Л.Е. Потапова, Т.Г. Алейникова

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПАСКАЛЬ

Учебно-методическое пособие

2008

УДК 004.43(075)
ББК 32.973.26-018.1я73
П64

Авторы: доцент кафедры информатики и информационных технологий УО «ВГУ им. П.М. Машерова», кандидат физико-математических наук **Л.Е. Потапова;** доцент кафедры инженерной физики УО «ВГУ им. П.М. Машерова», кандидат физико-математических наук **Т.Г. Алейникова**

Р е ц е н з е н т:
доцент кафедры прикладной математики и информатики УО «БГПУ им. М. Танка»,
кандидат физико-математических наук *А.И. Шербаф*

Учебно-методическое пособие содержит основной материал по программированию на языке Паскаль, необходимую справочную информацию по системе Turbo Pascal, вопросы и индивидуальные задания для лабораторных занятий и самостоятельной работы.

Данное издание ориентировано на обучение методам алгоритмизации и приемам программирования, которые проиллюстрированы большим количеством примеров с подробными комментариями.

Предназначено для студентов 1и 2 курса, дневной и заочной форм обучения специальности «Математика. Информатика» по дисциплине «Технологии программирования и методы алгоритмизации».

УДК 004.43(075)
ББК 32.973.26-018.1я73

С О Д Е Р Ж А Н И Е

В В Е Д Е Н И Е

Понятия алгоритма и программы являются базовыми в процедурном стиле программирования. Разработка алгоритма – это один из наиболее сложных этапов решения задачи с использованием ЭВМ. В начале обучения программированию, на наш взгляд, лучше использовать языки, специально разработанные для этой цели. Наиболее популярным и выдержавшим проверку временем является язык Паскаль. Его достоинства – простота, структурность, строгий синтаксис, дисциплина типов данных позволяют успешно применять этот язык для обучения алгоритмизации и хорошему стилю программирования. Хотя профессиональные программисты, как правило, используют другие языки программирования (C, C++, Java и др.), на наш взгляд, начинать обучение с них нецелесообразно из-за их сложности и нестрогости синтаксиса. Несмотря на то, что Никлаус Вирт создал Паскаль именно для целей обучения, он содержит все конструкции и средства, присущие современным языкам высокого уровня, что позволяет его использовать для разработки серьезных программ. Приобретя опыт алгоритмизации и программирования на Паскале, освоив структурную технологию разработки программ, студенту легче ориентироваться и совершенствоваться в других языках и технологиях программирования.

Учебно-методическое пособие содержит теоретический материал по основным типам данных, операторам и конструкциям языка Паскаль, вопросы и индивидуальные задания для лабораторных занятий и самостоятельной работы. В приложениях приводится необходимая справочная информация по операциям, отношениям, процедурам, функциям, типам языка, а также краткое описание среды программирования Turbo Pascal 7.0.

В учебном издании приводится большое количество примеров программ с подробными комментариями и объяснениями, направленными на обучение разработке алгоритмов и освоение приемов их программирования.

Учебно-методическое пособие написано на основе отработанной методики и лекционного материала, который использовался в процессе обучения студентов математического факультета УО «ВГУ им. П.М. Машерова» в течение ряда лет.

1. ЛИНЕЙНАЯ ПРОГРАММА. ВВОД И ВЫВОД ДАННЫХ

Данный раздел посвящен формированию представления о структуре программы на Паскале, об использовании стандартных функций и процедур, средствах ввода–вывода данных, знакомству с назначением и возможностями системы программирования TURBO PASCAL.

1.1. Структура программы на Паскале

Программы, написанные на языке программирования Турбо Паскаль, строятся в соответствии с правилами синтаксиса стандартного Паскаля. Программа, написанная на Паскале, состоит из заголовка и тела программы.

Формат программы:

```
Program <имя программы> [(Input, Output)];
   < Раздел описаний  >
Begin
   < Раздел операторов >
End.
```

Слово **Program** зарезервировано в Паскале и означает начало программы. Далее записывается <имя программы> – идентификатор программы, **[(Input, Output)]** – необязательный перечень файлов, через которые программа связана с внешней средой, используются по умолчанию.

В разделе объявлений и описаний программист сообщает компилятору о подключаемых модулях (программные объекты, описанные в другом месте), какими идентификаторами он обозначает данные (константы и переменные), определяет собственные типы данных, метки, описывает процедуры и функции. «Процедура» и «функция» – термины, применяемые в Паскале для обозначения специальным образом оформленной последовательности команд (подпрограммы). Доступ к такой подпрограмме может быть осуществлен из любого места основного блока программы, а также из любой процедуры или функции.

Раздел описаний ::= <описание модулей > | < описание меток > | < описание констант > | < описание типов > | < описание переменных > | < описание процедур и функций >.

Как правило, большую часть раздела описаний занимают

описания типов, констант и переменных. Типы величин, используемых в Турбо Паскале, приводятся в приложении 2.

Основной блок программы – это раздел операторов. Раздел операторов состоит из последовательности операторов, причем работа программы начинается именно с первого оператора основного блока программы. Тело основного блока программы ограничено словами **Begin** и **End**.

Пример 1.

```
Program prog1_1;
   < Раздел описаний  >
Begin
   < Раздел операторов >
End.
```

Разделитель **;** отмечает конец оператора или описания. Использование разделителя позволяет располагать несколько операторов на одной строке.

После последнего оператора **end** всегда ставится точка, тем самым компилятор получает информацию об окончании текста программы.

1.2. Процедуры ввода

Для ввода данных в языке Паскаль предусмотрены стандартные встроенные процедуры – `read` и `readln`. Оператор ввода служит для ввода данных в процессе выполнения программы. Процедуры `read` и `readln` используются в виде:

а) `read <список ввода>;` – каждое вводимое значение присваивается последовательно переменным из списка. Следующий оператор ввода будет вводить данные с той же строки.

б) `readln <список ввода>;` – отличается от `read` тем, что следующий оператор ввода будет вводить данные с новой строки.

в) `readln;` – переход на новую строку при вводе данных.

При вводе числовые данные должны разделяться пробелом или символом окончания ввода (клавиша Enter).

Логические данные в Паскале вводить не разрешается.

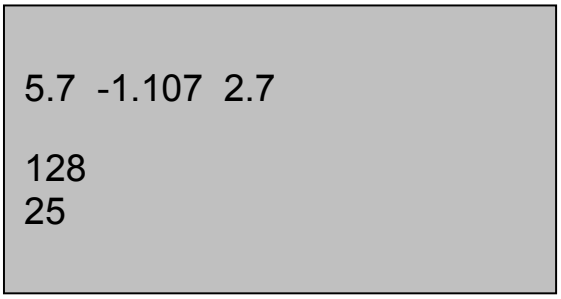
Пример 2.

```
Program prog1_2;
  var a, b, c: real; к, t : integer;
Begin
  read(a, b, c);
  readln; read(k, t);
```

```
End.
```

Первая процедура `read(a, b, c)` вводит с клавиатуры три вещественных числа, следующая `readln` приостанавливает работу программы до нажатия любой клавиши. Процедура `read(k, t)` вводит два целых числа.

Экран пользователя в примере 2 может, например, принять вид:



В данном примере переменные получат значения: `a = 5.7; b = -1.107; c = 2.7; k = 128; t = 25`.

1.3. Процедуры вывода

Для вывода результатов в Турбо Паскале предусмотрены две процедуры: `write(<список вывода>)` и `writeln(<список вывода>)`.

Элементы списка вывода (константы, переменные, выражения) разделяются запятыми. Оператор `write` выводит указанные в списке величины на экран и оставляет курсор в конце только что выведенной строки. Оператор `writeln` после вывода устанавливает курсор в начало следующей строки. Элементы списка вывода выводятся подряд без пробелов между ними.

Действительные числа выводятся в экспоненциальной форме, т.е. в виде: #.# # # # … #E±# # # #. Количество знаков после десятичной точки зависит от конкретного типа вещественного числа.

Операторы вывода допускают использование в явном виде указаний о ширине поля, отводимого под значение выводимой величины. Форма представления выводимых переменных определяется типом переменных. Величины целого типа выводятся в обычной форме или в формате `I:p`, где `p` – целое число, указывающее количество позиций экрана, отводимых под запись числа. При выводе значений действительных типов с фиксированной точкой указывается ширина поля, отводимая под

все значение и под дробную часть, т.е. в формате `R:p:q`. Каждое значение выводимых величин занимает крайние правые позиции отведенного поля, при этом незаполненные остаются свободными, образуя пробелы. Если при выводе действительных значений не указывается количество позиций, отведенных под дробную часть числа, то результат получается в экспоненциальном виде с десятичным порядком.

Пример 3.

```
Program prog1_3;
  var a, b, c : real; k, t : integer;
Begin
  read (a, b, c); readln; read(k, t);
  write(a:8:3); writeln(b:10:4,c:10:4);
  writeln(k,t:5);
End.
```

Экран пользователя в примере 3 примет вид:

```
5.7  -1.107  2.7

128
25

  5.700   -1.1070    2.700
 128   25
```

Значения переменных `a, b` и `c` размещаются в одной строке и занимают `a` – восемь, `b` и `c` – десять позиций. Под дробную часть отводится 3, 4, 4 позиции соответственно. Значения `k, t` выводятся на следующей строке, оба в поле по пять позиций.

1.4. Стандартные процедуры и функции

В Турбо Паскале есть возможность использования некоторых стандартных процедур и функций без предваряющего их описания (общее правило гласит: все процедуры должны быть описаны) (см. приложение 4).

Вызов функций может присутствовать в выражениях. Функция передает в выражение свое значение.

Правила записи стандартных функций:

1. Имя функции записывается буквами латинского алфавита.

2. Аргумент функции записывается в круглых скобках после имени функции.

3. Аргументом функции может быть константа, переменная или выражение.

Пример 4. Вычислить выражения:

Результаты вывести на печать в форматах: R, R:p, R:p:q, где R – вещественное число, p и q – параметры формата. Перед каждым результатом указать формат вывода (R, R:p или R:p:q).

```
Program prog1_4;
  var a, b, c, y, z: real; k, t : integer;
Begin
  read(a, b, c); readln; read(k, t);
  write(a:8:3); writeln(b:10:4, c:10:4);
  writeln(k, t:5);
  {расчет по формулам}
  y:=sqr(k)+abs(ln(a))/(sqr(y)+1);
  z:=sin(2*c+b);
  {вывод на экран}
  writeln('R      ',y,',',z);
  writeln('R:p    ',y:10,',',z:10);
  writeln('R:p:q',y:10:5,',',b:10:4);
  readln; {для просмотра результата:  пока не будет нажат ВВОД,
  результат будет на экране}
End.
```

Контрольные вопросы:

1. Каковы составные части программы, записанной на языке Pascal? Как они оформляются?
2. Какие типы числовых переменных имеются? В каком разделе и как они описываются?
3. Каковы основные правила использования стандартных функций?
4. Чем отличаются процедуры `read` и `readln`?
5. Как осуществляется ввод данных с клавиатуры?
6. Как обеспечить вывод результатов на экран дисплея и на принтер?
7. Для чего предназначен и как используется форматный вывод данных?
8. Основные приемы работы в системе программирования TURBO PASCAL:
   - Как сохранить программу в файле с заданным именем?
   - Как загрузить программу?
   - Как осуществить набор новой программы?

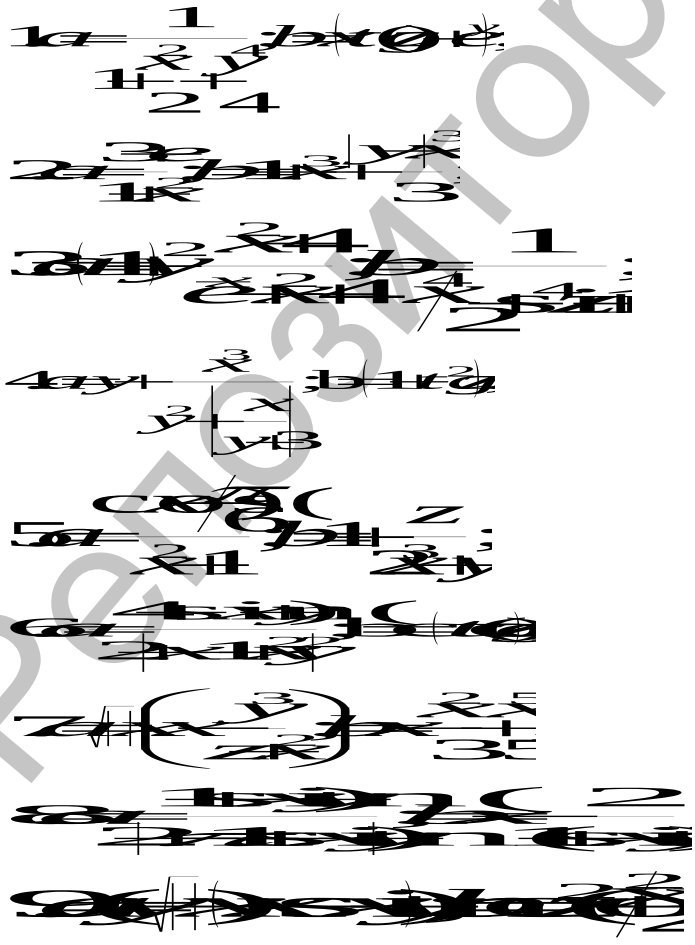
- Как откомпилировать и выполнить программу?
- Как просмотреть результаты выполнения программы?
- Как получить распечатку текста программы?

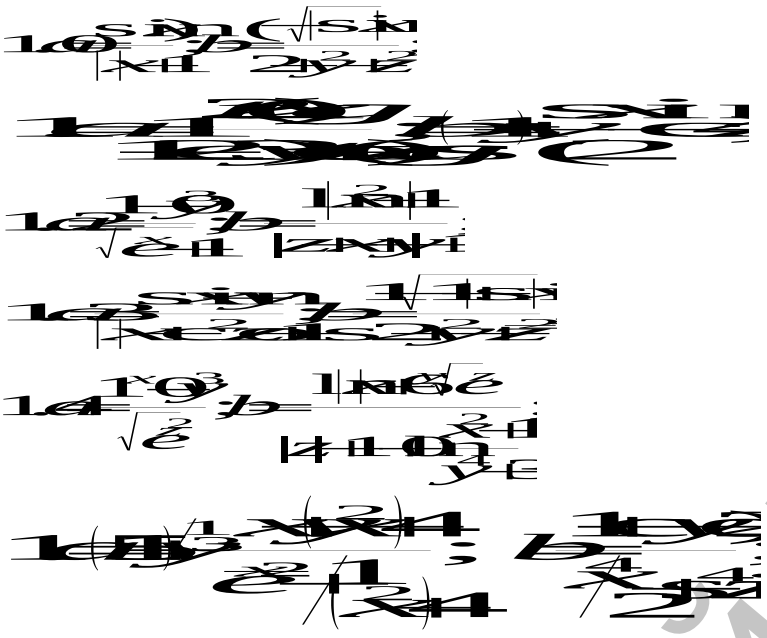
Лабораторная работа № 1

Задание 1. Запустите интегрированную среду Турбо Паскаль. Загрузите файл с готовой программой (указывается преподавателем). Сохраните программу с новым именем в вашей папке. Ознакомьтесь с текстом программы. Подумайте, каков должен быть результат ее выполнения. Выполните программу (Run) и познакомьтесь с результатами ([Alt]+[F5]).

Задание 2. Ввести вещественные числа x, y, z из области допустимых значений исходных данных. Вычислить a, b. Результаты вывести на печать в форматах: R, R:p, R:p:q, где R – вещественное число, p и q – параметры формата (задать самостоятельно). Перед каждым результатом указать формат вывода (R, R:p или R:p:q).

Варианты заданий:





Задание 3. Составьте программу для решения следующей задачи, используя операции `div`, `mod`, а также стандартные функции (приложение 4).

Варианты заданий:

1. С начала месяца прошло *m* часов. Определить, какое сейчас число.
2. С начала года прошло *k* дней. Определить, какая сейчас неделя, если 1 января было в понедельник.
3. С начала суток прошло *k* минут. Определить, который сейчас час (в часах и минутах).
4. Идет *k*-ая секунда суток. Определить, который сейчас час (в часах, минутах и секундах).
5. С начала месяца прошло *t* минут. Определить какое сейчас число и который сейчас час (в часах и минутах).
6. Пусть *k* целое от 1 до 365. Присвоить переменной n значение 0, 1, 2,..., 6 в зависимости от того, на какой день недели попадает *k*-ый день невисокосного года, в котором 1 января – понедельник (1 – понедельник, 2 – вторник,..., 0 – воскресенье);
7. Преобразовать произвольное вещественное число путем округления его до десятых.
8. Определить *f* – угол (в градусах) между положением часовой стрелки в начале суток и ее положением в *h* часов *t* минут ($0 \le h < 12$, $0 \le t < 60$).
9. По году вашего рождения определить год выхода на пенсию. (У мужчин и женщин программы должны отличаться.)
10. Найти произведение цифр трехзначного числа.
11. Определить полное количество часов и минут,

прошедших от начала суток до того момента (в первой половине суток), когда часовая стрелка повернулась на f градусов ($0 < f < 360$, f – целое число).

12. Преобразовать произвольное действительное число путем округления его дробной части.

13. Владелец коммерческого магазина всю мелочь откладывает на благотворительные цели. Написать программу, которая запрашивает стоимость проданного товара и печатает сумму, которая достается владельцу магазина, а также сумму мелочи для благотворительных нужд.

14. Если сумма налога исчисляется в рублях и копейках, то налоговая служба округляет ее до ближайшего рубля (до 50 копеек – с недостатком, свыше 50 копеек – с избытком). Требуется использовать компьютер, чтобы ввести точную сумму налога и напечатать, сколько следует уплатить.

15. Алеша, Дима и Слава Зайцевы каждую неделю получают некоторые суммы денег P, V и K от своих родителей. Ежедневно в течение недели каждый из мальчиков тратит одну и ту же максимально возможную целую сумму денег, а образовавшийся остаток вносит в общую копилку. Выяснить, какую сумму денег потратил каждый из ребят на прошедшей неделе, и сколько денег было внесено ребятами в копилку.

2. ПРОГРАММЫ С ВЕТВЛЕНИЯМИ

Цель: знакомство со структурными операторами Паскаля, формирование умения выбора необходимых конструкций при реализации разветвляющихся алгоритмов.

2.1. Условный оператор

Условный оператор позволяет проверить некоторое условие и в зависимости от результата выполнить то или иное действие. С помощью этого оператора программируются алгоритмы разветвляющейся структуры. Структура условного оператора:

if <условие> **then** <оператор 1> **else** <оператор 2>,

где **if**, **then**, **else** – зарезервированные слова (если, то, иначе); <условие> – произвольное выражение логического типа; <оператор 1>,<оператор 2> – любые операторы языка.

Условный оператор работает по следующему алгоритму. Вначале вычисляется условное выражение <условие>. Если результат есть TRUE (истина), то выполняется <оператор 1>, а <оператор 2> пропускается; если результат есть FALSE (ложь), наоборот, <оператор 1> пропускается, а выполняется <оператор 2>. Перед **else** никогда не ставится точка с запятой.

Пример 1. Найти максимальное значение из двух чисел.

```
Program prog2_1;
  var x, y, max: real;
Begin
  read (x, y);
  if x > y then max:= x else max:= y;
  writeln(max:10:4);
End.
```

Если выполняется условие x > y, то max := x, иначе max := y.

Пример 2. Найти максимальное значение из двух чисел. Предусмотреть вариант, когда числа равны.

```
Program prog2_2;
  var x, y, max: real;
Begin
```

```
  read (x, y);
  if x > y then begin max:= x;
  writeln(max:10:4) end
           else if x < y then begin max:= y;
           writeln(max:10:4) end
                else writeln('числа равны');
End.
```

Если удовлетворено условие x > y, то переменная max получит значение x, и это значение будет выведено. Если условие x > y не выполняется, то второй условный оператор (после первого **else**) позволяет найти max или установить равенство чисел.

Часть **else** <оператор 2> условного оператора может быть опущена. Тогда при значении TRUE условного выражения выполняется <оператор 1>, в противном случае этот оператор пропускается.

Пример 3. Программа вычисляет абсолютное значение переменной x.

```
Program prog2_3;
  var x: real;
Begin
  read (x);
  if x < 0 then x := -x;
  writeln(x:8:2);
End.
```

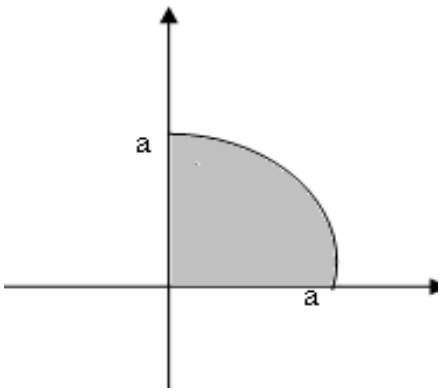
Пример 4. Вычислить абсолютные значения чисел, меньших – 1, а отрицательные числа, большие – 1, заменить их квадратами.

```
Program prog2_4;
  var x: real;
Begin
  read (x);
  if x < 0 then if x < -1 then x := -x
                          else x:=sqr(x);
  writeln(x:8:2);
End.
```

Если **else** отсутствует, а после **then** вновь стоит условный оператор **if**, то возникает неоднозначность трактовки условий. Эта неоднозначность решается следующим образом: любая встретившаяся часть **else** соответствует ближайшей к ней части **then** условного оператора. Приведенное выше выражение понимается так:

```
if <выражение 1> then
begin
  if <выражение 2> then <оператор 1>
                   else <оператор 2>
end.
```

Пример 5. Если точка с координатами, введенными с клавиатуры, попадает в закрашенную область на координатной плоскости, найти расстояние этой точки от начала координат.



