=array[1..10] of real;
  Var x,y: mas;
   Nx,Ny: byte;
   Alfa,Sx,Sy: real;
{Описание процедуры ввода массива}
  Procedure Vvod(var n:byte; var z:mas);
              Var i:byte;
  Begin
              Writeln('Веди размер массива'); Readln(N);
              Writeln('Введи ', N,' чисел');
              For i:=1 to N do
              Read(Z[i]);
End;
{Описание процедуры вычисления суммы элементов массива}
Procedure Sum ( n:byte; z:mas; var s:real);
  var j:byte;
  begin
              s:=0;
              for j:=1 to N do s:=s+z[j];
  end;
{основная программа}
  begin
{ввод массивов x,y}
              Vvod(Nx,X);
              Vvod(Ny,Y);
{обращение к процедуре SUM для массива X}
              Sum(Nx,X,Sx);
{обращение к процедуре SUM для массива Y}
              Sum(Ny,Y,Sy);
              alfa := sqrt(abs(Sx))/sqr (Sy);
              writeln ('alfa=',alfa:6:3);
  end.
```

Можно использовать при оформлении параметров функций и процедур параметры без типа (тогда соответствующая фактическим параметрам переменная может быть любого типа).

Procedure S (**var** x);

Бестиповые параметры можно использовать внутри функции и процедуры, только наделенные определенным типом.

Absolute – конструкция выделяет общую область памяти для сохранности переменных.

```
Var x:array [1..5]of real;
Var x:T (тип) absolute S;
```

Удобно использовать для передачи одномерных массивов переменной длины.

```
Procedure Demo (var x);
Var y:array [1..10] of real absolute x;
```

6.7. Процедурные типы

Как параметры могут передаваться имена других функций и процедур, то есть мы можем процедуру или функцию использовать при обращении к другим процедурам и функциям. Для этого нужно оформить процедурный тип.

```
Type   fun=function(w:integer):real;
```

```
    pr=procedure (a,b: real);
Var   x:pr; y:fun;
```

Важно, чтобы компиляция осуществлялась для дальней модели связывания.

Для этого применяют директивы компилятора:

{$F+} - включение дальней модели вызова;

{$F-} – выключение дальней модели вызова;

Или после описания процедуры или функции пишут зарезервированное слово Far;

Пример.

Вычислить $Z=\sum_{i=1}^{40}\sin x_i + \sum_{i=1}^{50}\cos y_i - \sum_{i=1}^{40}\ln x_i$, где x_i и y_i - заданные массивы. Все суммы вычислять в одной функции.

```
Uses wincrt;
Type  single_matrix=array[1..20] of integer;
     fun=function(w:integer):real;
Var   x,y:single_matrix;
     z:real;
     m,n:integer;
Function Cos1(xx:Integer):Real; Far;
Begin
  Cos1:=Cos(xx);
End;
Function Sin1(xx:Integer):Real; Far;
Begin
  Sin1:=Sin(xx);
End;

Function Ln1(xx:Integer):Real; Far;
Begin
  If xx>0 then Ln1:=Ln(xx)
  else Ln1:=0;
End;

Procedure Generate (Var a:single_matrix; v:integer);
{Заполняет массив случайными числами}
Var k:integer;
Begin
  For k:=1 to v do begin
  a[k]:=Random(20)-10;
  Write(a[k],' ');
  end;
 Writeln;
 end;

Function Sum(a:single_matrix; v:integer; f1:fun):real;
Var k:integer;
    s:real;
begin
  S:=0;
  For k:=1 to v do S:=s+f1(a[k]);
  Sum:=s;
end;

Begin
  ClrScr;
  Randomize;
  Write('Введите размеры массивов x и y:');
  ReadLn(m,n);
  Generate(x,m);
  Generate(y,n);
  z:=Sum(x,m,sin1)+Sum(x,m,cos1)-Sum(y,n,ln1);
  WriteLn('Результат:',z:3:5);
  Readkey;
End.
```

6.8. Опережающее описание

Возможны случаи, когда подпрограмма обращается к себе опосредованно путем вызова другой подпрограммы, в которой содержится обращение к первой, например:

```
Procedure A(i: real);
          begin
        ...
        B(i);
        ...
        end;
Procedure B(j: integer);
```

```
...
begin
...
A(j);
...
end;
```

Если строго следовать правилу, согласно которому каждый идентификатор перед употреблением должен быть описан, то такую программную конструкцию использовать нельзя. Для того чтобы такого рода вызовы стали возможны, вводится ***опережающее*** описание:

```
Procedure B(j: integer);  forward;
 Procedure A(i: real);

        begin
            ...
           B(i);
            ...
        end;
    Procedure B;
        begin
            ...
           A(j);
            ...
        end;
```

Как видим, опережающее описание заключается в том, что объявляется лишь заголовок процедуры **B**, а ее тело заменяется стандартной директивой **FORWARD.** Теперь в процедуре **A** можно использовать обращение к процедуре **B** - ведь она уже описана, точнее известны ее формальные параметры, и компилятор может правильным образом организовать ее вызов.

Обратите внимание: тело процедуры **B** начинается заголовком, в котором уже не указываются описанные ранее формальные параметры.

Используя **Forward -** описания (предописания), вы можете делать процедуры или функции “известными” без фактического определения ее операторной части. С точки предописания другие процедуры и функции могут вызвать описанную предварительно подпрограмму, делая возможной взаимную рекурсию. Где-нибудь после предописания тело процедуры или функции должно быть определено соответственно объявлению, которое определяет операторную часть подпрограммы..

Пример. Forward - процедура

```
 uses wincrt;
var k:integer;
Procedure Flip(N : Integer); Forward;