Чтобы точки попадали в закрашенную область, значения их координат должны лежать в диапазоне: x >= 0, y >= 0, и удовлетворять условию $x^2 + y^2 <= a^2$. В данном случае можно использовать три вложенных оператора **if.** Однако проще записать один оператор со сложным условием. Так как все три условия должны выполняться одновременно, то для связи условий необходимо использовать логическую операцию **and**.

```
Program prog2_5;
  var x, y, a, r: real;
Begin
  read (x, y, a);
  if (x > 0) and (y>0) and (x*x+y*y <= sqr(a))
  then begin r:= sqrt(sqr(x)+sqr(y));
  writeln('r= ', r:10:4); end
  else writeln('точка вне области');
End.
```

В случае выполнения всех трех условий, будет выведено значение r. Если хотя бы одно из условий не выполняется, будет выдано сообщение ‘точка вне области’.

2.2. Оператор выбора

В Паскале существует еще один тип условного оператора. Оператор выбора или переключатель позволяет выбрать одно из нескольких возможных вариантов решения задачи. Структура оператора выбора:

```
case <выражение - селектор> of
<список констант 1> : <оператор 1>;
```

```
  <список констант 2> : <оператор 2>;
  ……..
  <список констант k> : <оператор k>;
  else <оператор > end;
```

Выражение-селектор имеет любой скалярный тип, кроме вещественного. Список констант – одна или несколько констант того же типа, что и выражение-селектор, записанные через запятую.

Оператор **case** работает следующим образом:

1. Вычисляется значение выражения-селектора.
2. Найденное значение сравнивается со списками констант.
3. Если значение совпадает с одной из констант, то выполняется соответствующий ей оператор, а затем – выход из оператора **case**.
4. Если значение выражения-селектора ни с одной из констант не совпадает, то выполняется оператор, следующий за словом **else**.

Часть **else** <оператор > может отсутствовать. Тогда при отсутствии в списках констант нужной константы, оператор выбора просто завершит свою работу.

Пример 6. С клавиатуры вводится номер месяца, выдать сообщение о соответствующем времени года.

```
Program prog2_6_1;
  var n : byte;
Begin
  read (n);
  case n of
    1, 2, 12  : writeln('зима');
    3, 4, 5   : writeln('весна');
    6, 7, 8   : writeln('лето');
    9, 10, 11 : writeln('осень');
  else writeln('неверный номер месяца');
  end;
End.
```

Если задать более точно тип месяца, то можно опустить **else** writeln('неверный номер месяца'):

```
Program prog2_6_2;
  var n : 1..12;
Begin
  read (n);
  case n of
```

```
    1, 2, 12  : writeln('зима');
    3, 4, 5   : writeln('весна');
    6, 7, 8   : writeln('лето');
    9, 10, 11 : writeln('осень');
  end;
End.
```

Контрольные вопросы:

1. Выражением какого типа является условие в операторе **if**? Какие значения оно может принимать?
2. Как выполняются действия в операторе **if** в зависимости от значения условия?
3. Как работает оператор **if**, если отсутствует часть **else** <оператор 2>?
4. В каких случаях используется оператор **case**?
5. Какого типа может быть выражение-селектор в операторе **case**?
6. Что представляет собой <список констант>?
7. Как выполняются действия в операторе **case**?

Лабораторная работа № 2

Задание 1. Составьте программу для решения следующей задачи:

Варианты заданий:

1. Найти решения уравнения $ax + b = 0$ для любых значений параметров a и b.

2. Дана окружность радиуса R с центром в начале координат. Определить, лежит ли точка с координатами (X, Y) внутри окружности, на окружности или вне окружности.

3. Даны три вещественные переменные A, B, C. Определить среднее геометрическое этих переменных, если все они отличны от нуля, и их среднее арифметическое в противном случае.

4. Посчитать количество четных среди чисел a, b, c. Нечетные числа заменить 0.

5. Поменять местами значение целых переменных A, B, C таким образом, чтобы оказалось $A>=B>=C$.

6. Определить, является ли треугольник со сторонами a, b, c равнобедренным. Если «да», то вывести, что является основанием, а что боковыми сторонами.

7. Определить, лежат ли в одной координатной четверти две точки с координатами (A_1, B_1) и (A_2, B_2).

8. Даны действительные положительные числа *x, y, z*. Выяснить, существует ли треугольник с длинами сторон *x, y, z*.

9. Дан номер года. Определить, является ли он високосным (год является високосным, если его номер делится на 4, за исключением тех, которые делятся на 100 и не делятся на 400).

10. По номеру Y (Y > 0) некоторого года определить C – номер его столетия (учесть, что, к примеру, началом XX столетия был 1901, а не 1900 год).

11. Вводится время дня и, в зависимости от введенного значения, печатается пожелание доброго утра, доброго дня, доброго вечера или спокойной ночи. Программа должна реагировать на ввод неправильного времени: меньше 0 или больше 24.

12. Определить, попадает ли точка *M(x,y)* в круг радиусом *r* с центром в точке (x_0,y_0). Если точка *M* принадлежит внутренней части круга, то найти расстояние от точки до центра круга.

13. Даны действительные числа *a, b, c, x, y*. Выяснить, пройдет ли кирпич с ребрами *a, b, c* в прямоугольное отверстие со сторонами *x* и *y*. Просовывать кирпич в отверстие разрешается только так, чтобы каждое из его ребер было параллельно или перпендикулярно каждой из сторон отверстия.

14. Выяснить, в какой координатной четверти расположен треугольник, образованный прямой, заданной уравнением $y = ax + b$ и осями координат.

15. Дана тройка чисел *a, b, c*. Проверить, могут ли они быть датой (например, 3.7.2005 – дата).

Задание 2. Используя оператор выбора, составьте программу для решения следующей задачи:

Варианты заданий:

1. Вводится число от 1 до 10. Дать название этого числа (1 – один, 2 – два, ..., 10 – десять).

2. Вводится целое число. Определить, является ли оно цифрой и какой: нулем, четной или нечетной.

3. Дано натуральное число *N* (< 20), определяющее сумму денег в рублях. Вывести для этого числа наименование: «рубль», «рубля», «рублей».

4. Дано натуральное число *N* (*N* <100), определяющее возраст человека в годах. Вывести для этого числа наименование: «год», «года», «лет».

5. Вводится число от 1 до 20. Вывести название этого числа (1 – один, 2 – два, ..., 20 – двадцать).

6. Вводится число от 20 до 30. Вывести название этого числа (21 – двадцать один, 22 – двадцать два, ...).

7. Вводится число от 1 до 100. Вывести название этого числа (1 – один, 2 – два, ..., 100 – сто).

8. После введенного с клавиатуры числа (1–99) дописывается слово «копейка» в правильной форме (пример 12 копеек, 1 копейка).

9. Запрашивается номер недели и затем выводится название по номеру (например, 2 -> Вторник).

10. Вводится с клавиатуры число *X*, имеющее смысл месяца, и выводится пора года («Зима», «Весна», «Лето» или «Осень») в зависимости от введенного месяца. Программа должна реагировать на ввод неправильного месяца: меньше 1 или больше 12.

11. Вводятся с клавиатуры вещественные числа *X*, *Y* и символ *K$*. В зависимости от значения *K$* (+, -, * или /) вычислить соответственно сумму чисел *X* и *Y*, их разность, произведение или частное. Результат вывести на экран.

12. По выбору пользователя произвести расчет площади одной из следующих фигур: прямоугольника, треугольника, круга.

13. Вводится номер одного из месяцев, а выводится количество дней в этом месяце.

14. Известно, что астрологи делят год на 12 периодов и каждому из них ставят в соответствие один из знаков Зодиака:

20.01 – 18.02 Водолей	23.07 – 22.08 Лев
19.02 – 20.03 Рыба	23.08 – 22.09 Дева
21.03 – 19.04 Овен	23.09 – 22.10 Весы
20.04 – 20.05 Телец	23.10 – 22.11 Скорпион
21.05 – 21.06 Близнецы	23.11 – 21.12 Стрелец
22.06 – 22.07 Рак	22.12 – 19.01 Козерог

По введенной дате (день и месяц) определить соответствующий знак Зодиака.

15. В старояпонском календаре был принят 60-летний цикл, состоящий из пяти 12-летних подциклов. Подциклы обозначались названиями цвета: зеленый, красный, желтый, белый и черный. Внутри каждого подцикла годы носили названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. Начало очередного цикла было в 1984 году – году зеленой крысы. По введенному номеру некоторого года нашей эры определите его

название по старояпонскому календарю.

3. ЦИКЛИЧЕСКИЕ ПРОГРАММЫ

Данный раздел посвящен формированию умений выбора необходимых конструкций при реализации циклических алгоритмов.

3.1. Операторы повторений

Операторы повтора предусматривают выполнение тела цикла – простого или составного оператора – несколько раз. В языке Паскаль имеются три различных оператора, с помощью которых можно запрограммировать повторяющиеся фрагменты программ: цикл с параметром, цикл с предусловием и постусловием. Если число повторений в циклическом алгоритме известно заранее (до начала повторений), то в такой ситуации лучше воспользоваться оператором цикла с параметром. В других случаях следует использовать операторы цикла с предусловием и постусловием.

3.1.1. Оператор цикла с предусловием

На языке Pascal структура цикла с предусловием («цикл-пока») записывается следующим образом:

WHILE <условие> **DO** <оператор>;

WHILE, DO – зарезервированные слова (пока выполняется условие, делать);

<условие> – выражение логического типа;

<оператор> – произвольный оператор.

Если выражение <условие> имеет значение true, то выполняется <оператор>, после чего вычисление выражения <условие> и его проверка повторяются. Если <условие> имеет значение false, оператор **while** прекращает свою работу.

Пример 1**.** Найти сумму ряда с точностью $\varepsilon = 10^{-4}$, общий член которого:

$$a_n = \frac{1}{n}$$

Слагаемые, по модулю меньшие □ , не учитывать.

```
Program prog3_1;
  const e:=0.0001;
  var s,a:real; i : integer;     {s – сумма, a – слагаемое}

Begin
```

```
  s:=0;    i:=1;    a:=1/i; {задание начальных данных}
  {подсчет суммы в цикле while}
  while a>=e do begin
    s:=s+a;    i:=i+1;    a:=1/i;
  end;
  writeln('Ответ(while):', s:8:4); {вывод результата
  на экран}
End.
```

Перед входом в цикл задаем начальные значения исходных данных. Цикл будет выполняться, пока слагаемое будет больше заданного ε. Возможна ситуация, когда заданное ε больше первого слагаемого. В этом случае оператор цикла не выполнится ни разу. Значение суммы s будет равно 0.

3.1.2. Оператор цикла с постусловием

Этот вид цикла отличается от предыдущего в основном тем, что проверка условия повторения тела цикла находится не перед ним, а после. Поэтому цикл с постусловием называют «циклом-до».

Формат «цикла-до» на языке Pascal:

```
REPEAT <тело цикла> UNTIL <условие>;
```

REPEAT, UNTIL – зарезервированные слова (повторять до тех пор пока не будет выполнено условие);

<тело цикла> – произвольная последовательность операторов;

<условие> – выражение логического типа.

Оператор REPEAT будет выполняться, пока <условие> ложно.

Здесь не требуется использование составного оператора, потому, что сами слова Repeat и Until являются операторными скобками.

Поскольку условие выполнения тела цикла стоит в конце, этот цикл будет выполнен хотя бы один раз, даже если условие цикла изначально истинно. Именно это отличие «цикла-до» от «цикла-пока» привело к тому, что в программировании они не подменяют друг друга, а используются для решения задач, к которым они более подходят.

Пример 2. Напишем программу для примера 1 с использованием оператора цикла с постусловием.

```
Program prog3_2;
  const e:=0.0001;
  var s, a : real; i : integer;
Begin
```

```
  s:=0;   i:=0; a:=0;          {задание начальных данных}
  repeat                       {подсчет суммы в цикле repeat }
    s:=s+a;   i:=i+1;   a:=1/i;
  until a<e;
  writeln('Ответ(repeat):', s:8:4); {вывод результата
  на экран}
End.
```

Обратите внимание, что начальные значения `i` и `a` равны 0. Это изменение необходимо, чтобы результаты цикла **`while`** и **`repeat`** были одинаковы для всех исходных данных. В «цикле-пока» значение суммы `s` равно `0`, если e больше первого слагаемого. В «цикле-до» первое слагаемое прибавляется к сумме до проверки условия. Поэтому если оставить `i:=1; a:=1/i`, то сумма будет равна `1/i`, вместо 0.

В рассмотренном примере правильный результат получен как в случае использования цикла **`while`**, так и цикла **`repeat`**.

Пример 3. Дано целое число M. Требуется найти наименьшее целое неотрицательное число k, при котором y = 3k >M.

Эту задачу можно решить по следующему алгоритму: предварительно задать `y=1` и `k=0`. Затем в цикле домножать значение y на 3, и увеличивать значение k на 1 до тех пор, пока текущее значение не окажется больше значения `M`.

```
y:=1;   k:=0;
repeat  y:=y*3;   k:=k+1;  until y >M;
```

Однако при `y=1` будет получен неправильный результат `k=1`, тогда как правильное значение `k=0`. Это происходит из-за того, что `y` умножается на 3 в любом случае хотя бы раз. Оператор **`while`** позволяет избежать этой ошибки.

```
y:=1;   k:=0;
while y <=M do begin  y:=y*3;  k:=k+1;
end;
```

Если `M < 1`, то цикл не выполнится ни разу и результат будет верным: `k=0`.

3.1.3. Оператор цикла с параметром

У этого вида цикла предусмотрено два формата:

`FOR` <парам. цикла>:=<нач. знач.> **`TO`**<кон. знач.> **`DO`** <оператор>;

`FOR` <парам. цикла> := <нач. знач.> **`DOWNTO`** <кон. знач.> **`DO`** <оператор>;

FOR, TO, DOWNTO, DO – зарезервированные слова (для, до, выполнить).

<парам. цикла> – переменная скалярного типа;

<нач. знач.> и <кон. знач.> – выражения, определяющие начальное и конечное значение параметра;

<оператор> – произвольный оператор.

При выполнении оператора **FOR** вначале вычисляется выражение, задающее значение параметру цикла, затем это значение присваивается параметру цикла. После этого следует:

1) проверка условия <парам. цикла> <= <конеч. знач.> (<парам. цикла> >= <конеч. знач.>*);

если условие выполняется, то перейти к пункту 2, иначе к пункту 4;

2) выполнение оператора <оператор>;

3) изменение переменной <парам. цикл> на единицу (-1*) и переход к пункту 1;

4) завершение работы.

Пример 4.

```
i : integer;
```

оператор	*результат*
`for i:= 10 to 14 do write(i:3);`	**10 11 12 13 14**
`for i:= 14 to 10 downto write(i:3);`	**14 13 12 11 10**

В данном случае параметром будет являться целочисленная переменная, которая будет изменяться на 1 (-1*) при каждой итерации цикла. Таким образом, задав начальное и конечное значения для такой переменной, можно точно установить количество выполнений тела цикла.

Пример 5.

```
ch : char;
```

оператор	*результат*
`for ch:= 'a' to 'e' do write(ch:2);`	**a b c d e**
`for ch:= 'e' to 'a' downto write(ch:3);`	**e d c b a**

В первом случае параметр, который является символьной переменной, с каждой итерацией получает следующее значение в списке типа, во втором – предыдущее.

---

* Для варианта **DOWNTO**

Таким образом, в отличие от первых двух видов цикла, этот цикл используется тогда, когда известно необходимое количество выполнений тела цикла.

Вообще говоря, цикл «Пока» является универсальным, то есть любая задача, требующая использования цикла, может быть решена с применением этой структуры. Циклы «До» и «С параметром» созданы для удобства программирования.

Пример 6. Найти сумму ряда k слагаемых гармонического ряда: $a_n = 1/n$ (сравнить с примером 1).

```
Program prog3_6;
  var s : real;  k : integer; {s – сумма, k – количество
  слагаемых}
Begin
  s:=0; {задание начальных данных}
  for i:= 1 to k do begin
  s:=s + 1/i;                    {подсчет суммы в цикле for}
  end;
  writeln('Ответ(while):', s:8:4);  {вывод результата
  на экран}
End.
```

3.2. Примеры использования циклов

Пример 7**.**

Найти p – количество трехзначных натуральных чисел, сумма цифр которых равна S (1<S<27)**.** Операции деления (/ , div, mod) не использовать.

```
Program prog3_7;
  uses CRT; {подключение модуля для работы с экраном}
  var S, p, ns, nd, ne:word; {ns – сотни, nd – десятки,
  ne – единицы}
Begin
  CLRSCR; {очистка экрана}
  write('Введите число S ');
  readln (S);
  p:=0;
  for ns:=1 to 9 do       {цикл для задания сотен}
    for nd:=0 to 9 do       {цикл для задания десятков}
      for ne:=1 to 9 do     {цикл для задания единиц}
        {проверка заданного условия}
        if ns+nd+ne=S then p:=p+1;
  writeln('всего p ',p:3);
End.
```

Пример 8.

Вычислить сумму цифр заданного натурального числа.

```
Program prog3_8;
  USES CRT;
  var  i,k,f,n,s:integer;
Begin
CLRSCR;
                          {«цикл-пока»}
  write('Введите число n ');
  readln (n);
  s:=0;
  while n>0 do begin          {пока n>0 :делай}
    k:=n mod 10;              {остаток от деления на 10}
    n:=n div 10;              {целая часть, от деления на 10}
    s:=s+k                    {сумма цифр числа n}
  end;
  writeln ('s=',s:7);
                          {«цикл-до»}
  write('Введите число n ');
  readln (n);
  s:=0;
  repeat                      {начало цикла}
    k:=n mod 10;              {остаток от деления на 10}
    n:=n div 10;              {целая часть, от деления на 10}
    s:=s+k                    {сумма цифр числа n}
  until n=0;    {пока условие имеет значение false выполняем}
  writeln ('s=',s:7);
End.
```

Пример 9. Найти сумму членов последовательности:

$$q=(-1)^i\frac{x^{2i+1}}{(2i+1)!},\quad (i = 0, 2, \ldots), \text{ где } |x|<1,$$

с точностью ε.

Если не задумываясь программировать задачу, то можно на каждом шаге цикла возводить `x` в степень, используя функции `ln` и `exp`, а для вычисления $(-1)^i$ и знака числителя использовать оператор `if`. Однако это совершенно неэффективное решение, так как функции `ln` и `exp` сами вычисляются разложением в степенные ряды. Поэтому при решении таких задач пытаются найти связь между предыдущим и следующим членами последовательности (рекуррентное соотношение). Каждый следующий член последовательности будет получаться умножением предыдущего на найденный множитель. В нашем

случае a_i член связан с a_{i-1} соотношением:

Множитель, с помощью которого связаны два соседних члена последовательности равен $\frac{(2i-1)x^2}{2i+1}$.

```
Program prog3_9;
  const e=0.0001;     {значение ε }
  var i:integer; a,x,s:real;
Begin
  write('введите n и x');
  readln(n, x);              {ввод начальных данных}
                {цикл-пока}
  s:=0; i:=1;
  a:=x;      {значение элемента последовательности для n=0: a0}
  while a>=e do begin       { пока ai>= e:делай}
    s:=s+a;          {добавление к сумме очередного слагаемого}
    a:=-a*(2*i-1)*sqr(x)/(2*i+1);{вычисление
    очередного члена}
    i:=i+1;                    {увеличение номера слагаемого}
  end;
  writeln ('s=',s:7);
                {цикл-до}
  write('введите n и x');
  readln(n, x);              {ввод начальных данных}
  a:= 1/x;    {сдвиг последовательности влево до a-1, чтобы
              решение давало значение s=0 для a0<e}
  s:=-a; i:=0;
  repeat                     {начало цикла}
    s:=s+a;   {добавление очередного слагаемого: s=0 на первом
              шаге цикла}
    a:= -a*(2*i -1)*sqr(x)/(2*i+1);
    {вычисление очередного члена, начиная с a0}
    i:=i+1;     {увеличение номера слагаемого}
  until a<e;      {пока условие имеет значение false выполняем}
  writeln ('s=',s:7);
  writeln ('s=',s:7);
End.
```

Пример 10**.** Дано натуральное число a. Найти его разложение на простые множители. Множители вывести на экран.

```
Program prog3_10;
  USES CRT;
```

```
  var  a,k,p:word;
Begin
  CLRSCR;
  write('Введите число a ');
  readln (a);          {ввод заданного числа}
  K:=0;
  p:=2;                {наименьшее простое число}
  repeat               {начало цикла}
    k:=0;
    while a mod p=0 do begin {пока a делится на p}
      a:=a div p;k:=k+1 {делим a на p и считаем степень k
      делителя p}
    end;
    if k<>0 then writeln('число',p:3,'
    степень',k:3); {вывод делителя p и степени k}
    if p=2 then p:=p+1 else p:=p+2; {изменяем
    значение p}
  until a=1;   {пока условие имеет значение false выполняем}
End.
```

Контрольные вопросы:

1. Как программируются циклические алгоритмы с явно заданным числом повторений цикла?
2. Как программируются циклические алгоритмы с незаданным числом повторений цикла?
3. В чем отличие операторов while и repeat?
4. Можно ли изменять программным путем параметр цикла в операторе for?
5. Можно ли использовать оператор for для программирования цикла с заданным числом повторений и шагом переменной отличным от 1 и -1?
6. Как избежать «зацикливания»? Как прервать программу при зацикливании?

Лабораторная работа № 3

Задание 1. Составьте программу для решения следующей задачи, используя цикл **`for`**:

Варианты заданий:

1. Дано натуральное число *n*. Определить, сколько в числе нулей.
2. Найти наименьшее четырехзначное число, куб суммы цифр которого равен ему самому.
3. Найти в диапазоне *[a, b]* первое число, кратное *c* и *d*.
4. Найти количество отрицательных значений функции $y = x^2 - a*b+c*x$ на отрезке от x_1 до x_2 с шагом *h*. Параметры *a, b, c* заданы.
5. Найти наибольшее значение функции $y = ax^3+bx+c$ при заданных параметрах *a, b, c*, если *x* изменяется от x_1 до x_2 с шагом *h*.
6. Дано *n*-значное целое число, записать его цифры в обратном порядке.
7. Вычислить: [illegible].
8. Вычислить при данном *n* и действительном *x*

$$\underbrace{\sqrt{x+\sqrt{x+\ldots+\sqrt{x}}}}_{n}.$$

9. Дано натуральное число *n*. Вычислить произведение первых *n* сомножителей: $\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot\ldots$.
10. Для действительного числа *a* и натурального числа *n* найти: $a^{n-1}/n!$.
11. Даны вещественное число *a* и натуральное число *n*, вычислить *((...((1+a)+a)+...+a)+a)*.
12. Вычислить: [illegible].

Число *m* ввести с клавиатуры.

13. Выдать на печать таблицу значений функции:

$y=\sin\sqrt{x}$ для $x\in[0;2\pi]$ с шагом $h=\frac{\pi}{6}$.

14. Пусть $x_1=x_2=x_3=1$; $x_i=x_{i-1}+x_{i-2}+x_{i-3}$ ($i=4,5,\ldots$). Дано натуральное *n*. Найти сумму: $\frac{x_1}{2}+\frac{x_2}{2^2}+\ldots+\frac{x_n}{2^n}$.

15. Вычислить π по формуле Валлиса, взяв в разложении *n* сомножителей:

$$\frac{\pi}{2}=\frac{2\cdot2\cdot4\cdot4\cdot6\cdot6\cdot\ldots}{1\cdot3\cdot3\cdot5\cdot5\cdot7\cdot\ldots}.$$

Задание 2. Составьте программу для решения следующей

задачи, используя цикл **`while`** или **`repeat`**:

Варианты заданий:

1. Найти в последовательности образованной по закону $a_i=a_{i-1}-2\times a_{i-2}$, где $a_{i-1}=3$, $a_{i-2}=-1$, первый элемент, больший 10 и его номер в последовательности.

2. Число *N* содержит нечетное количество цифр. Среднюю цифру заменить на 7.

3. Найти степень числа *N*, у которой три последние цифры одинаковы.

4. Имеется товар в ящиках по *a, b, c* кг. Получить *M* кг товара, не вскрывая ящиков.

5. Найти первый элемент, больший *K*, последовательностей *{x} и {y},* определяемых рекуррентными соотношениями:

$x_i = x_{i-1} + 2y_{i-1}$

$y_i = y_{i-1} - 2x_{i-1}$

$x_1 = 1, y_1 = 0.$

6. Дано натуральное число *n.* Определить, сколько в числе нулей.

7. Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n=(-1)^n\frac{(n+3)}{2n+1}$.

Слагаемые, по модулю меньше ε, не учитывать.

8. Найти сумму ряда с точностью $\varepsilon=10^{-3}$, общий член которого $a_n=(-1)^{n+1}\frac{x^{2n+1}}{4n^2-1}$.

Слагаемые, по модулю меньше ε, не учитывать.

9. Вычислить значение функции $y=\frac{k}{x^k+y^{k-1}}$, для $k=1,2,3\ldots$. Вычисления производить до тех пор, пока $y\ge z$. Исходные данные – x, y, z (x, y>1) ввести с клавиатуры.

10. Вычислить с точностью $\varepsilon=10^{-3}$:

$$1-\frac{x^2}{2}+\frac{x^3}{3}-\ldots$$

Слагаемые, по модулю меньше ε, не учитывать.

11. Для x<=1 вычислить с точностью $\varepsilon=10^{-3}$:

$$x+\frac{x^3}{1\cdot2\cdot3}+\frac{x^5}{1\cdot2\cdot3\cdot4\cdot5}+\ldots$$

Слагаемые, по модулю меньше ε, не учитывать.

12. Для $x\in[-1,1]$ вычислить с точностью $\varepsilon=10^{-3}$:

Слагаемые, по модулю меньше ε, не учитывать.

13. Сумма в *R* рублей положена в банк. При этом прирост составляет *P%* ежегодно. Через какой промежуток времени сумма достигнет *M* рублей. *(M>R)?*

14. Резервуар наполнен *m* литрами водного раствора, содержащего *s* кг сахара. Приток воды в сосуд составляет *x* литров в минуту, а расход смеси из сосуда *y* литров. Концентрация поддерживается равномерной посредством помешивания. Каждые *h* минут в резервуар засыпают *f* кг сахара *(f<s)*. Какую концентрацию будет иметь раствор через *к* минут?

4. МАССИВЫ

Данный раздел посвящен усвоению основных приемов работы со структурированными переменными типа массив, формированию представлений о различных алгоритмах обработки данных этого типа.

4.1. Структура массива в Паскале

Массивом называется конечная именованная последовательность однотипных величин. Положение каждого элемента в массиве определяется индексом. Чтобы описать массив, надо определить, какого типа его элементы и каким образом они пронумерованы (какого типа его индекс).

Паскаль допускает использование массивов произвольной размерности, но занимать они могут не более 65520 байт.

Формат описания массива:

```
type <имя_типа> = array [<тип_индекса>] of
<тип_элемента>;
var <идентификатор, …>:<имя типа>;
```

Рассмотрим формат описания массивов в разделе var, без предварительного описания типа:

```
var <идентификатор, …>:array [<тип_индекса>] of
<тип_элемента>;
```

Здесь **array** и **of** – ключевые слова. Индексы представляют собой выражения любого порядкового типа. Тип индекса определяет границы изменений значений индекса. Элементами массива могут быть данные любого типа, включая структурированные.

Примеры описания типа:

```
type mas = array [1 .. 10] of real;
Color = array [byte] of mas;
Menu = (F1, F2, F3, F4, F5);
Active = array [Menu] of boolean;
```

В первом примере описан тип массива из вещественных элементов, которые нумеруются от 1 до 10. Во втором – элементами массива являются массивы типа mas, а нумеруются они в пределах, допустимых для типа byte, то е от 0 до 255. В

третьем в качестве индекса использовано имя перечисляемого типа данных, а сами элементы могут принимать значения true или false.

При обращении к элементу массива индексы указываются в квадратных скобках после имени массива. Над элементами массива допустимы все действия, определенные для данного типа. Единственным действием, которое возможно произвести с массивом целиком – присваивание и сравнение. Но для этого массивы должны быть описаны через имя типа, либо в одном списке раздела **var**.

Пример 1**.**

```
type d=array[1..m]of integer;
var d1,d2:d; d3,d4:array[1..m]of integer;
d5:array[1..m]of integer;
d1:=d2; d3:=d4; {возможные действия}
d1:=d3; d5:=d1; d5:=d3;{ошибка, так как d1,d3, d5 имеют разные типы}
```

4.2. Одномерные массивы

Линейный массив является линейной таблицей, в которой для точного указания на элемент данных достаточно знания только одного индекса.

Пример 2**.**

```
  Var
  s, bb : array [-20..20] of real;
  n : array ['A'..'Z'] of Integer;
  r : array [boolean] of word;
```

Примеры обращения к элементам массива: `s[-5], bb[i], n['h'], r[false].`

Пример 3. Написать программу, которая вводит с клавиатуры 20 целых чисел, а затем распечатывает их в обратном порядке.

```
Program Prog4_1;
  var A : array [1..20] Of integer;
  i : integer;
Begin
  for i:=1 to 20 do      {организуем цикл с параметром i }
    readln(A[i]);        {вводим A[i] с клавиатуры }
  {выводим массив в обратном порядке}
  for i:=20 downto 1 do write(A[i]:5);
End.
```

Пример 4. Заполнить массив значениям квадратов индексов элементов.

```
Program Prog4_2;
  const
  n=50;     {константа n задает количество элементов массива}
  var
  A : array [1..n] Of integer;
  i : integer;
Begin
  for i:=1 to n do
    A[i]:=i*i;{присваиваем элементам массива значениям
    квадратов индексов }
  for i:=1 to n do
  write(A[i]:5);
End.
```

Пример 5. Программа находит сумму всех элементов и количество отрицательных элементов в целочисленном массиве из 10 элементов.

```
Program Prog4_3;
  const n=10;          {константа n будет содержать количество
                                          элементов массива}
  var A : array [1 . n] of integer;
   i, sum, num : integer;
Begin
  writeln('Введите ', n, ' элементов массива'):
  for i:=1 to n do read(A[i]);    {вводим массив}
  sum:=0; num:=0;  {задаем начальные значения sum и num}
  for i := 1 to n do begin sum:=sum+A[i];
    if A[i]<0 then inc(num);          {подсчет числа
                                  отрицательных элементов}
  end;
  writeln('Сумма элементов:   ', sum);
  writeln('Отрицательных элементов:  ', num);
End.
```

Пример 6. В линейной целочисленной таблице найти индекс первого элемента, являющегося двузначным числом.

```
Program Prog4_4;
  const n=50;      {константа n будет содержать количество
                                          элементов массива}
  var a : array [1..n] of integer;
  i:integer;       { индекс искомого элемента}
Begin
```

```
  randomize;
  {формируем a[i] генератором случайных чисел}
  for i:=1 to n do a[i]:=random(1000)-500;
  i:=1;
  while (i<=n) and not((a[i]>9) and (a[i]<100))
  do inc(i);       {увеличиваем i, если не находим элемент,
  удовлетворяющий условию задачи }
  {если есть такой элемент, то выводим его индекс}
  if i<= n then writeln('индекс = ', i);
End.
```

4.3. Двумерные массивы

В Паскале можно использовать массивы произвольной размерности, однако практика программирования показывает, что массивы размерности больше двух используются редко. Если речь идет о двумерных (в общем случае – многомерных) массивах, то в описаниях должны быть заданы диапазоны изменения всех индексов. Это можно делать по-разному.
Пример 7.

```
  type dim= array [1..10] of word;
  var dim1: array[1..6, 1..10] of word;
  var dim1: array[1..6] of array [1..10] of
            word;
  var dim1: array[1..6] of dim;
```

Обращаться к элементам получившегося массива можно:
`dim1[4,8] или dim1[4][8].`
Пример 8. Подсчитать в целочисленной двумерной таблице a[1:7, 1:10] количество положительных элементов.

```
Program Prog4_5;
  var a : array [1..20,1..20] of integer;
  m,n,i,j k:integer; { m – количество строк, n – количество
      столбцов массива, k – количество положительных элементов}
Begin
  writeln('Введите m,n');
  readln(m,n);
  writeln('Введите массив');
  for i:=1 to m do
    for i:=1 to n do read(a[I,j]);{вводим массив}
  k:=0;
  for i:=1 to m do
    for j := 1 to n do
```

```
      if a[i,j]>0 then
        k:=k+1;
  writeln('k=',k);
End.
```

Пример 9. Дан двумерный массив A(m,n). Найти максимальный элемент массива и номер содержащего его столбца. Исходную матрицу вывести на экран.

```
Program Prog4_6;
  const m=6,n=5;          {константа m – количество строк,
                             n – количество столбцов массива}
  var a:array [1..m,1..n] of real;
  max: real;      {max – значение максимального элемента }
  i, j, num:integer; {num – номер столбца с максимальным
                                                элементом}
Begin
  writeln('Введите массив');
  for i:=1 to m do
    for j:=1 to n do read(a[i,j]); {вводим массив}
{поиск максимального элемента в цикле}
  max:=a[1,1];
  for i := 1 to m
    for j := 1 to n do
      if a[i,j]>max then
      begin max:=a[i,j];num:=j;end;
  writeln('макс=',max:8:3,'в столбце', num);
End.
```

Контрольные вопросы:

1. Что понимается под массивом?
2. Каковы возможные способы описания массивов (одномерных и многомерных)?
3. В каких случаях целесообразно описывать двумерный массив с помощью одномерных?
4. Какие типы допустимы для описания индексов массивов?
5. Какие типы могут использоваться в качестве базовых для описания массивов?
6. Как осуществляется ввод и вывод массивов?

Лабораторная работа № 4

Задание 1. Составьте программу для решения следующей задачи обработки одномерных массивов произвольной длины. Выведите на монитор исходные данные и результат.

Варианты заданий:

1. Заданы два одномерных массива различных размеров. Объединить их в один массив, включив второй массив между *K*-м и (*K+1*)-м элементами первого (*K* задано),

2. Найти среднее арифметическое заданного массива размером *M*. Преобразовать исходный массив, вычитая из каждого элемента среднее значение.

3. Найти количество перемен знака в массиве из *N* чисел. Нулевые элементы заменить абсолютным значением предыдущего. Если первый элемент нулевой, то заменить его числом *F*.

4. Задан массив из *K* чисел, составить программу замены нулей полусуммой следующего и предыдущего чисел. На место последнего или первого нуля ставить соответственно предыдущее или последующее число.

5. Найти минимальный элемент среди положительных чисел и максимальный среди отрицательных.

6. Удалить из массива целых чисел размером *P* элементов все четные числа, стоящие на нечетных местах, сдвинув оставшиеся в начало массива.

7. Удалить из массива вещественных чисел все максимальные элементы, сдвинув оставшиеся влево на освободившиеся места, справа записать нули.

8. Найти среднее арифметическое элементов, расположенных между первым четным и последним нечетным числом, встречающимся в целочисленной линейной таблице.

9. В заданный массив целых чисел из *N* элементов вставить элемент, равный *M* после последнего минимума.

10. Задан массив целых чисел из *T* элементов. Найти первую пару соседних противоположных чисел (их сумма равна 0).

11. В заданном массиве вещественных чисел найти наибольшую длину цепочки стоящих рядом знакочередующихся элементов.

12. Исключить из заданной целочисленной таблицы из *M* элементов числа *k*-го десятка, на их место сдвинуть элементы

влево.

13. Найти количество различных элементов в заданном массиве вещественных чисел.

14. В целочисленном одномерном массиве длины K найти число, повторяющееся наибольшее количество раз. Если таковых несколько, то сохранить их в массиве.

15. Даны два одномерных массива различных размеров. Упорядочить их по убыванию, получить из них один упорядоченный массив. Примечание: объединять массивы и упорядочивать элементы необходимо одновременно.

Задание 2. Составьте программу для решения следующей задачи обработки двумерных массивов произвольной длины. Выведите на монитор исходные данные и результат.

Варианты заданий:

1. Элементы столбца матрицы с максимальным по модулю элементом в k-ой строке заменить на число X.

2. Переставить i-ую и j-ую строки матрицы.

3. Упорядочить m-ую строку по невозрастанию элементов. Вывести исходный массив и полученный вектор.

4. Из заданной матрицы удалить k-ую строку и l-ый столбец.

5. Из матрицы $B(m,n)$ сформировать матрицу $C(m,n)$, каждый элемент которой получается путем вычитания из соответствующего элемента матрицы B первого элемента данной строки.

6. Задана матрица размером $m \times n$. Просуммировать элементы, расположенные на главной и побочных (соседних с главной) диагоналях. Результат получить в виде вектора.

7. Вычислить количество положительных элементов, расположенных в нечетных строках матрицы, и найти среди них минимальный.

8. В данной действительной матрице найти суммы элементов строк, в которых расположен элемент с наименьшим значением.

9. Найти строки матрицы с наибольшей и наименьшей суммой элементов. Сформировать из этих строк вектор.

10. Задан двумерный массив размерности $m \times n$. Дополнить его строкой и столбцом, в которых записать суммы элементов соответствующих строк и столбцов исходного массива. В элементе $(m+1,n+1)$ должна храниться сумма всех элементов первоначального массива.

11. Целочисленная двумерная матрица разделяется главной диагональю на два треугольника. Из одинаковых чисел среди элементов верхнего и нижнего треугольников сформировать вектор.

12. В заданной двумерной матрице замените строки, содержащие максимальный элемент, на соответствующие строки единичной матрицы.

13. Заданы натуральное четырехзначное число и двумерный массив, элементами которого являются натуральные четырехзначные числа. Определить, имеется ли в таблице число с обратным порядком расположения цифр по отношению к данному. Найти его индексы.

14. Задана вещественная матрица размерности $m \times n$. Переставляя строки и столбцы добейтесь перемещения минимального элемента матрицы в верхний левый угол (если минимум встречается несколько раз, то переставить самый близкий элемент к правому нижнему углу).

15. Дан символьный двумерный массив. Найти номер последнего по порядку столбца, в котором содержится наибольшее количество различных символов.

5. ПРОЦЕДУРЫ И ФУНКЦИИ

Цель: обучение модульной организации алгоритма, формирование умения создания программ с подпрограммами, усвоение понятий формальных и фактических параметров и способов их передачи между основной программой и подпрограммами, выработка навыков построения рекурсивных алгоритмов.

При решении сложных объемных задач часто целесообразно разбивать их на более простые. В этом случае говорят о вспомогательных алгоритмах или подпрограммах. Использование подпрограмм позволяет сделать основную программу более наглядной, понятной, уменьшить вероятность ошибок и облегчить процесс отладки программы, а в случае, когда одна и та же последовательность команд встречается в программе несколько раз, сократить объем программы.

Подпрограмма – это поименованная последовательность операторов, которую можно многократно вызывать для исполнения в любом месте программы. При обращении к подпрограмме в нее передаются исходные данные, а после выполнения операторов подпрограммы передаются в основную программу результаты расчетов. В языке Паскаль существует два вида подпрограмм, определяемых программистом: процедуры и функции, которые отличаются способом использования в программе. Процедуры и функции, используемые в программе, должны быть соответствующим образом описаны в разделе описаний до первого их упоминания. Процедуры и функции, входящие в программу, могут содержать свои подпрограммы и вызвать процедуры и функции более низкого уровня и т.д.

5.1. Процедуры

Процедурой в Паскале называется именованная последовательность инструкций, реализующая некоторое действие. В нужное место программы процедуру вызывают с помощью оператора вызова. После выполнения процедуры программа перейдет к выполнению оператора, следующего за оператором вызова.

Формат описания процедуры:

```
Procedure<Имя процедуры>(<форм. параметры>);
  <Раздел описаний>
Begin
```

```
  <Тело процедуры>
End;
```

Формальные параметры – перечень имен для обозначения исходных данных и результатов работы процедуры, используемых для описания процедуры, с указанием их типов.

Раздел описаний может иметь такие же подразделы, как и раздел описаний основной программы. Однако все описанные здесь объекты доступны лишь в этой процедуре. Они локальны так же, как и имена формальных параметров. Объекты, описанные ранее в разделе описаний основной программы и не переопределенные в процедуре, называются глобальными для этой подпрограммы и доступны для использования.

Формат оператора вызова:

```
<имя процедуры> (<фактические параметры>);
```

Фактические параметры – данные, с которыми выполняется процедура. Между формальными и фактическими параметрами должно быть соответствие по количеству, типу и порядку следования.

Процедуры могут быть без параметров, например, `Procedure Pl1;`. При описании процедур без параметров используются глобальные переменные.

5.1.1. Параметры процедуры

Существует два способа передачи фактических параметров в подпрограмму: по значению и по ссылке. Соответственно параметры называются параметрами-значениями и параметрами-переменными.

В первом случае значение переменной – фактического параметра при вызове подпрограммы присваивается локальной переменной, являющейся формальным параметром подпрограммы. Изменение локальной переменной никак не отражается на соответствующей глобальной. В качестве фактических параметров может служить любое выражение соответствующего типа.

Передача параметров по ссылке отличается тем, что при обращении к подпрограмме имя формального параметра будет указывать на ту же область памяти, что и имя соответствующего фактического параметра. При описании подпрограммы перед именем параметра-переменной ставится служебное слово **`Var`**. В этом случае изменения выполняются в ячейках памяти фактических параметров. Поэтому в качестве фактических параметров можно использовать только имена переменных.

5.1.2. Примеры использования процедур

Пример 1. Написать программу сложения полиномов.

Рассмотрим упрощенный случай, когда оба многочлена имеют одинаковые степени.

```
Program Prog5_1;
  const m =30;
  type pol = array[0..m] of real;
   var f, g, h: pol; k : word; {полиномам-слагаемым и }
                    {полиному-сумме соответствуют массивы f, g и h}
  { Процедура сложения коэффициентов полиномов}
Procedure sum(n:integer; var m,h,s:pol);{параметр
n задает размерность массива <=30}
  var i:integer;
Begin
  for i:=1 to n do
  s[i]:=m[i]+h[i];
End;
Begin
  Randomize;
  for k:=0 to m do begin
    f[k]:=(Random-0.5)*50;{коэффициенты полиномов }
    g[k]:=(Random-0.5)*50;{задаются случайными числами}
  end;
  sum(m,f,g,h);
  {вывод коэффициентов полиномов в три столбца}
  writeln('  f        g        h ');
  writeln('----------------------');
  for k := 0 to m do
    writeln (f[k]:6:3, g[k]:7:3, h[k]:7:3);
End.
```

В процедуру передаем размерность массива, чтобы программа могла складывать полиномы различных степеней, меньших 30. Исходные массивы передаем по ссылке, в этом случае в стеке не создается копия исходных массивов, что улучшает быстродействие и экономит память.