 Procedure Flop(N : Integer);
                                  Begin
 WriteLn('Flop ',n);
 If N > 0 Then Flip(N-1);
End;

 Procedure Flip;
                                  Begin
 WriteLn('Flip ',n);
 If N > 0 Then Flop(N-1);
End;
                                  begin
    write('k -? ');readln(k);
    flip(k);
    end.
```

Результат работы:

```
'k -? 5
Flip 5
Flop 4
Flip 3
Flop 2
Flip 1
Flop 0
```

6.9. Рекурсии

Рекурсия – такая организация блока, при которой он обращается сам к себе.

В Pascal рекурсии организовываются с помощью процедур и функций, которые обращаются сами к себе.

Пример. Найти **n!**

```
Program factor;
uses wincrt;
var n: word;
Function f (k: word):real;
                begin
                        if (k = 0) or (k=1) then f:=1
                                else f:= k * f(k-1);
                end;
begin
        readln (n);
        writeln ('N – факториал = ', f(n));
end.
```

Разберемся детально в работе этой процедуры. Для вычисления факториала числа n, т.е. n! надо умножить последовательно n натуральных чисел от 1 до n: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n.$

Пусть введено в программу значение 4 для вычисления факториала 4! Как будет работать процедура?

После начала ее работы будет выполнена проверка:

if (k = 0) **or** (k = 1) **then** f := 1

Понятно, что 4 не равно 0 и не равно 1, значит будет выполняться оператор после **else**, т.е. f:=4* f(k – 1), а это означает, что снова будет вызвана та же функция, но значение k уменьшится на единицу и станет равным 3; затем снова будет выполнена проверка условия:

if (k = 0) **or** (k = 1) **then** f := 1 и переход к вызову функции f(k - 1).

Значение **k** уменьшится на 1 и станет равным 2 и т. д. до тех пор, пока **k** не станет равным 1, а значение f получит значение 1 (надо заметить, что при всех предыдущих операциях значение f оставалось равным 0, а точнее говоря, неопределенным).

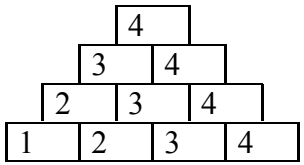
После этого, начнется обратный процесс, в котором будет выполняться команда f := f*k. Он будет происходить так:

f(1)=1;

f(2):= 2*f(1); → f =2;
f(3):= 3*f(2); → f := 3*2=6;
f(4):= 4*f(3); → f=4*6=24.

Образно говоря, при первоначальном процессе значения n от 4 до 1 "*запоминаются*" в стековую память "*Паскаль-машины*", а при следующем процессе значения n "*считываются*" из стековой памяти "*Паскаль-машины"* и умножаются на значения f.

Условно-схематически это можно изобразить так: значения n запоминаются в стек-память "*Паскаль-машины*", а затем начинают считываться в обратном порядке, начиная с единицы.



Обязательным элементом в описании всякого рекурсивного процесса является некоторое утверждение, определяющее условие завершения рекурсии; иногда его называют ***опорным условием рекурсии*** (или "***якорем***"). В нашем случае это условие

if (n = 0) **or** (n = 1) **then** f := 1.

В опорном условии может быть задано какое-то фиксированное значение, заведомо достигаемое в ходе рекурсивного вычисления и позволяющее организовать своевременную остановку процесса; применительно к вычислению факториала им будет равенство 1! = 1. Таким образом, любое рекурсивное определение всегда содержит два элемента: условие завершения и способ выражения одного шага решения посредством другого, более простого шага.

Для четкого понимания происходящих при этом процессов необходимо иметь в виду, что:

а) при каждом вызове процедуры создается новый экземпляр f;

б) все экземпляры f накапливаются во внутреннем стеке "*Паскаль-машины*";

в) в любой момент обработке доступен только один, хронологически последний экземпляр переменной f;

г) хронологически последний экземпляр переменной f по завершению очередного рекурсивного вызова автоматически уничтожается.

При каждом входе в рекурсивную подпрограмму ее локальные переменные размещаются в особым образом организованной области памяти, называемой программным стеком. *Об этом следует помнить, дабы избежать переполнения программного стека.*

СПИСОК ЛИТЕРАТУРЫ

1. Абрамов С.А., Зима Е.В. Начала информатики. - М.: Наука, 1989.
2. Абрамов В.Г., Трифонов Н.П., Трифонова Г.Н. Введение в язык Паскаль.-М.:Наука,1988.
3. Епанешников А., Епанешников В. Программирование в среде Turbo Pascal 7.0. - М.: ДИАЛОГ-МИФИ, 1993.
4. Зубов В. С. Программирование на языке Turbo Pascal.- М.:Филинъ, 1997г.
5. Зуев Е.А.. Программирование на языке Turbo Pascal 6.0,7.0.- М.:Радио и связь, Вестаю-1993.
6. Марченко А.И., Марченко Л.А. Программирование в среде Turbo Pascal 7.0. – К.: Век, 2000.
7. Фаронов В. В. Турбо Паскаль: (В 3 кн.). Книга 1. Основы Турбо Паскаля. - М.: Учебно-инженерный центр "МВТУ-ФЕСТО ДИДАКТИК", 1992.
8. Федоров А., Рогатин Д. Borland Pascal в среде Windows.-К.: Диалектика, 1993.
9. Федоров А. Особенности программирования в Borland Pascal.- К.:Диалектика,1994.
10. Джонс Ж., Харроу К. Решение задач в системе Турбо Паскаль/Пер. с англ.; Предисл. Ю.П. Широкого. - М.: Финансы и статистика, 1991.