Пример 2. Напишите процедуру нахождения линейной комбинации (k1*a+k2*b) векторов. Найдите какую-либо линейную комбинацию трех векторов размерности 7.

```
Program Prog5_2;
  type Tarray = array[1..100] of real;
```

```
                                  {предположим, что наибольшее число
                                  элементов в массиве не превышает 100}
  var a,b,c,s: Tarray;
  n,i,k1,k2,k3:integer;
  {процедура ввода массива размерности n}
Procedure vvod_data(n:integer; var m:Tarray;);
  var i:integer;
Begin
  writeln('Введите ',n,' чисел');
  for i:=1 to n do read(m[i]); {вводим массив}
End;
  {процедура вычисления линейной комбинации массивов
  размерности n с коэффициентами k и l}
Procedure summ(n:integer;k,l:real; var
m,h,s:TArray);
  var i:integer;
Begin
  for i:=1 to n do s[i]:=k*m[i]+l*h[i];
End;
Begin
  write('Введите размерность массива N= ');
  readln(n);
  vvod_Data(n,a);       {ввода массива a}
  vvod_Data(n,b);       {ввода массива b}
  vvod_Data(n,c);       {ввода массива c}
  summ(n,k1,k2,a,b,s);      {вычисление линейной
  комбинации
                  массивов a, b с коэффициентами k1 и k2}
  summ(n,1,k3,s,c,s);       {вычисления линейной
  комбинации}
                  {полученного массива s с коэффициентом 1}
                    {и массива c с коэффициентом k3}
  writeln('результат= ',s);
End.
```

Пример 3. Вставить элемент x до и элемент y после каждого элемента таблицы, меньшего некоторого числа p. Вставку x и y выполнять в процедуре.

```
Program Prog5_3;
  const n=12; x=1;y=-1;{n – размерность исходного
  массива}
  type mas = array[1..3*n]of integer;
var m:mas; j,i,k,l,:integer;
```

```
Procedure vst(k,x,y,l:integer; var m:mas);{k – номер элемента, меньшего p, l – текущее количество элементов массива}
  var i:integer;
Begin
  for i:=l downto k do m[i+2]:=m[i];
    m[i+1]:=m[i]; m[i]:=y; m[i+2]:=x
End;

Begin
  writeln('введите число p'); readln(p);
  writeln('введите массив');
  for i:=1 to n do read(m[i]);
  writeln;
  l:=n; i:=1;
  while i<= l do
    if m[i] < p then begin
      vst(i,x,y,l,m);{ вставка элементов }
      l:=l+2; i:=i+3; end {увеличиваем число элементов в массиве и номер очередного элемента}
    else i:=i+1;
  for i:=1 to k do write(m[i]:5);
  writeln;
End.
```

Задаем размерность массива 1..3×*n*, т.е. в три раза длиннее исходного, на случай, если все элементы массива окажутся меньше числа *p*.

Пример 4**.** Дана двумерная таблица. Напишите процедуру, которая элементы главной диагонали записывает в одномерную таблицу. Применить к двум таблицам разной размерности.

```
Program Prog5_4;
  type mat=array[1..10,1..10]of integer;
    tab= array[1..10]of integer;
  var a,a1:mat; b,b1:tab; j,i,m:integer;
                    {процедура ввода массива}
Procedure vvod_data(n:integer; var m:mat);
  var i,j:integer;
Begin
  writeln('Введите массив');
  for i:=1 to n do
    for i:=1 to n do read(m[i,j]);
End;
Procedure proc(m:integer; var a:mat;var b:tab);
```

```
  Var i,j:integer;
Begin
  for i:=1 to m do
       b[i]:=a[i,i]; {пересылаем диагональные элементы в
       массив b}
End;
Begin
  write ('введите размерность первого массива'); readln (m);
  vvod_Data(m,a);        {ввода массива a}
  proc(m,a,b);
  for i:=1 to m do write(b[i]:5);
  writeln;
  write ('Введите размерность второго массива '); readln (m);
  vvod_Data(m,a1);       {ввода массива a}
  proc(m,a1,b1);
  for i:=1 to m do write(b1[i]:5);
  writeln;
End.
```

Пример 5. В двумерном массиве переставить строки в порядке возрастания элементов 1-го столбца.

```
Program Prog5_5;
  const m=15;n=7;{mxn – размерность массива}
  type mat=array[1..m,1..n]of integer;
  var a:mat; j,i:integer;
Procedure sort(m1,n1:integer;var a:mat);
  var i,j,m,l,n:integer;
Begin
  for i:=1 to m1 do begin
    m:=i;{m – номер строки, в которой элемент меньше текущего}
    for j:=i+1 to m1 do
      if a[i,1]>a[j,1]then begin m:=j;
      {сравниваем элементы первого столбца}
       for l:=1 to n1 do begin n:=a[i,l];
       a[i,l]:=a[m,l];a[m,l]:=n end;{переставляем
       строки с номером i и m}
      end;
    end;
End;
Begin
  writeln('введите массив');
  for i:=1 to m do
    for j:=1 to n do read(a[i,j]);
  writeln;
```

```
  sort(m,n,a);
  for i:=1 to m do begin
    for j:=1 to n do write(a[i,j]:5);
    writeln;
  end;
End.
```

5.2. Функция

Подпрограмма-функция предназначена для вычисления какого-либо параметра. Функция состоит из заголовка и тела функции. Заголовок функции имеет вид:

Function <имя>(формальные параметры):<тип>;

где **Function** – служебное слово; <имя> – имя функции, определяемое в соответствии с общими правилами построения идентификаторов; <тип> – тип функции, т.е. тип возвращаемого параметра. Функция может возвращать типы вещественные, порядковые, строкового и любого указателя.

У этой подпрограммы два основных отличия от процедуры. Первое отличие – заголовок. Второе отличие в том, что процедура может иметь несколько выходных параметров – результатов, а функция только одно значение, передаваемое через ее имя. Именно этим объясняется то, что в теле функции хотя бы один раз имени функции должно присваиваться вычисленное значение.

Структура функции такая же, как и процедуры. Передача параметров осуществляется так же, как в процедуре.

Пример 6. Функция вычисления тангенса Tan(x) от аргумента x:

```
Function Tan(x: real):real;
  Var tangens: real;
Begin
   tangens:= Sin(x)/Cos(x); Tan:= tangens;
End;
```

Для вызова функции из основной программы или другой подпрограммы в выражении, где необходимо использовать значение функции, следует указать имя функции со списком фактических параметров. Например, чтобы вычислить $y=\frac{tg(x)}{1+tg^2(x)}$, оператор присваивания будет иметь вид:

```
Y:=Tan(x)/(1+Sqr(Tan(x));
```

5.2.1. Примеры использования функций

Пример 7. Найти максимальное из трех чисел.
Поскольку результатом является скалярная величина, то целесообразно использовать подпрограмму-функцию.

```
Program Prog5_7;
  var a,b,c :real;
function max(a,b:real):real; {описываем функцию max с формальными параметрами a и b, которая принимает значение максимального из них}
Begin
  if a>b then max:=a
    else max:=b {здесь a и b – локальные переменные }
end;
Begin
  writeln('введите три числа');
  readln(a,b,c);
  writeln('максимальное ', max(max(a,b),c))
End.
```

Пример 8. Использовать функцию для вычисления максимального элемента в массиве. В основной программе определяется тип – массив.

```
Program Prog5_8;
  type aarr=array [1..50] of integer;
  var mas:aarr; maxim:integer;
   {функция в этом случае может иметь вид}
function max(var mas: aarr; n: byte): integer;
  var ma : integer; i : byte;
begin
  ma:=mas[1];
  for i:= 2 to n do
    if ma < mas[i] then ma ;= mas[i]; max := ma
end;
Begin
  write('Введите размерность массива '); readln(m);
  writeln('введите массив');
  for i:=1 to n do read(mas[i]); {вводим массив}
  maxim:=max(mas,35); {определяем максимальное число из
                       первых 35 чисел массива}
  writeln('максимальное ', maxim)
End.
```

Пример 9. Упорядочить таблицу в порядке возрастания количества цифр в числах.

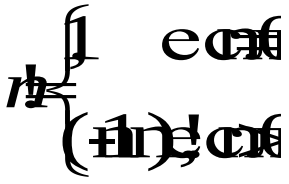
Для определения количества цифр в числах будем использовать функцию.

```
Program Prog5_9;
  const m1=12; s:array[1..m1]of integer = (11,
  9,6,147,3,5,10,5789,8,112,7,12); {описываем массив
  как типизированную константу}
  var n,j,i,k:integer;
function cifra(t:integer):integer;
  var d,k:integer; {k – счетчик цифр в числе}
begin
  d:=t; k:=1;d:=d div 10; {количество цифр в числе}
  while d>0 do begin  {находим целочисленным }
    d:=d div 10; k:=k+1 {последовательным делением на 10}
  end;
  cifra:=k;
end;
Begin
  for i:=1 to m1 do write(s[i]:5); writeln;
  k:=0;
  for j:=2 to m1-1 do begin {выполняем сортировку}
    for i:=m1 downto j do {массива по возрастанию}
      if cifra(s[i-1])>cifra(s[i]){количества цифр}
        then begin k:=s[i];          {в числах}
        s[i]:=s[i-1]; s[i-1]:=k; end;
  end;
  for i:=1 to m1 do write(s[i]:5); writeln;
End.
```

5.3. Рекурсивные процедуры и функции

При вычислениях возможны случаи, когда подпрограмма обращается к себе самой. Такую ситуацию называют рекурсией.

Примером рекурсии служит подпрограмма вычисления факториала натурального числа:



```
function f(n:integer ):integer;
begin
  if n=0 then f:=1 else f:=f(n-1)*n;
end;
```

Обращение к данной функции, например, с фактическим параметром 3, влечет следующие действия.

```
f:=3*f(2)          (первый шаг)
  f:=2*f(1)        (второй шаг)
    f:=1*f(0)      (третий шаг)
```

На первом и втором шагах вычисление откладывается, на третьем шаге f получает значение, равное 1, и происходит вычисление факториала в обратном порядке.

Любое рекурсивное описание должно содержать явное определение результата для некоторого значения аргумента (аргументов), которое будет обеспечивать завершение рекурсивных вызовов подпрограммы. В рассмотренном примере окончание вычислений обеспечивается оператором **if** n=0 **then** f:=1.

5.3.1. Примеры рекурсивных функций

Пример 10. Вычислить произведение двух целых положительных чисел с помощью рекурсивной функции, не используя цикл.

```
Program Prog5_10;;
  var a,b,pr:integer;
function p(a,b:integer ):integer;
Begin {Произведение чисел a и b представим как сумму из b чисел
       a, т. е. a*b= a*(b-1)+ a}
  if b=1 then p:=a else p:=p(a,b-1)+a;
End;
Begin
  writeln('введите два числа');
  readln(a,b);
  pr:=p(a,b);
  writeln(pr);
End.
```

Пример 11. Вычислить  используя рекурсивную функцию.

Решение этой задачи аналогично примеру 8 темы 3. Находим связь между предыдущим и следующим членами последовательности (рекуррентное соотношение). Член a_i связан с a_{i-1} соотношением: $a_i = a_{i-1} \cdot \frac{x}{i}$. Множитель, с помощью которого связаны два соседних члена последовательности, равен $\frac{x}{i}$.

```
Program Prog5_11;
```

```
  var n:integer; x:real;
function s(x:real; n:integer; v:real;
```

```
k:integer):real; {n – количество слагаемых, v – множитель}
                                {k – номер очередного слагаемого}
begin
  if n=0 then s:=0 else begin
    s:=s(x,n-1,v*x/k,k+1)+v; end; {к сумме
    предыдущего шага прибавляем очередное слагаемое. Значение
    очередного члена и его номер вычисляется рекурсивно через
    обращение к функции}
end;
Begin
  writeln('введите x и n');
  readln(x,n);
  writeln(s(x,n,1,1):11:5);
End.
```

При обращении к функции задаем в качестве начальных значений `v = 1` и `k = 1`.

Пример 12. Вычислить $\sin x \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^k \frac{x^{2k+1}}{(2k+1)!}$, с точностью ε.

Отличие этой программы от предыдущей в том, что вместо количества слагаемых задаем точность вычислений. Поэтому выход из рекурсии выполняется по условию `v< ε`.

```
Program Prog5_12;
  const e=0.001;
  var x,f:real;
function s(x,e:real; v:real; k:integer):real;
begin
  if v<e then s:=0 else s:=s(x,e,v*x/k,k+1)+v;
end;
Begin
  writeln('введите x');
  readln(x);
  f:=s(x,e,1,1);
  writeln(f:11:5);
End.
```

Пример 13. С помощью рекурсивной функции вычислить цепную дробь:

$$t = k + \cfrac{1}{k + \cfrac{1}{k + \cfrac{1}{k + \cfrac{1}{k + \ldots}}}}$$

Рекуррентное соотношение состоит в следующем: если вычислена цепная дробь из n-1 элемента, то чтобы вычислить дробь, содержащую на один элемент больше, нужно 1 разделить на полученную дробь и прибавить к k.

```
Program Prog5_13;
  var k,n:integer; c:real;
  function t(k,n:integer):real;{n – количество
                                элементов дроби}
begin
  if n=1 then t:=k else t:=k+1/t(k,n-1);
end;
Begin
  readln(k,n);
  c:=t(k,n);
  writeln(c:11:5);
End.
```

Контрольные вопросы:

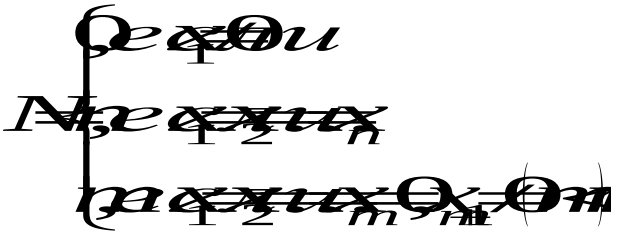
1. В каких случаях используются подпрограммы?
2. В чем отличие между процедурой и функцией, для каких задач каждая из них предназначена?
3. Какая существует связь между формальными и фактическими параметрами?
4. В чем особенности механизма передачи параметров – значений и параметров переменных?
5. Какие типы можно использовать для описания значения функции?
6. Что представляет собой рекурсивный алгоритм?
7. В чем особенности описания рекурсивной функции? Как обеспечивается конечность рекурсивного алгоритма?
8. Как реализовать циклический алгоритм с помощью рекурсивной функции без оператора цикла? Что такое рекурсия с накапливающимся параметром?

Лабораторная работа № 5

Задание 1. Составьте программу с использованием процедуры – функции. Выполните ее тестирование и отладку.

Варианты заданий:

1. Для заданного вектора $(x_1, x_2, \ldots x_n)$ вычислить величину:



2. Определить количество дней в месяце по его порядковому номеру. Учесть високосные года.

3. Вычислить сумму значений функции:

z = f(|x|,y)+f(a,b)+f(|x|+1,-y)+f(|x|-|y|,x)+f(x+y,a+b),

где $f(u)=\begin{cases} u+2 & u\ge 0, \\ u+t & u<1, \\ u^2-2u+1 & -1<u<0. \end{cases}$

4. Найти наименьшую сторону треугольника, заданного координатами своих вершин.

5. Даны действительные числа s и t. Получить:

$$z=h(s,t)+\max(s-t,st)+h(t,t),$$

где $h(a,b)=a/(1+b^2)-(a-b)^3$.

6. По заданному значению величины x определить, имеет ли функция

$$y(x)=\frac{1}{\sqrt{x+1}}+\frac{1}{x^2-8x+1}$$

значение, и найти его.

7. Вычислить значения функции $y=ax^2+bx+d$, где

используя функцию $sum=\sum_{i=k}^{l} m_i$.

8. Пусть элементами круга являются радиус (1-й элемент), диаметр (2-й элемент), длина окружности (3-й элемент). По номеру одного из перечисленных элементов и его значению вычислить площадь круга.

9. Вычислить корни x и y системы уравнений:

$$\begin{cases} a_1x+b_1y=c_1 \\ a_2x+b_2y=c_2 \end{cases}.$$

10. Вычислить площади треугольника по координатам его

вершин.

11. Вычислить величину:

используя функцию $shx=\frac{e^x-e^{-x}}{2}$.

12. Даны действительные числа s и t, получить:

где 

13. Составить процедуру-функцию для вычисления определителя третьего порядка. Описать матрицу коэффициентов как двумерный массив.

14. Проверить, верна ли гипотеза Гольдбаха о том, что каждое четное $n > 2$, представляется в виде суммы двух простых чисел.

15. В произвольном массиве найти наиболее часто повторяющееся число. Оформить функцию вычисления количества повторений одного числа. Дополнительных массивов не использовать.

Задание 2. Составьте программу с использованием рекурсивной подпрограммы для решения следующей задачи:

Варианты заданий:

1. Найти сумму ряда с заданной точностью ε, общий член которого имеет вид:

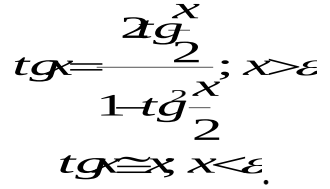
$$a_n=\frac{2n-1}{2^n}$$

2. Вычислить *n*-ый член последовательности, общий член которой имеет вид:

3. Последовательность чисел задана соотношениями:

Вычислить *n*-ый член последовательности.

4. Вычислить *tg x* с заданной точностью ε по формулам:

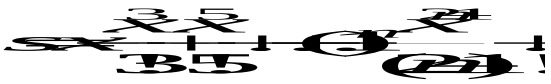


5. Вычислить для заданных *x* и *n*:

$$e^x = \sum_{k=0}^{n} \frac{x^k}{k!}$$

6. Найти *n*-ый член последовательности, заданной соотношениями:

7. Вычислить значение суммы членов бесконечного ряда



с точностью до члена ряда, меньшего по модулю ε.

8. Вычислить приближенное значение кубического корня из *a*, используя соотношение:

9. Для вещественного *x (x<>0)* и целого *n* вычислить x^n согласно формуле:

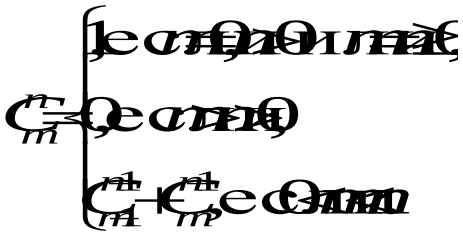
10. Найти произведение  для заданных *a* и *n*.

11. Даны неотрицательные целые числа *n, m*; вычислить *A(n,m)* – функцию Аккермана, где

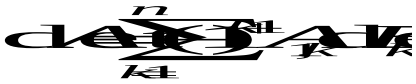
12. Вычислить *ln x*, пользуясь формулами:

ε – заданная точность.

13. Для заданного $M \geq m \geq n > 0$ вычислить все биномиальные коэффициенты C_m^n по формулам:



14. Вычислить определитель заданной матрицы, пользуясь формулой разложения по первой строке (матрица *B* получается вычеркиванием из *A* первой строки и *k*-го столбца):



15. Вычислить по формуле:

$$e^x = \frac{1+t}{1-t}, \text{ где } t = \cfrac{1}{\frac{1}{q} + \cfrac{1}{\frac{3}{q} + \cfrac{1}{\frac{5}{q} + \ldots + \cfrac{1}{\frac{2n+1}{q}}}}}, \quad q = \frac{x}{2}$$

Задание 3. Составьте программу с использованием процедуры для решения следующей задачи:

Варианты заданий:

1. Найти наименьшие элементы двух различных матриц произвольной размерности.
2. «Сожмите» одномерные массивы размерности *m* и *n*, удалив из них нулевые элементы.
3. Найти сумму диагональных элементов матриц *A* и *B*.
4. В двух целочисленных последовательностях различной длины замените все элементы, следующие за элементом с максимальным значением, на значение минимального элемента.
5. Вычислить  , где x_1, x_2 – корни уравнения $2x^2+x-4=0$, y_1, y_2 – корни уравнения $3y^2+2y-1=0$.
6. Напишите процедуру для объединения двух упорядоченных линейных массивов в один упорядоченный. Объедините три массива размерности *m, n, k.*
7. Определить, в каком из двух заданных массивов натуральных чисел больше палиндромов (чисел, которые читаются одинаково как с начала, так и с конца, например, 313, 66).
8. Вычислить:

$$Z = \frac{\sum_{i=1}^{8} \sin x_i + \sum_{i=1}^{10} \cos y_i}{\sum_{i=1}^{40} |x_i|},$$

где y_i и x_i заданы массивами. Все суммы поочередно вычислять одной процедурой.

9. Вычислить *B*A*C*. Произведение двух матриц находить в процедуре.

10. Вычислить $z=\frac{x_{\max}\overline{y_{\min}}}{2}$, где $x_{\max}$ – максимальный элемент массива *X(n);* $y_{\min}$ – минимальный элемент массива *Y(m).* Элементы $x_{\max}$ и $y_{\min}$ вычислять одной процедурой.

11. Два двумерных массива различной размерности преобразуйте в линейные.

12. Заданы координаты вершин *n* треугольников. Выяснить, сколько из них прямоугольных, равносторонних и равнобедренных. Определение вида треугольника оформить в виде процедуры.

13. Найти сумму минимумов дополнительных миноров диагональных элементов матрицы.

14. В двух двумерных массивах найти наиболее часто повторяющиеся элементы.

15. Симметричная двумерная матрица представлена своим верхним треугольником в виде линейного массива. Преобразуйте два линейных массива произвольной размерности в симметричные матрицы.

6. СТРОКИ

Цель: выработка навыков использования строковых переменных для обработки текстовой информации.

6.1. Структура строки в Паскале

Строка есть последовательность символов языка Паскаль. В выражениях константу-строку заключают в апострофы.

Формат описания строки:

```
var <идентификатор1>, …:string[<длина строки>];
```

<длина строки> – количество символов в строковой переменной. Число символов не может превышать 255. Фактическая длина строки может быть короче указанной в описании. Длину строки можно не указывать. В этом случае ее длина будет предельной – 255 символов. Для хранения строки в памяти отводится количество байтов на 1 больше длины строки. В нулевом байте хранится реальная длина строки. Доступ к элементам строки – символам – осуществляется так же, как к элементам массива, с помощью индекса, который записывается в квадратных скобках за идентификатором строковой переменной.

Примеры описания строк:

```
const slovo = 'massiv';
  frasa = 'Я пишу программу';
type subekt = string[30];
var B,A : subekt;
fio : string[30];
text: string;
```

Над строками определена операция соединения (конкатенации) «+». Она соединяет две строки в одну результирующую строку.

Пример 1.

Выражение	Результат
'Дом '+'номер'+' 5/43'	'Дом номер 5/43'

Строки можно сравнивать. При сравнении двух срок символы сравниваются слева направо по значению их ASCII-кода. Строки равны, если их длины и содержание одинаковы.

Пример 2.

Выражение	Результат
`'akkord'>'AKKORD'`	`true`
`'Принтер  '<'Принтер'`	`false`
`'XXX'>'XX'`	`true`

Если значение переменной после выполнения оператора присваивания больше заданной длины строки, то все лишние символы справа отбрасываются.

Пример 3.

```
var Str1,Str2: string[20];
  Str1 := 'Группа учащихся';
  Str2 := Str1 + ' первого курса';
```

Значение `Str2` окажется равным `'Группа учащихся перв'`.

Допускается использование в одном выражении операндов строкового и символьного типа.

Пример 4**.**

```
var s1,s2:string[20]; ch:char;
.....
s1:= 'учусь'; ch:= 'Я'; s2:=ch+s1; {возможные действия}
ch:=s1;      {ошибка}
```

6.2. Процедуры и функции для работы со строками

6.2.1. Функции

`Concat(si, s2, sn)` возвращает строку, являющуюся слиянием строк si, s2, ..., sn. Ее действие аналогично операции конкатенации.

`Copy(s, i, n)` возвращает подстроку длиной n, начинающуюся с позиции i строки s. Параметры n и i должны быть целого типа.

`Length(s)` возвращает фактическую длину строки s, результат имеет тип byte.

`Pos(s1, s)` возвращает номер первого символа подстроки s1 в строке s или нуль, если s1 не содержится в s.

6.2.2. Процедуры

`Delete(s, i, n)` удаляет из строки s, начиная с позиции i, подстроку длиной n.

`Insert(s1, s, i)` вставляет в строку s подстроку s1, начиная с позиции i.

`Str(x, s)` преобразует числовое значение x в строку s, при этом для x может быть задан формат, как в процедурах вывода.

`Val(s, x, code)` преобразует строку s, содержащую символьное представление числа, в значение числовой переменной x. В случае успешного преобразования переменная code равна нулю. Если же обнаружена ошибка, то code будет содержать номер позиции первого ошибочного символа, а значение x не определено.

6.2. Примеры программ со строками

Пример 5**.** Написать программу, которая заменяет заданную с клавиатуры последовательность символов в тексте на многоточие.

```
Program Prog6_1;
  Var
  s, str:string[10]; { str – текст, s –
  последовательность символов}
  dl,j: integer;
Begin
  writeln('Введите текст');
  readln(str);
  writeln('Какую последовательность заменять?');
  readln(s);
  dl:=length(s);i:=1;
  j:=Pos(s,str); {ищем в str позицию первого вхождения s }
  while j <> 0 do begin
    Delete(str,j,dl); { если найдена удаляем символы s из
    str}
    Insert('...', str, j); {вставляем на их место '...'}
    j:=Pos(s,str);
  end;
  writeln(str);
End.
```

Пример 6. Составить программу, которая бы составляла словарь данного предложения. Слова разделены одним пробелом.

Словарь организуем в виде массива, элементами которого будут слова предложения. Слова выделяем посимвольно из предложения по достижении пробела.

```
Program Prog6_2;
  Var
  s: string; {s – текст}
  s1:array[1..250] of string; {массив для словаря}
```

```
  i, k, dl: integer;
Begin
  writeln('Введите текст');
  readln(s);
  s:=s+' ';    {добавляем в конец предложения пробел для
  единообразного выделения слов}
  dl:=length(s);
  i:=1; k:=1;
  while i<=dl do begin
    while s[i]<>' ' do begin
    s1[k]:=s1[k]+s[i]; {выделяем слово до пробела}
    inc(i); end;
    i:=i+1; k:=k+1;
  end;
  for i:=1 to k-1 do writeln(s1[i]);
End.
```

Контрольные вопросы:

1. Какая информация может быть представлена с помощью строк? Каковы основные правила их описания?
2. Как осуществляются ввод и присваивание значений строковым переменным?
3. В чем сходство и отличие строковой переменной и массива символов?
4. Какие основные действия над строками можно выполнять с помощью стандартных процедур и функций?

Лабораторная работа № 6

Задание 1. Задано слово – последовательность произвольных символов. Составьте программу для решения следующей задачи:

Варианты заданий:

1. Заменить символ *x* на *iks*.
2. Вычеркнуть буквы *А* и *Б*.
3. Удалить все буквы с четными номерами.
4. Найти сумму цифр, встречающихся в данной последовательности символов.
5. Удалить все предыдущие вхождения последней буквы.

6. Каждой букве алфавита (ё не учитывать) поставить в соответствие код – номер места буквы в алфавите (от 1 до 32). Заданное слово записать с помощью кода, разделяя коды букв пробелами.

7. Дана символьная строка и слово, состоящее из четырех символов. Определить, есть ли в данной строке все буквы данного слова.

8. Составить слово-перевертыш к заданному (символы записываются в обратной последовательности).

9. Удвоить каждую букву в слове, иные символы оставить без изменения.

10. Удалить среднюю букву, если слово нечетной длины, и вставить посередине пробел в ином случае.

11. Подсчитать наибольшее количество идущих подряд одинаковых символов.

12. Оставить в слове только первые вхождения каждой буквы.

13. Расположить все буквы данного слова в алфавитном порядке.

14. Определить, есть ли в данной последовательности два любых одинаковых символа, и вывести соответствующее сообщение.

15. Получить новую строку, взяв из данной все символы, находящиеся между первой открывающейся скобкой и последней закрывающейся (если какие-либо скобки отсутствуют, то вывести соответствующее сообщение).

Задание 2. Дана символьная строка. Слово – последовательность символов между пробелами, не содержащая пробелы внутри себя. Составьте программу для решения следующей задачи:

Варианты заданий:

1. Разбить текст на строки длиной в два слова. Перенос на новую строку осуществить на месте пробела.

2. Определить количество слов в данной строке.

3. Расположите слова данного предложения в порядке возрастания числа букв в словах.

4. Определить количество и вывести все самые длинные слова.

5. Определить длину слова, стоящего на N-ом месте и вывести все слова, состоящие из такого же количества символов,

что и найденное слово. Если N больше количества слов в предложении, то вывести соответствующее сообщение.

6. Определить количество слов, которые начинаются и оканчиваются одним и тем же символом.

7. Определить количество слов в строке, оканчивающихся на заданный символ.

8. Вывести все слова, в которых есть заданный символ.

9. Отредактировать заданное предложение, удаляя из него слова, которые уже встречались.

10. Для каждого из слов указать, сколько раз оно встречается в данной строке.

11. Найти слова, в которых нет повторяющихся букв.

12. Даны две символьные строки. Вывести слова, которые встречаются в обеих строках.

13. Подсчитать количество слов, имеющих удвоенные буквы.

14. Расположить слова данного предложения в алфавитном порядке по первой букве.

15. Отредактировать заданное предложение, удаляя из него слова с нечетными номерами и переворачивая слова с четными (например, how do you do --> od od).

7. ФАЙЛЫ

Цель: формирование представления о файлах данных, выработка умений использования различных способов организации информации в виде файла.

Файл в Паскаль-программе представляет структурированный тип данных, образованный последовательностью компонентов одного типа. В Паскале имеются три разновидности файлов: типизированные, безтиповые и текстовые. Доступ к физическим файлам происходит через так называемые файловые переменные.

Для организации ввода/вывода в программе необходимо выполнить следующие действия:

- объявить файловую переменную;
- открыть файл для чтения и/или записи;
- выполнить операции ввода/вывода;
- закрыть файл.

В любой момент доступен только один компонент файла. Другие компоненты становятся доступными по мере продвижения по файлу. В отличие от массива число компонентов в файле не фиксируется. Его длина определяется в процессе выполнения программы и ограничивается только доступной памятью. Позиция файла, следующая за последним компонентом файла (или первая позиция пустого файла), помечается специальным маркером, который отличается от любого компонента файла.

Для работы с файлами используют специальные процедуры и функции. С их помощью осуществляются все необходимые действия по созданию файлов и доступа к их компонентам.

Перед выполнением операций файловая переменная связывается с физическим файлом, после чего он в тексте программы не упоминается. В Паскале есть подпрограммы, применяемые для файлов любого типа, а также подпрограммы для работы только с определенными типами файлов.

Для эффективного выполнения операций ввода/вывода из внешних файлов в Паскале целесообразно использовать нетипизированные файлы, так как при работе с ними можно использовать быстрые дисковые операции низкого уровня. Нетипизированные файлы дают возможность прямого доступа к любому файлу на диске независимо от его типа и структуры.

7.1. Подпрограммы для работы со всеми типами файлов

Процедуры:

`assign(var fp; filename : string)` – связывает файловую переменную `fp` с физическим файлом, имя которого задано в строке `filename`. Если путь к файлу не задан, предполагается, что он находится в текущем каталоге.

`rewrite(var fp)` – открывает новый файл для записи данных. Если физический файл, с которым связана переменная `fp`, существовал ранее, он очищается, то есть вся информация из него теряется.

`reset(var fp)` – открывает существующий файл для последующего чтения или записи данных.

`close(var fp)` – закрывает открытый файл.

7.2. Типизированные файлы

Формат типизированного файла:

```
type <имя типа> =file of <тип компонента>;
var <имя файловой переменной>:<имя типа>;
```

Для ввода информации из файла, открытого для чтения, используется оператор:

```
read(<имя файловой переменной>,<список ввода>);
```

При этом происходит считывание данных из файла в переменные, имена которых указаны в списке ввода. Переменные должны быть того же типа, что и компоненты файла.

Вывод информации выполняется оператором:

```
write(<Имя файловой переменной>,<список вывода>);
```

Примеры описания файлов:

```
type n = file of integer; { тип – файл целых чисел};
  c = file of char; {тип – файл символов};
  massiv=array[1..5] of real;
var f1:n; f2:c;
  z : file of word;
  f: file of massiv; {файл, компонентами которого являются
  массивы вещественных чисел}
```

Пример 1. Записать в файл 20 чисел, образуемых по формуле $i^2/2$.

```
Program Prog7_1;
  var
  fp: file of real; {файл вещественных чисел }
  i: integer; a:real;
Begin
  assign(fp; 'mas.dat');
  rewrite(fp);       {создаём новый файл с именем mas.dat }
  for i:=1 to 20 do begin
    a:=sqr(i)/2;     {вычисляем очередной элемент }
    write(fp,a);     {записываем его в файл}
  end;
  close(fp); {закрываем файл}
End.
```

Пример 2. Найти сумму первых 10 элементов файла 'mas.dat'

```
Program Prog7_2;
  var
  f: file of real; {файл вещественных чисел }
  i: integer; s,a:real;
Begin
  assign(f; 'mas.dat');
  reset(f);     {открываем для чтения файл с именем mas.dat }
  s:=0;
  for i:=1 to 10 do begin
    read(f,a);     {читаем из файла очередной элемент }
    s:=s+a;        {вычисляем сумму}
  end;
  write('s=',s:10:3);
  close(f); {закрываем файл}
End.
```

Таблица 1

Некоторые стандартные процедуры и функции для работы с файлами

Подпрограмма	**Описание**
Функции	Возвращает
`Eof(var fp): boolean`	для файла, связанного с файловой переменной `fp`, состояние 'конец файла': true – если текущее положение указателя находится в конце файла
`FilePos(var fp): Longint`	текущую позицию для файла, связанного с файловой переменной `fp`. Для текстовых файлов не используется

`FileSize(var fp): Longint`	текущий размер файла, связанного с файловой переменной `fp`. Если файл пустой, возвращает нулевое значение. Для текстовых файлов не используется
`IOResult: Integer`	целое значение, являющееся состоянием последней выполненной операции ввода/вывода. Нулевое значение соответствует нормальному завершению
Процедуры	
`Seek(var fp; n: Longint)`	Перемещает текущую позицию в файле, связанном с файловой переменной `fp`, на заданный элемент. Используется для организации произвольного доступа.
`Truncate(var F)`	Уничтожает все компоненты файла, связанного с файловой переменной `fp`, после текущей позиции и подготавливает файл для записи. Вся информация после текущего положения указателя теряется.

Пример 3**.** Добавить еще 10 записей, задаваемых по правилу примера 1 в конец файла `'mas.dat'`.

```
Program Prog7_3;
  var
  f: file of real; {файл вещественных чисел }
  i: integer;
Begin
  assign(f; 'mas.dat');
  reset(f);    {открываем для чтения файл с именем mas.dat }
    seek(f,FileSize(f)); {устанавливает указатель}
                         {за последним компонентом файла}
  for i:=21 to 30 do
    write(f,sqr(i)/2);
  close(f);          {закрываем файл}
End.
```

Пример 4. В файл `f1` записана целочисленная матрица, размерностью `mxn` поэлементно. Переписать содержимое файла `f1`, начиная с `k`-ой строки в файл `f2`.

```
Program Prog7_4;
  var
  f1,f2: file of integer; {файлы целых чисел }
  a,k,n: integer;
Begin
  assign(f1; 'matr.dat');
```

```
  reset(f1);    {открываем для чтения файл matr.dat}
  assign(f2; 'posl.dat');
  rewrite(f2);
  write('введите число столбцов и номер
  строки');
  readln(n,k);
  seek(f1,n*(k-1)-1); {устанавливает указатель}
                      {на первый элемент k-ой строки}
  while not eof(f1) do begin {пока не достигнут конец
                              файла}
    read(f1,a);       {читать элемент из файла f1}
    write(f2,a);      {писать этот элемент в файл f2}
  end;
  close(f1);        {закрываем файл}
  close(f2);        {закрываем файл}
End.
```

7.3. Текстовые файлы

Для описания текстовых файлов в Паскале имеется стандартный файловый тип `text`, например:

```
var in_file:text;
```

Доступ к текстовому файлу организуется последовательно, то есть программа не может в любой момент времени считать из него произвольную порцию информации или произвести запись в произвольное место файла.

Для работы с текстовым файлом используются процедуры `assign`, `rewrite`, `reset`, `close`. Однако текстовый файл, открытый процедурой `reset`, доступен только для чтения, в отличие от типизированных файлов.

Для записи в текстовый файл или чтения из него можно использовать процедуры `Write`, `WriteLn`, `Read` и `ReadLn`. В этом случае в качестве первого параметра в этих процедурах указывается файловая переменная, например

```
Readln(in_file, a, x);
```

Здесь происходит присваивание переменным `a` и `x` значений двух очередных элементов из файла, связанного с файловой переменной `in_file`.

Текстовый файл может использоваться для хранения численных значений. При считывании значений или их записи в файл происходит автоматическое преобразование из числового формата в символьный, и наоборот.

Оператор вывода допускает описание формата вывода. Если a является выражением целого, булевого или строкового типа, то оператор `WriteLn(in_file, a:n);` означает запись значения переменной `a` в правые позиции поля размером в `n` позиций.

Таблица 2

Процедуры и функции обработки текстовых файлов

Функция/процедура	Описание
`Append(var fp);`	Открывает существующий файл, связанный с файловой переменной `fp`, для добавления в него новых записей
`Eoln(var fp)`	Возвращает `True`, если указатель файла достиг маркера конца строки. В противном случае `False`.
`SeekEoln(var fp)`	Аналогична предыдущей, но указатель файла пропускает все знаки табуляции, предшествующие маркеру конца строки.

Пример 5**.** Переписать из текстового файла `f` в файл `g` строки в перевернутом виде.

```
Program Prog7_5;
  var
  f,g:text;
  i: integer;
  st,st1:string;
Begin
  assign(f,'inp1.dat');
  reset(f);      {открываем для чтения файл 'inp1.dat'}
  assign(g,'out1.dat');
  rewrite(g);{открываем для записи файл  'out1.dat'}
  while not eof(f) do begin  (пока не конец файла)
    readln(f,st);          {читать строку из файла f}
    st1:='';{формируем новую строку-перевертыш}
    for i:= 1 to length(st) do st1:= st[i]+st1;
    writeln(g,st1);       {писать эту строку в файл g}
  end;
  close(f);          {закрываем файл f}
  close(g);          {закрываем файл  g}
End.
```

Пример 6. Дописать файл g в конец файла f.

```
Program Prog7_6;
  var
  f,g:text;
  i: integer;
  st:string;
Begin
  assign(f; 'inp1.dat');
  append(f);    {открываем файл 'inp1.dat' для дозаписи}
  assign(g,'out1.dat');
  reset(g); {открываем для чтения файл  'out1.dat'}
  while not eof(g) do begin (пока не конец файла)
    readln(g,st);          {читать элемент из файла g}
    writeln(f,st);       {писать этот элемент в файл f}
  end;
  close(f);          {закрываем файл f}
  close(g);          {закрываем файл g}
End.
```

Контрольные вопросы:

1. Что такое файл? Какие существуют виды файлов? Для чего они предназначены?
2. Какая связь между файлом и файловой переменной? Как она устанавливается?
3. Какие стандартные процедуры обязательно включаются в программу, обрабатывающую файлы?
4. Какие типы допустимы для компонентов файла?
5. Как осуществляется обращение к компонентам файлов прямого и последовательного доступа?
6. Приведите различные способы записи в файл табличной информации?
7. Как вывести содержимое файла на монитор, на печать?
8. Каковы особенности работы с текстовыми файлами? Чем эти файлы отличаются от файлов из строк?

Лабораторная работа № 7

Задание 1. Запишите в файл исходных данных числовой двумерный массив.

Задание 2. Напишите программу, изменяющую в файле исходных данных из задания 1 элементы, соответствующие вашему варианту:

Варианты заданий:

1. вторая строка;
2. третий столбец;
3. элементы (2;3) и (4;1);
4. последняя строка;
5. элементы (1;2) и (3;3);
6. третья строка;
7. элементы (2;1),(2;3) и (4;1);
8. последний столбец;
9. второй столбец;
10. элементы (3;2),(1;4) и (5;4);
11. первый столбец;
12. элемент (*r*,*k*);
13. диагональные элементы;
14. элементы в нечетных строках;
15. крайние элементы матрицы (по периметру).

Задание 3. Прочитать из полученного файла элементы, соответствующие вашему варианту:

Варианты заданий:

1. первый столбец;
2. элементы (1;3) и (3;3);
3. вторая строка;
4. элементы (4;3) и (2;1);
5. последняя строка;
6. элементы (3;4) и (4;3);
7. последний столбец;
8. элементы (2,5) и (3:1);
9. элементы (2;2),(3;2) и (4;3);
10. четвёртый столбец;
11. элементы (1;3),(2;1),(3;4);
12. элементы (2;2),(2;3);
13. третья строка;
14. второй столбец;
15. угловые элементы матрицы.

Задание 4. Создать текстовый файл (3–4 строки). Дописать в конец файла 2 строки. Вывести на монитор исходный и полученный файлы.

8. ЗАПИСИ

Цель: формирование понятия о возможности представления разнородной информации об объекте с помощью комбинированного типа «запись» и навыков использования данного типа для описания информационных моделей.

8.1. Структура записи в Паскале

Записи являются структурированным типом данных. Они состоят из компонент, имеющих в общем случае разные типы. Компонент записи называется полем и обозначается идентификатором – именем поля. Описание записей возможно с применением типов или в разделе описаний переменных:

```
type
<имя типа> = record
  <идентификатор поля 1>: тип 1;
  <идентификатор поля 2>: тип 2;
  ..........................................................
  <идентификатор поля k>: тип k;
end;
var <идентификатор>:<имя типа>;
  <идентификатор>:record
  <идентификатор поля 1>: тип 1;
  <идентификатор поля 2>: тип 2;
  ..........................................................
  <идентификатор поля k>: тип k;
end;
```

В секции могут перечисляться несколько имен полей одного типа. Тип поля может быть любым, кроме файлового.

К каждому компоненту записи можно обратиться, используя имя переменной типа записи и имя поля, разделенные точкой: `<идентификатор>.<имя поля>`. При обращении к элементу массива записей, индекс указывается после имени массива: `<идентификатор массива>[индекс].<имя поля>`

Введение в Паскале такого типа вызвано необходимостью описания реального объекта в целом.

Пример 1. Описание времени и даты.

```
type
time = record
  hour:1..12;
```

```
  minute:0..59;
end;
var t1,t2:time;
date : record
  day:1..31;
  month:0..12;
  year:word
end;
```

К компонентам записи можно обратиться, например: `t1.hour, t2.minute, date.month.`

Над компонентами записи определены операции, допустимые для соответствующего типа.

Пример 2**.** Описать объект – пациент:

- ФИО
- возраст
- болезнь
- семейное положение

Выяснить, поступил ли в отделение конкретный пациент.

Вывести количество больных гриппом.

```
Program Prog8_1;
  const n=55;
  type
  pacient = record
    name, bolezn:string[40];
    wozrast:byte;
    sem_pol:char
  end;
  var
  nekto: pacient;
  otdel: array[1..n]of pacient; {массив записей}
  i, k: integer;
Begin
  writeln('Введите список отделения');
  for i:=1 to n do begin
    readln(otdel[i].name, otdel[i].bolezn);
    readln(otdel[i].wozrast, otdel[i].sem_pol);
  end;
  writeln('Введите фамилию пациента');
  readln(nekto.name); i:=1;
  while (i<= n) and (otdel[i].name <>
    nekto.name) do i:=i+1;
  if i<= n then writeln('поступил')
    else writeln('не поступил');
```

```
  k:=0; {инициализация числа больных гриппом}
  for i:=1 to n do
    if otdel[i].bolezn = 'грипп' then k:=k+1;
  writeln('больны гриппом ', k, 'пациентов');
End.
```

8.2. Оператор присоединения

Обращение к элементам записи их полным именем громоздко. Этот недостаток устраняют с помощью оператора присоединения **with**.

Формат оператора:

```
with R do S1;
```

где R – имя переменной записи, а S1 – тело оператора присоединения. Внутри него выборка поля переменной R может быть обозначена просто именем этого поля.

Пример 3. Программу примера 2 записать с использованием оператора **with.**

```
Program Prog8_2;
  const n=55;
  type
  pacient = record
    name, bolezn:string[40];
    wozrast:byte;
    sem_pol:char
  end;
  var
  nekto: pacient;
  otdel: array[1..n]of pacient; {массив записей}
  i, k: integer;
Begin
  writeln('Введите список отделения');
  for i:=1 to n do
  with otdel[i] do begin
    readln(name, bolezn);
    readln(wozrast, sem_pol);
  end;
  writeln('Введите фамилию пациента');
  readln(nekto.name); i:=1;
  while (i<= n) and (otdel[i].name <>
    nekto.name) do i:=i+1;
  if i<= n then writeln('поступил')
    else writeln('не поступил');
```

```
  k:=0; {инициализация числа больных гриппом}
  for i:=1 to n do
  with otdel[i] do
    if bolezn = 'грипп' then k:=k+1;
  end;
  writeln('больны гриппом ', k, 'пациентов');
End.
```

8.3. Вложенные записи

Поле записи может быть в свою очередь тоже записью. Так, если в примере 3 описание записи дополнить полем `Date` со сведениями о дате рождения, получим следующее:

```
type
  pacient = record
  name, bolezn:string[40];
  wozrast:byte;
  date: record     {вложенная запись}
    day:1..31;
    month:0..12;
    year:word
  end;
  sem_pol: char
end;
```

При такой структуре записи обращение к ее компонентам `day, month, year` будет иметь следующий вид:

```
  nekto.date.day:=13;
  nekto.date.month:=9;
  nekto.date.year:=1988;
```

Используя оператор **with**, запись можно упростить.

```
with nekto, date do begin
  name: = 'Мария ';
  wozrast:= 20;
  bolezn := 'грипп ';
  day:=13;
  month:=9;
  year:=1988;
  sem_pol:='ж';
end;
```

Допустимо также включение между словом **do** и **begin** других операторов:

```
with nekto, date do
  if k=1 then begin … end;
begin
…
end
```

Контрольные вопросы:

1. Как описывается переменная типа запись?
2. Как обратиться к полям записи?
3. Какие действия допустимы над компонентами записи?
4. Что такое вложенная запись? Как она описывается?
5. Как сократить имя для обращения к полям записи?
6. В чем отличие записей от массивов?

Лабораторная работа № 8

Задание. Опишите информационный объект с помощью записи. Введите набор исходных данных и сохраните в файле на диске. Составьте программу, в которой исходные данные читаются из файла, производится их соответствующая обработка, результат выдается на экран.

Варианты заданий:

1. Дана экзаменационная ведомость с фамилиями студентов, номером группы, номером зачетной книжки, оценкой за экзамен. Выдайте список задолжников для данной группы. Распечатайте всю информацию в виде таблицы.

2. Составить список своей подгруппы. Для каждого человека указать: фамилию, год рождения, пол, место жительства (город/район, улица/деревня, дом). Распечатать этот список в виде таблицы, а также фамилии самой молодой девушки и самого молодого юноши.

3. Описать группу учеников, включив следующие данные о них: имя, пол, рост, возраст. Распечатать эти данные в виде таблицы и выдать имя и рост самого высокого мальчика, самой маленькой девочки и средний рост группы.

4. В пресс-центре выставки программных средств хранятся данные о каждом экспонате: название, назначение, автор, количество заявок на него. Распечатать данную информацию для 10–12 экземпляров. Выявить трех кандидатов для награждения призами (по числу заявок) и распечатать данные о них.

5. В больнице имеется список больных, каждый из которых характеризуется данными: фамилия и инициалы, номер палаты, диагноз. Требуется вывести на печать перечень больных по каждой палате и отдельно по каждому диагнозу.

6. Дана ведомость абитуриентов, сдавших вступительные экзамены, которая содержит фамилии абитуриентов и полученные ими оценки. Необходимо определить средний балл группы и распечатать список абитуриентов, имеющих средний балл выше среднего в группе.

7. Информация о студентах вводится в форме: фамилия, имя, возраст, пол, курс. Распечатайте данные 10–15 записей, а также номер курса, на котором наибольший процент мужчин.

8. В справочной аэропорта хранится расписание вылета самолетов на следующие сутки. Для каждого рейса указаны его номер, тип самолета, пункт назначения, время вылета. Определить все номера рейсов, типы самолетов и времена их вылета для заданного пункта назначения.

9. Информация о жителях различных городов записана в массив, содержащий их фамилии и адреса (город, улица, дом, квартира). Составить программу «Ирония судьбы», которая печатает фамилии двух (любых) жителей, живущих в разных городах по одинаковому адресу.

10. Дана записная книжка с фамилиями, инициалами и телефонами. Составить программу, позволяющую по фамилии, введенной с клавиатуры, выдать номер телефона, и наоборот: по номеру – фамилию. Если запрос вводится об отсутствующей в книжке записи, то сообщить об этом.

11. В предвыборной кампании проводится регистрация кандидатов в депутаты. Каждый кандидат, подавая заявление на регистрацию, указывает номер округа, в котором он собирается баллотироваться, наименование партии, которую он, представляет, свой возраст и профессию. Пресс-служба центральной избирательной комиссии выдает информационный бюллетень, в котором приводит следующую информацию: число поданных заявлений на регистрацию кандидатов каждой политической партии и средний возраст кандидатов от каждой политической партии. Написать соответствующую программу.

12. Информация об итогах сессии подгруппы задана в виде: ФИО, предметы и полученные оценки. Выдать на печать эти данные в форме ведомости, а также название предмета, который был сдан лучше всего.

13. Имеется список учета нуждающихся в улучшении жилищных условий. Каждая запись этого списка содержит

фамилию, имя, отчество и дату постановки на учет. Список упорядочен по дате постановки на учет. Известно число квартир, выделяемых по данному списку в течение года. Рассчитать, какое количество лет в среднем необходимо ожидать получения квартиры, и вывести на экран весь список с указанием ожидаемого года получения квартиры.

14. В библиотеке имеется список книг. Каждая запись этого списка содержит фамилии авторов, название книги, год издания. Определить, имеются ли в данном списке книги, в названии которых встречается некоторое ключевое слово (например, «программирование»). Если имеются, то выдать на печать фамилии авторов, название и год издания всех таких книг. Ключевое слово ввести с клавиатуры.

15. В бюро по занятости населения (трудовой бирже) ведется список вакантных рабочих мест на предприятиях города. Каждая запись такого списка содержит следующую информацию: наименование организации, местоположение организации (расстояние в км от центра города), наименование должности, требуемая квалификация (разряд или образование), требуемый стаж работы по специальности, заработная плата в месяц, наличие социального страхования (да или нет), продолжительность ежегодного оплачиваемого отпуска. Клиент бюро вводит информацию о своей квалификации и требования (например, максимальная удаленность от центра города). Написать программу, которая бы распечатывала для каждого клиента список рабочих мест в соответствии с его требованиями.

9. МОДУЛИ

Цель: формирование навыков модульного построения программ и умений их реализации в Паскале.

Для размещения в памяти большой программы может не хватить одного сегмента памяти (его размер – до 64 кбайт). Поэтому разработчики Турбо Паскаля для устранения этих недостатков включили в язык механизм так называемых модулей, которые размещаются в разных сегментах. Модули позволяют создавать библиотеки процедур и функций, которые можно подключать к разным программам.

Модуль – это автономно компилируемая программная единица, включающая в себя специальным образом оформленную библиотеку описаний (типы, константы, переменные, процедуры и функции).

Модуль может быть запущен только из программы или из другого модуля. Он компилируется независимо от использующей их программы. Результатом компиляции модуля является файл с тем же именем и расширением имени .TPU, который заносится на диск.

Общий объем модульной программы может быть много более 64 кбайта, но каждый .TPU-файл не может превышать этот предел.

9.1. Структура модуля

Модуль можно разделить на несколько разделов: заголовок; интерфейсная часть; исполняемая часть; инициализационная часть.

```
UNIT <имя>          {заголовок модуля}
interface           {интерфейсная часть}
  uses <список>     {используемые при объявлении модули}
  label             {блок объявлений библиотечных меток}
  const             {блок объявлений библиотечных констант}
  type              {блок объявлений библиотечных типов}
  var               {блок объявлений библиотечных переменных}
  <заголовки доступных процедур и функций с указанием
  параметров>
implementation      {секция реализации}
  uses              {используемые при реализации модули}
  label             {блок объявлений внутренних меток}
  const             {блок объявлений внутренних констант}
```

```
  type                  {блок объявлений внутренних типов}
  var                   {блок объявлений внутренних переменных}
  <заголовки без параметров и тела процедур и функций>
Begin                   {секция инициализации}
<операторы>
End.
```

Все блоки, составляющие разделы интерфейса, реализации и инициализации необязательны и могут отсутствовать.

Заголовок модуля состоит из зарезервированного слова **unit** и следующего за ним имени модуля. Это имя должно совпадать с именем дискового файла, в который помещается исходный текст модуля. Если, например, имеем заголовок **unit** `Stat;` то дисковый файл должен называться `Stat.pas`.

В интерфейсной части содержатся объявления всех глобальных объектов модуля (типов, констант, переменных и подпрограмм), которые должны стать доступными любой программе и/или другим модулям. При объявлении библиотечных подпрограмм в интерфейсной части указывается только их заголовок.

В разделе реализации можно использовать все объекты, описанные в разделе **interface.** Он, как правило, содержит описания подпрограмм, объявленных в интерфейсной части. Описанию подпрограммы, объявленной в интерфейсной части модуля, в исполняемой части должен предшествовать заголовок, в котором можно опускать список формальных переменных. В разделе реализации могут объявляться локальные для модуля объекты – типы, константы, переменные и т.п. Они будут глобальными по отношению к подпрограммам этого раздела и инициирующей части. В программе, подключающей модуль, объявленные при реализации данные и типы недоступны.

Завершает модуль инициирующая часть. Она может отсутствовать вместе с начинающим ее словом **Begin**. Тогда сразу после раздела реализации следует признак конца модуля **End.**
(с точкой). В инициирующей части размещаются исполняемые операторы, которые выполняются до передачи управления основной программе и обычно используются для подготовки ее работы. Обычно в ней выполняется присваивание начальных значений библиотечным переменным, открытие файлов и т.п. Инициализация происходит только при работе программы. При подключении модуля к модулю инициализации не происходит.

В приведенном ниже примере использования модуля раздел инициализации отсутствует, так как в нем нет необходимости.

Пример 1**.** Составьте модуль из процедур сложения и вычитания дробей. Напишите программу с примерами использования различных процедур модуля.

```
unit myunit;  {дисковый файл “myunit.pas” }
interface     {интерфейсная часть }
  type drob=record a,b:integer end; {тип дробь}
  procedure print(x:drob); {вывод на экран}
  procedure minus(x,y:drob;var s:drob); {вычитание}
  procedure plus(x,y:drob;var s:drob); {сложение}
implementation {раздел реализации модуля}
  procedure print;
  begin
    with x do begin
      if (a*b>=0) then begin a:=abs(a);b:=abs(b)
      end
      else begin a:=-abs(a);b:=abs(b)end;
      if b<>0 then
        if a<>0 then
          if b=1 then write(a)
          else write(a,'/',b)
        else write(0)
      else write('unknown');
    end;
  end;
{процедура сложения}
  procedure plus(x,y:drob;var s:drob);
    var q,w:integer;
  begin
    w:=x.b*y.b;q:=x.a*y.b+y.a*x.b;
    s.a:=q;s.b:=w;
  end;
  procedure minus(x,y:drob;var s:drob);
    var q:drob;
  begin
  {разность определяется через сложение}
    q:=y;q.a:=-q.a;
    plus(x,q,s);
```

```
  end;
end.
Program Prog9_1; {программа, использующая модуль}
  uses crt, myunit; {подключение модулей}
  var q,w,e,r:drob;
begin
  clrscr;
{инициализация переменных}
  q.a:=1;q.b:=3; {q=1/3}
  w:=q;w.a:=2;   {w=2/3}
  plus(q,w,e);minus(q,w,r); {e=q+w,r=q-w}
{вывод результата на экран}
  print(q); write('+'); print(w); write('=');
  print(e); writeln;
  print(q); write('-'); print(w); write('=');
  print(r);
  readln;
end.
```

Контрольные вопросы:

1. Как программируются циклические алгоритмы с явно заданным числом повторений цикла?
2. В чем отличие модуля от программы и подпрограммы?
3. Какие основные разделы модуля?
4. Каково назначение описаний в интерфейсной части модуля и в разделе реализации?
5. Какой порядок создания модуля в TURBO PASCAL? Как использовать готовый модуль в паскаль-программе?

Лабораторная работа № 9

Задание. Составьте модуль из подпрограмм, указанных в варианте. Напишите программу с примерами использования модуля.

Варианты заданий:

1. Процедуры нахождения площадей произвольного треугольника разными способами: по формуле Герона, по основанию и высоте, по двум сторонам и углу между ними.
2. Функция, определяющая первое вхождение заданного

символа в строку, и процедура замены в строке всех вхождений заданного символа, кроме первого, на другой заданный символ.

3. Процедуры нахождения длин векторов и угла между векторами на плоскости.

4. Функции нахождения числа четных и нечетных чисел одномерного массива, а также ввода и вывода массива.

5. Процедуры ввода, вывода целочисленной квадратной матрицы, нахождения номеров строк, все элементы которых делятся на 3 без остатка.

6. Процедуры ввода, вывода, поиска суммы отрицательных элементов одномерного массива.

7. Функции определения високосных лет, числа дней в месяце и проверки правильности обозначения даты. Тип даты опишите в модуле как запись, при этом месяц опишите как перечисляемый тип пользователя.

8. Процедуры удаления лишних пробелов из строки символов, разделения введенного текста на отдельные слова и занесение их в строковый массив, который распечатывается (каждое слово на новой строке).

9. Процедуры создания файла из записей, которые содержат сведения о студентах: фамилия, имя, оценки последней сессии, нахождения среднего балла для каждого студента, выдачи списка фамилий с указанием среднего балла.

10. Процедуры ввода, вывода матрицы произвольного размера и подсчёта количества строк с нулевыми элементами.

11. Процедуры выделения из текста слов и функции, определяющей сколько раз слово встречается в тексте.

12. Процедуры вычисления числа перестановок из n элементов по формуле $P_n = n!$, числа размещений из n элементов по m по формуле $A_n^m = \frac{n!}{(n-m)!}$, числа сочетаний из n элементов по m по формуле $C_n^m = \frac{n!}{m!(n-m)!}$

13. Процедуры действий с комплексными числами (сложение, вычитание, произведение, деление). Комплексное число описать как запись. Описание типа включить в модуль.

14. Процедуры формирования массива строк, каждый элемент которого может быть интерпретирован как целое двоичное число без знака, процедуры преобразования представления значений элементов массива в десятичную систему счисления.

15. Функций, не реализованных в Паскале: *tg, ctg* и целой степени действительного числа.

10. ССЫЛКИ, ДИНАМИЧЕСКИЕ ПЕРЕМЕННЫЕ И СТРУКТУРЫ

Цель: получить практические навыки использования указателей и динамических структур данных с помощью средств Turbo Pascal.

10.1. Распределение памяти при выполнении программ

Адресное пространство компьютера при запуске программы (EXE-файла) Турбо Паскаля составляет 1 Мбайт и состоит из сегментов по 64 Кбайт. Первые 256 байт памяти отводятся под префикс структуры программы, который содержит служебную информацию о программе. После префикса начинаются сегменты кода и модулей, которые содержат рабочий код основного блока программы, рабочий код системного модуля, рабочие коды подключаемых модулей. После них располагается сегмент данных, где хранятся статические глобальные переменные основного блока и все типизированные константы, включая локальные. За сегментом данных следует область стека. В ней располагаются локальные переменные и параметры-значения процедур и функций во время их работы по вызову. Выше стека отводится память под буфер для работы оверлеев – перекрывающихся частей программ. Еще выше располагается динамически распределяемая область памяти, называемая областью кучи или просто кучей (Heap-областью). Объемом распределяемой динамической памяти можно управлять с помощью директивы компилятора $M.

В динамической области размещаются объекты, память для которых отводится и освобождается непосредственно по ходу выполнения программы. Такие данные, размер которых задается во время выполнения программы, называются динамическими. Для объявления динамических данных в Паскале используется ссылочный тип, называемый еще типом-указателем. Значением переменных ссылочного типа является адрес ячейки памяти. Адрес занимает четыре байта и хранится в виде двух слов, одно из которых определяет сегмент, второе – смещение. В Паскале имеется стандартный тип – указатель и ссылочные типы, определяемые программистом.

10.2. Указатель

Величины стандартного типа `pointer` предназначены для хранения адресов данных произвольного типа. Описание указателя стандартного типа:

var <идентификатор> **:** pointer;

Такие указатели называются нетипизированными. Значение указателя не может быть в явном виде выведено на экран или печать. Для работы с указателями вводится специальный набор функций:

`Addr(X):Pointer`	Ссылка на начало объекта `X` в памяти. Аналогом этой функции является операция `@X`
`Seg(X):`**`Word`**	Сегмент, в котором хранится объект `X`
`Ofs(X):`**`Word`**	Смещение в сегменте для объекта `X`
`Ptr(S,O:`**`Word`**`):Pointer`	Ссылка на место в памяти, заданное значениями смещения `O` и сегмента `S`
`SizeOf(x):`**`Word`**	Размер объекта `X` в байтах

Указатели могут обмениваться значениями с помощью оператора присваивания и сравниваться с помощью операций отношения = или <> (не равно). Сравнение для указателей – ненадежная операция, так как два указателя, содержащие один и тот же адрес памяти, но записанный в них разными способами, считаются различными.

Пример 1.

```
Var p,q : pointer; x:string;
……….
p:=addr(x); q:=@p;          {p=q}
Writeln('Сегмент ', Seg(p), 'смещение ',Ofs(p));
```

Так как значение указателя состоит из двух слов `(Word)`, хранящих сегмент и смещение, можно вывести их в отдельности, используя функции `Seg` и `Ofs`.

В Паскале есть предопределенная константа **`nil`** типа `Pointer`, соответствующая адресу 0000:0000 (пустая ссылка). Если указателю присвоено значение **`nil`**, то этот указатель ни на какие данные не ссылается.

Чтобы обратиться к данным, находящимся по адресу, содержащемуся в указателе, используется символ ^, который ставится сразу после имени указателя. Эта операция называется операцией разыменования. Суть ее состоит в переходе от указателя к значению, на которое он указывает. Пусть p – указатель, то p=addr(p^), и, например, p^='строка'.

10.3. Ссылочные типы

Программист может определить ссылки на данные конкретного типа.

Формат описания ссылочного типа:

```
type <имя ссылочного типа>=^<имя базового типа>;
```

Ссылочные переменные указывают на объекты базового типа, определяя динамические переменные.

Пример 2.

```
type                {базовые типы }
  dimT=array[1..10000]of real;
  recT=record a,b:integer end;
var                 {ссылочные типы }
  dimP=^ dimT;
  recP=^recT;
  xP=pointer;
  i, j:^integer;
```

Операция разыменования состоит в переходе от ссылочной переменной к значению объекта, на который она указывает.

Пусть, например, i^=2, j^=5, тогда:

j^:=i^; означает присваивание значения, т.е. j^=2, при этом адреса, записанные в i и j не изменились;

j:=i; здесь адрес, на который указывает i записывается в ссылочную переменную j.

Доступ к полям и элементам массива, например, dimP^[I], recP^.a.

Присваивание dimP:= recP запрещено, поскольку dimP и recP указывают на разные типы данных. Это ограничение не распространяется на нетипизированные указатели. Можно совмещать разные типы, используя указатели, например, xP:= dimP; recP:= xP.

Пример 3.

```
Program Prog10_1; {программа демонстрации наложения типов}
  type t : array [1..2] Of char;
  var a :^t; {ссылка на массив }
  i : integer;
Begin
  i:=19022;
  a:=addr(i); {присваиваем a значение адреса переменной i }
  write(a^[1],a^[2]); {выводим массив }
End.
```

Переменная i занимает в памяти 2 байта и в 16-ричном коде равна 4A4E. После передачи адреса переменной a, эти два байта представляют два символа массива: младший байт 4E (78 десятичное) соответствует элементу a^[1], старший байт 4A (74 десятичное) соответствует элементу a^[2]. Результат: N – код 78 и J – код 74.

10.4. Размещение динамических переменных в куче

10.4.1. Процедуры для работы со ссылками

Под динамические переменные память выделяется в куче во время выполнения программы с помощью процедур new или getmem. После окончания работы с динамическими переменными память можно возвратить в кучу.

New(**var** p: тип указателя); – создает новую динамическую переменную и устанавливает на нее указатель p. Параметром обращения к этой процедуре является типизированный указатель. В результате обращения указатель приобретает значение, соответствующее адресу, начиная с которого, можно разместить данные.

Dispose(**var** p: тип указателя); – освобождает память, занимаемую динамической переменной.

Getmem (**var** p:pointer, Size:) – создает новую динамическую переменную размером Size и устанавливает на нее указатель p.

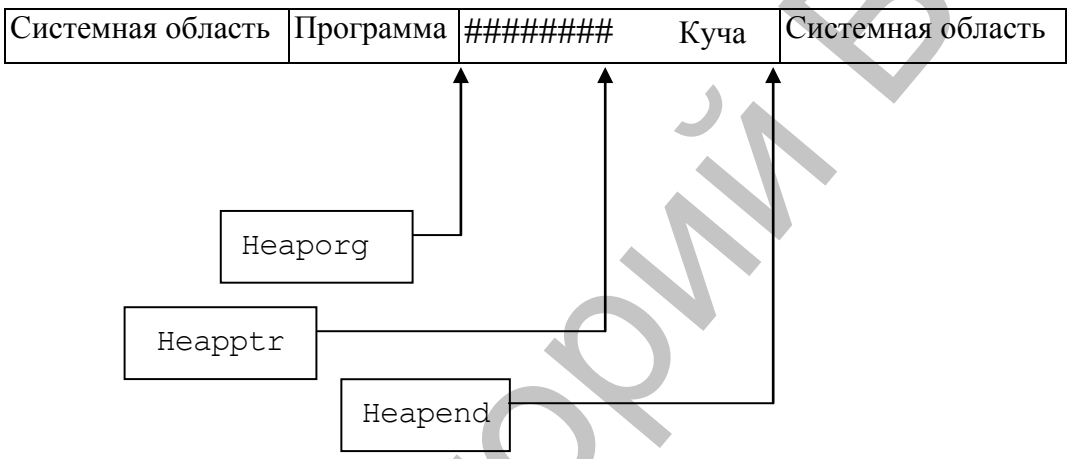
Freemem (**var** p:pointer, Size:word) – освобождение памяти, занятой динамической переменной p размером Size байт.

Процедуры Getmem и Freemem можно использовать для работы с нетипизированными указателями.

Перед использованием указателей им всегда нужно присваивать значения. Если разыменовывается указатель, которому еще не присвоено значение, то считанные из него данные могут представлять собой случайные биты, а присваивание значения указываемому элементу может затереть другие данные, программу или даже операционную систему.

10.4.2. Динамическая область памяти

Начало кучи хранится в стандартной переменной `Heaporg`, конец – в переменной `Heapend`. Текущую границу незанятой динамической памяти указывает указатель `Heapptr`.



При очередном обращении к процедуре `new` или `getmem` система отыскивает наименьший свободный фрагмент, в котором может разместиться требуемая переменная. Перед выделением большого объема в динамически распределяемой памяти полезно убедиться, что такой объем памяти имеется. Для этого можно использовать функцию `MaxAvail`, которая возвращает размер наибольшего доступного блока непрерывной памяти в динамически распределяемой области. Существует еще функция `MemAvail`, возвращающие общее число байтов, доступных для распределения в динамической памяти. Первоначально при запуске программы `MaxAvail` равно `MemAvail`, поскольку вся динамически распределяемая область памяти является доступной и непрерывной.

Освободить фрагмент кучи можно при использовании процедур `Mark` и `Release`. Процедура `Mark`**`(var`** `p : pointer`) запоминает состояние динамической памяти в тот момент, когда эта процедура вызывается. В указателе `p` сохраняется адрес первого байта свободной области памяти. Далее можно несколько раз выделить память. Процедура `Release`**`(var`** `p :`

pointer) возвращает динамическую память в состояние, которое было запомнено ранее при помощи процедуры Mark.

10.4.3. Использование переменных ссылочного типа

Наиболее часто ссылочные переменные используются в следующих случаях:

1. Использование одной области данных для представления разными типами.

2. Программа должна работать с большими объемами данных, под которые нет необходимости сразу выделять память.

3. Программа во время компиляции использует данные, для которых заранее неизвестен необходимый объем памяти. Если расположить переменные в динамически распределяемой области памяти во время выполнения программы, то для хранения данных будет выделено столько байтов памяти, сколько потребуется.

4. В программе необходимо использовать массив большой размерности, не помещающийся в сегменте стека.

5. Возможность построения объектов со сложной, меняющейся структурой. Примерами таких структур являются списки, стеки, деревья

Пример 4**.** Дано 10000 целых чисел a_1, a_2, ..., a_{10000}, значения которых лежат в диапазоне от -9000 до 9000, и 9000 вещественных чисел b_1, b_2, , b_{9000}. Вывести минимальное из чисел a_1, a_2, ..., a_{10000}. Вывести число b_k, номер которого k равен абсолютному значению этого минимального.

```
program Prog10_2;
  type ar=array[1..10000] of integer; {определяем массив}
  br=array[1..9000] of real; {определяем массив}
  var a:^ar; { a указатель на массив типа ar }
  b:^br; { b указатель на массив типа br }
  k,i:integer;
begin
  new(a); {выделение памяти под массив a }
  randomize;
  k:=9000;
  for i:= 1 to 10000 do begin
    a^[i]:=random(18000)-9000; {формируем массив a с помощью генератора случайных чисел}
    if a^[i]< k then k:= a^[i]; {находим минимальное }
```

```
  end;
  dispose(a); {освобождаем память}
  new(b); {выделение памяти под массив b }
  for i:= 1 to 9000 do
  readln(b^[i]);
  writeln('мин a ',k,' b[k]= ', b[abs(k)])
  dispose(b); {освобождаем память}
end.
```

Пример 5**.** Найти сумму элементов массива d, состоящего из 200 строк и 200 столбцов.

Под элементы двумерного массива такого размера потребуется 200×200×6 байтов памяти, что составит более 234К. Нужно реорганизовать массив так, чтобы он распался на части, не превышающие по отдельности 64К (сегмент данных).

```
program Prog10_3;
  type d200=array[1..200] of real; {определяем массив}
  dPtr =^d200;{объявлен тип ссылки на строку}
  dPtr200=array[1..200] of dPtr;{тип для массива указателей}
  var d:dPtr200; {объявили массив указателей, каждый элемент которого указывает на одномерный массив d200}
  s:real; i, j:integer;
begin
  randomize;
  s:=0;
  for i:= 1 to 200 do begin
    new(d[i]); {выделили память под строку из 200 элементов, на которую указывает ссылка d[i]}
    for j:= 1 to 200 do begin
      d[i]^[j]:=random(20); {формируем массив с помощью генератора случайных чисел}
      s:=s+d[i]^[j]; {находим сумму }
    end;
    dispose(d[i]); {освобождаем память}
  end;
  writeln('s= ',s)
end.
```

Пример 6**.** Создание массива с переменной верхней границей[1].

```
program Prog10_4;
  type d=array[1..2] of real;
  var a:^d; {a указатель на массив типа d}
  n,i:word;
begin
  writeln('введите размер массива');
  readln(n);
  getmem(a,n*6);  {выделение памяти под n вещественных чисел}
  for i:= 1 to n do a^[i]:=sqr(i);
  freemem(a,n*6); {освобождаем память}
end.
```

10.5. Динамические структуры данных

10.5.1. Списки

Список – это набор записей, каждая из которых имеет поле данных и ссылку на следующую запись в списке. Последний элемент списка содержит значение **Nil**, т.е. уже ни на что не ссылается. Начало списка формирует переменная ссылочного типа, содержащая адрес первого элемента списка. Поле данных еще называют информационной частью списка**.**

Указатель в списке должен быть типизированным. Базовым типом для него является тот же тип данных, что и тип информационной части списка.

Обозначим тип элемента списка идентификатором `Elem`, а ссылочный тип на элемент списка – идентификатором `Uk`. Тогда формат описание списка следующий:

```
type
Uk = ^Elem; {описание типизированного указателя}
 Elem = Record  {описание базового типа}
   <описание информационной части>
   next : Uk;
end;
var p, q : Uk;
```

В Паскале допускается описывать сначала типизированные указатели, а затем их базовый тип.

---

[1] Проверка значений индекса массива должна быть выключена.

Пример 7**.** Рассмотрим создание списка, информационная часть которого состоит из одного поля типа **integer**. Построим список из трех элементов, содержащих числа 5, 20 и -10.

Список обычно задается указателем на свой первый элемент. Назовем этот указатель p. Значением переменной p в процессе построения всегда будет ссылка на первый элемент уже построенной части списка. Переменная q будет использоваться для выделения памяти под размещение новых элементов списка. Выполнение оператора p := **nil** приводит к созданию пустого списка.

```
program Prog10_4;
  type
  Uk = ^Elem; {описание типизированного указателя}
  Elem = Record  {описание базового типа}
    x: integer;
    next : Uk;
  end;
  var p, q: Uk;
begin
  new(p); {размещение первого элемента}
  p^.x:=5; {задаем значение первого элемента списка}
  p^.next:=nil; {следующий элемент отсутствует}
        {создание второго элемента}
  q:=p;           {q указывает на первый элемент}
  new(q^.next); {выделение памяти под второй элемент}
  q:=q^.next;  {q теперь указывает на второй элемент}
  q^.x:=20;     {задаем значение второго элемента списка}
  q^.next:=nil; { следующий элемент отсутствует }
        {создание третьего элемента}
  new(q^.next); {выделение памяти под третий элемент}
  q:=q^.next;    {q указывает на третий элемент}
  q^.x:=-10;      {задаем значение третьего элемента списка}
  q^.next:=nil; { следующий элемент отсутствует }
end.
```

Пример 8**.** Сформировать список из n целых чисел. Значения списка вводить с клавиатуры. Вывести элементы списка на экран.

В начало списка добавим заглавный элемент. Информационная часть заглавного элемента или не используется вообще, или используется для специальных целей. Например, в случае целых чисел она может содержать число, равное количеству элементов списка. Добавление заглавного элемента в

список приводит к тому, что у всех элементов имеется предшественник, и действия по включению новых элементов в список и исключению из него проводятся одним способом для всех элементов.

```
program Prog10_5;
  type
  Uk = ^Elem; {описание типизированного указателя}
  Elem = Record  {описание базового типа}
    x: integer;
    next : Uk;
  end;
  var p, q, y: Uk;
  i:integer;
begin
  writeln('введите число элементов');
  readln(n);
  new(p); new(q);  {размещение заглавного и первого
  элементов}
  p^.next:=q; {в p заносим ссылку на первый элемент}
  writeln('введите элементы списка');
  for i:= 1 to n do begin
    readln(q^.x);   {вводим элементы списка}
    new(y);         {создание очередного элемента}
    q^.next:=y;     {в q заносим ссылку на очередной элемент}
    y^.next:=nil;   { следующий элемент отсутствует }
    q:=y;          {q теперь указывает на следующий элемент}
  end;
          {вывод элементов списка}
  q:= p^.next; { q указывает на первый элемент}
  while q^.next<>nil do begin {вывод элемента списка
    write(q^.x:4); q:=q^.next;  и переход к следующему }
  end;
end.
```

К основным операциям над списками относятся, кроме перехода от элемента к следующему элементу, включение нового элемента в список и исключение элемента из списка.

Пусть требуется включить в данный список после элемента, ссылка на который является значением переменной q, новый элемент, ссылка на который – y. Для этого нужно выполнить последовательность операторов:

```
new(y);                {размещение вставляемого  элемента}
```

```
readln(y^.x);          {вводим его значение}
y^.next:=q^.next; {в адресную часть нового элемента
                   переносим из q ссылку на следующий элемент}
q^.next:=y;        {в q^.next заносим ссылку на новый элемент}
```

Для исключения элемента, следующего за тем, ссылка на который является значением переменной q, нужно выполнить последовательность операторов:

```
y:=q^.next; q^.next := q^.next^.next;
y^.next := nil;
```

Первое присваивание выполняется для того, чтобы сохранить ссылку на исключенный элемент. В этом случае исключаемый элемент останется доступным и с ним можно выполнять нужные операции, например, освободить занимаемую им память с помощью Dispose. Второе из присваиваний – исключение элемента из списка. Третье присваивание выполняется для того, чтобы из исключенного элемента по ссылке нельзя было попасть в список, из которого этот элемент исключен.

Пример 9**.** В список вставить элемент перед первым элементом, значение которого задано, и удалить после него элемент, если он не последний.

```
program Prog10_6;
  type
  Uk = ^Elem; {описание типизированного указателя}
  Elem = Record  {описание базового типа}
    x:integer;
    next : Uk;
  end;
  var p,q: Uk;
  i,i1,a:integer;{i1 – заданное значение в списке, a –
  вставляемый элемент}
       {процедура формирования списка}
procedure form(var u:uk); { u – заглавный элемент списка}
  var n,i:integer; u1,u2:uk;
begin
  writeln('введите число элементов');
  readln(n
  new(u); new(u1); {размещение заглавного и первого
  элементов}
  u^.next:=u1; {в u заносим ссылку на первый элемент}
  writeln('введите элементы списка');
  for i:= 1 to n do begin
    readln(u1^.x);   {вводим элементы списка}
    new(u2);         {создание очередного элемента}
```

```
    u1^.next:=u2;   {вносим ссылку на очередной элемент}
    u2^.next:=nil;   { следующий элемент отсутствует }
    u1:=u2;   {u1 теперь указывает на следующий элемент}
  end;
end;
{процедура поиска заданного элемента и вставки перед ним нового}
procedure vstavka (u:uk;i1,a:integer var q:uk);
{u – ссылка на начало списка, q – ссылка на искомый элемент}
  var u1,u2:uk;
begin
  u1:=u;
  while (u1^.next<>nil) and (u1^.x <>i1) do begin
    u2:=u1; u1:=u1^.next; end;{пока не найден
    нужный элемент, переходим к следующему}
  new(u1);{вставка элемента}
  u1^.x:=a; u1^.next:= u2^.next; u2^.next:=u1;
  q:=u2^.next^.next; {q – ссылка на элемент, который нужно
  удалить}
end;
      {процедура вывода списка}
procedure vv(u:uk);
  var u1,u2:uk;
begin
  u1:=u.next;
  repeat
    write(u1^.x:6);
    u1:=u1^.next;
  until(u1^.next =nil);
writeln;
end;
{процедура удаления элемента из списка, следующего за заданным}
procedure udalenie(q:uk); {q – ссылка на элемент,
который нужно удалить}
  var u1:uk;
begin
  if q^.next^.next<>nil then begin
    u1:=q^.next; q^.next:=q^.next^.next;
    dispose(u1);
  end;
end;
begin
  form(p);vv(p);
  writeln('введите данные для поиска и вставки');
  readln(i1,a);
  vstavka(p,i1,a,q);
  udalenie(q);
```

```
  vv(p);
end.
```

Кроме рассмотренных списков, которые являются линейными однонаправленными списками, существуют двунаправленные и кольцевые. Двунаправленный список называется очередью. Очередь – линейный список, элементы в который добавляются только в конец, а исключаются только из начала списка. Каждый элемент структуры содержит указатель на следующий элемент и еще на предыдущий. Для очереди определяют два указателя: на первый и на последний элемент списка. Начало списка называют головой очереди, а конец списка – хвостом очереди.

В кольцевом списке для последнего элемента следующим является первый, а если список двунаправленный, то для первого предыдущим является последний. При просмотре кольцевого списка надо переходить к следующему элементу до тех пор, пока не достигнем указателя на первый элемент. Поэтому первый элемент должен обрабатываться отдельно, а просмотр при помощи цикла следует начинать со второго элемента.

10.5.2. Стек

Стек – линейный список, в котором добавления и исключения элемента производятся с одного конца, называемого вершиной стека. Другие операции не определены. Стек оптимален для случаев, когда требуется просчитать и запомнить большое число структур данных, а потом обработать их в обратном порядке.

Работа со стеком осуществляется через указатель стека. При выполнении загрузки элемента в стек, данные записываются на место, определяемое указателем стека, а указатель стека изменяет свое состояние и задает следующую свободную ячейку блока памяти. При извлечении элемента из стека указатель стека возвращается назад на один шаг.

Пример 10**.** Сформировать стек из целых чисел. Затем вывести содержимое стека.

```
program Prog10_7;
  uses crt;
  type
  Uk = ^Elem; {описание типизированного указателя}
  Elem = Record  {описание базового типа}
    x: integer;
    next : Uk;
  end;
```

```
  var p, q: Uk;    { p – указатель стека}
  i:integer;
begin
  clrscr;
  writeln('введите число элементов');
  readln(n);
  new(p); { выделение памяти под указатель }
  p:=nil;
  writeln('введите элементы списка');
  for i:= 1 to n do begin      {формирование стека}
    new(q);                  {выделение памяти для элемента}
    readln(q^.x);            {вводим его значение}
    q^.next:=p;    {в адресную часть нового элемента
                    переносим адрес указателя стека}
    p:=q;       {изменяем значение указателя стека }
  end;
            {вывод элементов стека}
  q:=p;
  while q <>nil do begin
  writeln(q^.x); p:=q^.next;
  dispose(q);
  end;
end.
```

10.5.3. Деревья

Структура «дерево» является обобщением линейного списка. В списке каждый узел содержит указатель на другой узел. В дереве каждый узел содержит несколько указателей на несколько узлов. Если указателей два (правый и левый), такое дерево называется *бинарным* (рис. 1). Один из указателей может быть равен **nil**.

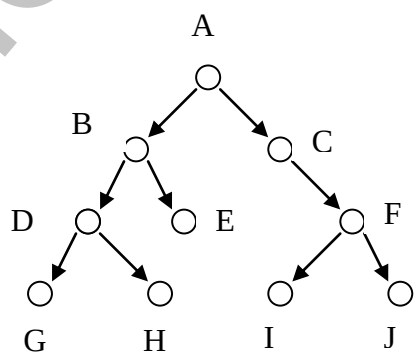


Рис. 1.

Начальная точка дерева называется корневым узлом. У корневого узла нет входящих в него ветвей, есть только

исходящие. Вершина, на которую имеется указатель из другой вершины, называется потомком этой вершины, последняя, соответственно, называется предком. Если вершина не имеет потомков, она называется терминальной вершиной (листьями дерева).

В бинарном дереве целочисленных значений часто придерживаются соглашения о том, что во всех левых вершинах должны находиться меньшие числа, а в правых – большие.

Основные операции над деревьями – это занесение элемента в дерево, удаление элемента из дерева и обход дерева.

Пример 11**.** Написать программу формирования двоичного дерева поиска. Ввести с клавиатуры целые числа и вывести в порядке возрастания с числом встречаемости его во введенной последовательности. Для хранения чисел и частоты их встречаемости использовать двоичное дерево поиска.

Ключом поиска будут сами числа: меньшие будем помещать в левое поддерево, большие – в правое. Для вывода чисел в упорядоченном виде выполним обратный обход дерева.

```
Program Prog10_8;
  type tr = ^tree;
  tree = record    {описание типа дерева}
  i:integer;       {число}
  k: byte;         {частота встречаемости}
  l, r:tr;    {указатели на поддеревья}
  end;
  var t, p, q:tr; w,n,j:integer;
  f: boolean; {p, q - рабочие переменные; w -
  вводимое число; f = true, если число уже есть в
  дереве}
procedure out(t: tr); {процедура вывода дерева}
                      {t – указатель на корень дерева}
begin
  if t<>nil then begin
    out(t^.l);
    writeln(t^.i,'-',t^.k);
    out (t^.r);
  end;
end;
begin {исполняемая часть программы}
  readln(w);
  new(t);       {формирование корня дерева}
  with t^ do begin i:=w; k:= 1;
    l:=nil; r:=nil; end;
```

```
  writeln('введите количество элементов');
  readln(n);
  for j:=1 to n do begin {поиск элемента в дереве}
    readln(w);     {ввод нового слова}
    f:=false; p:=t;
    while (p<>nil) and (f=false) do begin
      q:=p; {сохранить вершину, к которой будет присоединяться
      новый элемент}
      if w<p^.i then p:=p^l {если число меньше, то левая
      ветвь}
        else if w>p^.i then p:=p^r {если число больше,
        то правая ветвь}
          else f:=true;        {число уже есть}
    end;         {конец поиска в дереве}
    if f then inc(p^k) {если число уже есть, то увеличить
    k}
      else begin        {если числа еще нет}
        new(p);       {создать новый элемент}
        with p^ do begin
          i:=w; k:=1; l:=nil; r:=nil; end;
          if w<q^.i then q^.l:=p {если меньше, то
          присоединить слева}
          else q^.r:=p:        {иначе справа}
      end;
  end;           {конец ввода чисел}
  out(t);       {вывод по возрастанию}
end.
```

Контрольные вопросы:

1. Какие объекты хранятся в сегменте данных?
2. Что представляет собой динамически распределяемая область памяти?
3. Что является значением ссылочного типа?
4. Какие операции определены для указателей?
5. Что является значением константы `nil`?
6. Приведите формат описания ссылочного типа.
7. В чем состоит операция разыменования?
8. Чем отличаются процедуры `new` или `getmem`?
9. Зачем нужно освобождать память после работы со ссылками?
10. В каких случаях используются ссылочные переменные?
11. Назовите наиболее часто используемые динамические структуры.
12. Что представляет собой список?
13. Каковы основные операции над списками?

14. Что представляет собой очередь, стек?
15. Чем отличается дерево от списка?
16. Какое дерево называется бинарным?

Лабораторная работа № 10

Задание 1. Выполнить обработку массивов в соответствии с вариантом задания. Предусмотреть описание массивов как динамических.

Варианты заданий:

1. Дан массив записей из полей типа *array[1..2] of Char,* и *Word.* Преобразовать его в массив типа *Longint.*
2. Дан массив чисел типа *Word*. Выполнить наложение каждого числа на запись с двумя полями: *Byte* и *Char*. Вывести элементы массива и соответствующие поля записи.
3. Дан указатель на массив чисел типа *Longint.* Вывести числа и, интерпретируя каждое число как массив символов, вывести соответствующие символы.
4. Преобразовать массив *n* чисел типа *Byte* в массив целых чисел типа *Integer*, расположенный в той же области памяти.
5. Дан массив записей
```
record
  x : integer;
  y : integer
end
```
Преобразовать его в массив типа *Byte*.
6. Дан массив записей из полей типа *Char, Word, Byte*. Преобразовать его в массив типа *Longint.*
7. Даны два массива чисел: первый размерности $2 \times n$ типа *Byte*, второй размерности *n* типа *Integer.* Преобразовать их в массив типа *Longint,* каждый элемент которого образован из двух последовательных элементов первого массива и одного элемента второго массива.
8. Дан массив чисел типа *Integer*. Выполнить наложение каждого числа на запись с двумя полями: *Char* и *Byte*. Вывести элементы массива и соответствующие поля записи.
9. Преобразовать массив *n* чисел типа *Char* в массив целых чисел типа *Longint*, расположенный в той же области памяти.
10. Дан массив записей
```
record
  x : longint;
  y : integer
end
```
Преобразовать его в массив типа *Byte*.

11. Даны два массива чисел: первый размерности $3 \times n$ типа *Byte*, второй размерности *n* типа C*har*. Преобразовать их в массив типа *Longint,* каждый элемент которого образован из двух последовательных элементов первого массива и одного элемента второго массива.

12. Даны два массива типа *Integer*. Преобразовать их в массив записей с полями:

a : array[1..3]of char;
b : byte;

Каждая запись соответствует элементам массива с одинаковыми индексами.

13. Дан одномерный массив чисел типа *Longint*. Преобразовать его в массив типа *String[8]*, где строке соответствуют два числа. Вывести числа и соответствующие строки.

14. Дан массив типа *Longint*. Вывести два массива, элементами которого являются значения младшего байта, а второго – старшего байта чисел данного массива.

15. Дан массив строк. Вывести строку и соответствующие коды символов. Для этого использовать массив типа *Byte*.

Задание 2. Выполнить обработку массивов большой размерности в соответствии с вариантом задания. Массивы размещать в оперативной памяти динамически. Элементы массива формировать, используя генератор случайных чисел. Строковые массивы формировать, используя коды символов. Выводить на экран исходные и результирующие массивы.

Варианты заданий:

1. По квадратной матрице построить новую «сглаженную» матрицу, значением каждого элемента которой является среднее арифметическое значений элемента и его соседей в исходной матрице.

2. Определить количество взаимообратных соседних элементов строк двумерного массива. Результат сохранить в одномерном массиве.

3. Определить количество «особых» элементов двумерного массива. Считать элемент «особым», если слева от него находятся элементы, меньшие его, а справа – большие.

4. Определить, является ли заданная квадратная целочисленная матрица магическим квадратом, т.е. такой, в которой суммы элементов во всех строках и столбцах одинаковы.

5. Определить наименьший элемент среди наибольших элементов всех столбцов заданной матрицы.

6. Разработать программу нахождения первого простого числа в матрице размерности $n \times n$.

7. Дан массив строк. Разработать программу нахождения количества строк, заканчивающихся точкой и заменить ее восклицательным знаком.

8. Разработать программу сортировки (упорядочивания) матрицы размерности $n \times n$ гак, чтобы элементы в каждой строке отсортированной матрицы располагались по возрастанию и ни один элемент в i-ой строке не был больше любого элемента в $i+1$-й строке. Сортировку выполнять над одномерным массивом из n^2 элементов, который «накладывается» на исходную матрицу.

9. Разработать программу выявления седловой точки в матрице размерности $n \times n$. Седловой точкой в матрице называют элемент, одновременно наибольший в своей строке и наименьший в своем столбце.

10. Разработать программу нахождения количества трехзначных чисел в четных строках матрицы размерности $n \times m$.

11. Дан массив символов. Разработать программу формирования из него массива строк заданной длины.

12. Дан массив строк. Разработать программу нахождения количества строк заданной длины.

13. Разработать программу, которая в матрице размерности $n \times n$ меняет местами строку, содержащую элемент с наибольшим значением, со столбцом, содержащим элемент с наименьшим значением.

14. Написать программу записи в одномерный массив строк матрицы размерности $m \times n$, которые начинаются с произвольного элемента.

15. Разработать программу нормирования матрицы размерности $m \times n$, которое заключается в том, что каждый элемент в этой матрице вычисляется на основании исходной матрицы как отношение суммы всех других элементов в его строке к сумме всех других элементов в его столбце.

Задание 3. Разработать программу формирования из заданной последовательности однонаправленного списка в куче, и его обработки в соответствии с вариантом задания.

Варианты заданий:

1. Вычислить сумму элементов последовательности из n вещественных чисел. Вставить отрицательные числа после всех

элементов, больших 2. Удалить из списка первый и последний элементы, если они отрицательные.

2. Найти наибольший элемент последовательности, состоящей из *n* вещественных чисел. Вставить перед первым элементом три нулевых. Удалить все элементы, большие 3 и меньшие 5.

3. Дана последовательность строк. Найти количество строк, содержащих данную подстроку. Перед каждой такой строкой вставить двоеточие. Удалить три элемента после первой найденной строки.

4. Найти второй по величине элемент последовательности из *n* вещественных чисел. Дополнить список пятью элементами. Удалить найденный элемент.

5. Найти наибольший среди отрицательных элементов последовательности из *n* вещественных чисел. Перед каждым числом, большим 10, вставить отрицательный элемент. Удалить *m* элементов из конца списка.

6. Найти произведение положительных элементов последовательности из *n* вещественных чисел. Вставить *m* элементов в середину списка. Удалить элемент, равный заданному числу.

7. Дана последовательность строк. Найти первую строку, заканчивающуюся на данный символ. Удалить элемент после этой строки, если он не последний. После нее вставить еще два элемента.

8. Найти наименьший элемент среди четных элементов последовательности из *n* целых чисел. Удалить элемент перед десятым элементом списка. Вставить перед элементами, совпадающими с ним, еще по два четных элемента.

9. Найти количество положительных элементов последовательности из *n* вещественных чисел. После нулевого элемента удалить *k* элементов. Вставить элемент перед десятым элементом списка.

10. Найти все делители первого, большего 20, числа в последовательности из *n* целых чисел. Вставить найденные делители перед этим числом. Удалить после этого числа все отрицательные элементы, большие $k<0$.

11. Дана последовательность строк. Найти количество упорядоченных строк. Перед каждой из них вставить какой-либо элемент. Удалить первый и последний элемент списка.

12. Найти минимальный элемент среди трехзначных чисел целочисленной последовательности. Удалить все элементы, совпадающие с ним. Вставить перед ним два нечетных элемента.

13. Разместить в памяти целочисленную последовательность. Найти все элементы, кратные числу *k*, и перед каждым из них вставить *0*. После первого элемента, кратного *k*, удалить два элемента, если он не последний.

14. Разместить в памяти целочисленную последовательность. Найти последний простой элемент списка. Вставить после него два элемента. Удалить из списка предпоследний элемент.

15. Найти наименьший по длине элемент последовательности из *n* строк. Вставить перед первым элементом три пустые строки. Удалить все элементы, длины которых большие 2 и меньшие 4.

Задание 4. Сформировать динамически из заданной последовательности стек и выполнить его обработку в соответствии с вариантом задания.

Варианты заданий:

1. Задана последовательность из *n* вещественных чисел. Вычислить сумму отрицательных элементов.

2. Дана последовательность строк. Выдать сообщение, имеется ли в стеке строка, содержащая заданную подстроку и номер этого элемента, отсчитываемого от дна стека.

3. Дана последовательность целых положительных чисел, вводимых с клавиатуры. Процесс ввода должен прекращаться, как только среди вводимых чисел появляется отрицательное число. Вывести на экран содержимое стека до заданного элемента.

4. Дана последовательность значений коэффициентов $a_0, a_1, \ldots, a_n$ многочлена $P(x)= a_0 +a_1x+a_2x +\ldots+ a_nx^n$. Вычислить значение многочлена в целочисленной точке *x*.

5. Упорядочить по возрастанию элементы целочисленной последовательности, помещенной в стек. В процессе упорядочивания элементы стека перемещаться не должны.

6. Дана последовательность символов в виде отдельных слов, вводимых с клавиатуры. Каждое слово, помещённое в стек, следует вывести на экран; при этом порядок вывода символов в каждом слове должен быть обратным по сравнению с последовательностью их ввода.

7. Выдать сообщение, имеется ли в стеке заданный элемент, и номер этого элемента от вершины стека.

8. Дана последовательность целых положительных чисел. Преобразовать ее путем удаления всех четных чисел (в процессе преобразования стека его элементы в оперативной памяти перемещаться не должны).

9. Дана последовательность строк, вводимых с клавиатуры до появления пустой строки. Освободите стек наполовину.

10. Дана последовательность символов, вводимых с клавиатуры до появления точки. Измените направления ссылок в стеке на противоположное.

11. Вычислить значение выражения следующего вида: $x_1x_n + x_2x_{n-1} + \ldots + x_nx_1$. При этом значения x_1, x_2, ..., x_n вводятся с клавиатуры.

12. Определить, симметричен ли произвольный текст любой длины. Текст всегда должен оканчиваться точкой. Эту задачу рекомендуется решать с помощью двух стеков, в первый стек следует поместить весь текст, затем во второй стек перенести его половину так; чтобы последний символ текста находился на дне стека, далее путем поэлементного сравнения этих стеков получить ответ на вопрос о симметричности текста.

13. Разработать программу слияния двух стеков, содержащих возрастающую последовательность целых положительных чисел, в третий стек так, чтобы его элементы располагались также в порядке возрастания.

14. Добавить к стеку, содержащему возрастающую последовательность целых положительных чисел, новый элемент так, чтобы порядок возрастания в стеке не изменялся. При добавлении нового элемента в стек другие его элементы перемещаться не должны.

15. Выполнить сложение соответствующих элементов двух стеков разной глубины и поместить суммы в третий стек. Сложение закончить, когда будет достигнуто дно одного из стеков.

Задание 5. Написать программу формирования двоичного дерева и подсчета количества его листьев.

11. ТЕКСТОВЫЙ И ГРАФИЧЕСКИЙ РЕЖИМЫ РАБОТЫ

Цель: использование возможностей текстового режима для разработки интерфейса программ и представления данных в удобной для пользователя форме, формирование знаний о возможностях графического способа представления информации в Паскале, выработка практических навыков создания графических изображений средствами языка.

Турбо Паскаль имеет развитые программные средства текстового и графического режимов и поэтому перспективным является их использование для создания удобного и наглядного интерфейса пользователя.

11.1. Текстовый режим

Для того чтобы эффективно использовать текстовый режим для создания интерфейса, необходимо подключение стандартного модуля CRT. Модуль CRT имеет специальные процедуры и функции управления выводом на экран информации, некоторые из них приведены в табл. 3[2].

Таблица 1

`ClrScr;`	очистка экрана и перемещение курсора в верхний левый угол
`GotoXY(X,Y:byte);`	управление переводом курсора в заданную позицию (знакоместо) экрана
`TextBackground (Color:byte);`	задание цвета фона
`TextColor(Color:byte);`	задание цвета символов
`Sound/NoSound;`	включение/выключение внутреннего звукового устройства
`Delay(ms.word);`	включение задержки выполнения программы на заданное число миллисекунд (ms)
`HighVideo;`	установка повышенной яркости символов
`Windows(X1,Y1,X2,Y2);`	задание размера текущего окна

[2] См. другие подпрограммы в справке Турбо Паскаля.

KeyPressed:boolean	Возвращает значение true, если на клавиатуре нажата клавиша и false в противном случае
ReadKey:char	считывает символ из буфера клавиатуры, не отображая его на экране. Если буфер пуст, ожидает нажатия клавиши

При работе с окнами следует учитывать, что активным является всегда одно окно. По умолчанию окном является весь экран. Все координаты экрана (кроме самих координат окна) являются относительными координатами данного окна, т.е. координаты рассматриваются относительно левого верхнего угла окна. Многие процедуры и функции модуля Crt, в частности, read, readln, write, writeln действуют в активном окне так, как если бы это окно представляло весь экран.

Привлечение внимания к отдельным элементам текста может быть сделано выделением цветом или мерцанием символов.

При использовании в программе функции ReadKey следует учитывать, что специальные клавиши (функциональные клавиши, клавиши управления курсором, клавиши, модифицирующие значения других клавиш,: *Alt* и т.д.) генерируют расширенные коды. При нажатии такой клавиши функция ReadKey возвращает сначала нулевой символ (#0), а затем расширенный код клавиши.

Пример 1**.** Разработать программу, которая выводит на экран текстовое окно. При нажатии клавиши ← оно движется влево, при нажатии → – вправо, при нажатии ↓ – вниз, при нажатии ↑ – вверх. Окончание работы программы по клавише <Enter>.

```
program Prog11_1;
  uses crt; {подключение модуля}
  var k:byte; x,y:integer;
procedure setw(x,y:integer); {процедура вывода окна на
экран в заданном положении(x,y,x+19,y+19)}
begin
  TextBackground(black); {черный фон}
  clrscr;
  window(x,y,x+19,y+9);
  TextBackground(blue); {синий фон окна}
  clrscr;
end;
```

```
begin
  clrscr;
  readln(x,y); { ввод начальных данных }
  setw(x,y);
{бесконечный (пока не будет нажата <Enter>) цикл для движения окна по экрану}
  repeat
    k:=readkey;    { опрос буфера клавиатуры }
    if k=#0 then begin { если младший байт равен 0 }
      k:=readkey; { считываем следующий байт}
      if (k=#75) and (x>1) then dec(x); {нажатие клавиши ←}
      if (k=#72) and (y>1) then dec(y); {нажатие клавиши ↑}
      if (k=#77) and (x<61) then inc(x); {нажатие клавиши →}
      if (k=#80) and (y<16) then inc(y); {нажатие клавиши ↓}
    end;
    setw(x,y);
  until k=#13; {завершение цикла по нажатию <Enter>}
end.
```

Второе условие в операторах **if** служит для ограничения движения окна границами текстового экрана.

Пример 2**.** Печать строк в звуковом сопровождении.

```
program Prog11_2;
  uses crt; {подключение модуля}
  var k:byte; x,y:integer;
procedure st(x,y:byte; s:string;ms:word);
  const hz=50;  {частота тона}
  var i:byte;
begin
  x:=x-1;
  for i:=1 to length(s) do begin
    sound(hz); delay(ms); {первый сигнал}
    gotoXY(x+i,y); write(s[i]); {печать символов}
    sound(2*hz); delay(ms); {второй сигнал}
    nosound
  end;
end;
begin
  clrscr;
```

```
  st(20,10,'0123456789abcdeedcba9876543210',40)
  readln;        {пауза до нажатия <Enter>}
end.
```

11.2. Графический режим

Положение точки на экране дисплея в графическом режиме задают графические координаты. В качестве графических координат используются порядковые номера пикселов. Допустимый диапазон изменения графических координат составляет
[0..mx -1] для x-координаты и [0..my -1] для y-координаты, где mx и my определяют разрешение монитора. Точкой отсчета является верхний левый угол экрана. Значения x-координаты отсчитываются слева направо, а y-координаты – сверху вниз.

В состав Турбо Паскаля входит модуль Graph, который содержит более 80 процедур, функций, а также встроенные типы и константы, предназначенные для работы в графическом режиме.

Все процедуры и функции модуля Graph можно разделить на функциональные группы:

- управление графическими режимами и их анализ;
- рисование графических примитивов и фигур;
- управление цветами и шаблонами заполнения;
- битовые операции;
- управление страницами;
- графические окна;
- управление выводом текста.

11.2.1. Использование модуля Graph

Переключение в графический режим работы дисплея выполняется вызовом процедуры InitGraph:

```
InitGraph(gd, gm, 'DriverPath');
```

Первый параметр в этой процедуре задает тип видеоадаптера[3], второй определяет видеорежим, а третий указывает путь к драйверу на диске. Пустая строка означает, что графический драйвер находится в том же каталоге, что и программа. Если параметру gd присвоить значение detect (0), то система включается в режим автоопределения. Чтобы задать определенный графический режим, следует присвоить значение переменной gm.

[3] См. константы, соответствующие виду графического драйвера.

Завершение работы в графическом режиме производится с помощью процедуры `CloseGraph`, которая выгружает драйвер из памяти и восстанавливает предыдущий видеорежим.

Для правильного отображения рисунков на экране необходимо учесть различия между декартовой и графической системами координат. Геометрические декартовы координаты точки *(x, y)* для ее отображения на экране можно пересчитать в графические (x_g, y_g) по формулам

$$x_g = [s_x \times x] + dx,$$

$$y_x = my - [s_y \times y] - dy,$$

где [ ] – целая часть, s_x и s_y – масштабные множители, выбираемые из условий

$$mx = [s_x \times x_{max}]+1,$$

$$my = [s_y \times y_{max}]+1.$$

Здесь x_{max} и y_{max} – максимальные значения геометрических координат, dx и dy – смещение изображения относительно начала графических координат (левого верхнего угла экрана). Изображение будет смещено в центр экрана при

$$dx = [mx/2], \quad dy = [my/2].$$

11.2.2. Программирование движущихся объектов

Рисование на экране является действием с битами, т.е. побитовыми операциями над содержимым видеопамяти и битами изображения. В Паскале режимы вывода изображений на экран, задаются процедурой `SetWriteMode(Wmode:integer)`.

Параметр Wmode может в различных процедурах принимать пять значений: `copyput` (0), `xorput` (1), `orput` (2), `andput`(3), `notput` (4). Последним четырем значениям соответствуют логические операции над битами `xor, or, and, not`. При инициализации графики устанавливается режим замещения битов памяти монитора битами изображения – `copyput`. Поэтому, если на экране есть изображение, оно заменяется новым.

Если требуется прорисовать движущийся объект, то часто его рисуют сначала цветом объекта, потом цветом фона. Это допустимо, если движение объекта происходит по пустому экрану. Если фоном является другое изображение, то этот прием непригоден. В таком случае используют режим поразрядного совмещения изображений `xorput`, который при повторной прорисовке объекта восстанавливает исходное состояние видеопамяти.

Пример 3. Случайное перемещение прямоугольника по экрану, используя в качестве фона параллелипипед.

```
program Prog11_3;
  uses graph,crt;
  var gd,gm:integer;
  dx,dy,x,y:integer;
  mx,my:integer; {разрешение дисплея}
begin
  gd:=0;
  initgraph(gd,gm,' ');
  mx:=getmaxx;              {максимально адресуемые}
  my:=getmaxy;              {координаты экрана}
  dx:=mx div 4;             {вычисление сторон}
  dy:=my div 4;             {прямоугольника}
  bar3d(dx,dy,mx-dx,my-dy,30,true); { параллелипипед,
          левый верхний угол которого задается координатами
          dx,dy, правый нижний – mx-dx,my-dy, с глубиной –
          30 и отображением верхней плоскости – true }
  setwritemode(xorput); {установка режима совмещения}
  repeat
    x:=random(mx-dx); y:=random(my-dy); {задание
                  координат левого угла прямоугольника}
    rectangle(x,y,x+dx,y+dy); {рисует прямоугольник}
    delay(10000);           {пауза в 10000 мс}
    rectangle(x,y,x+dx,y+dy) {рисует прямоугольник
                                  на том же месте}
  until keypressed; {цикл до нажатия клавиши}
  closegraph
end.
```

Если перемещаемый объект имеет сложный образ, то целесообразно выполнить его рисование один раз, а затем использовать буфер для сохранения двоичного образа области экрана, занимаемого этим объектом и восстановления его на траектории движения в режиме совмещения. Для реализации такого алгоритма используются следующие подпрограммы:

`ImageSize(x1,y1,x2,y2:integer):word` – функция возвращает число байтов, занимаемых прямоугольной областью экрана;

`GetImage(x1,y1,x2,y2:integer; var b);` – процедура сохраняет в буфере, адрес которого указывает параметр

p – ссылка на битовый массив – двоичный образ заданной прямоугольной области экрана с координатами x1,y1,x2,y2;

PutImage(x,y:integer; **var** b; mode:word); – процедура выводит содержимое буфера в прямоугольную область экрана, с координатами левого верхнего угла x,y с использованием режима mode.

Последовательность действий для работы с буфером:

```
var size:word; p:pointer;
……
Size:=ImageSize(x1,y1,x2,y2);          {прямоугольный фрагмент в байтах}
GetMem(p,size); {размещение буфера в куче}
GetImage(x1,y1,x2,y2;p^);{занесение фрагмента в буфер}
……
PutImage(x,y; p^;xxxPut); {вывод фрагмента, вместо
        xxxPut используется один из режимов: copyput,
        xorput, orput, andput, notput}
FreeMem(p,size); {освобождение буфера}
```

Пример 4. Написать программу случайного перемещения шарика по фону, заданному в виде прямоугольника.

```
program Prog11_4;
  uses graph, crt;
  const r=10;
    var x1,y1,x2,y2,sx,sy:integer; {переменные для }
    maxx,maxy,xm,ym,gd,gm:integer; {оживления
    фрагмента}
    size : word; {размер фрагмента}
    p : pointer; {указатель на буфер}
begin
  gd:=detect;
  initgraph(gd,gm,''); { инициализация графики}
  maxx := GetMaxX; { максимальное поле экрана}
  maxy := GetMaxY;
  x1:=maxx div 2-r; {координаты области экрана}
  y1:=maxy div 2-r; {в которой будет нарисован}
  x2:=x1+2*r; {шарик и которая и будет}
  y2:=y1+2*r; {со храненным фрагментом}
  sx:=x1; xm:=3; {начальная точка движения и }
  sy:=y1; ym:=-1; {шаг перемещения шарика}
  SetFillStyle(SolidFill, Red); { выбор типа заливки}
  PieSlice(X1+r,Y1+r, 0,360,r); { рисование шарика}
  size:=ImageSize(x1,y1,x2,y2); {фрагмент в байтах}
  GetMem(p, Size);        { размещение буфера }
```

```
  Getimage(x1,y1,x2,y2,p^); { фрагмент в буфер}
  SetFillStyle(CloseDotFill,blue); {тип заливки фона}
  bar(50,50,maxx-50,maxy-50); {фоновая картинка}
  repeat            {начинается движение шарика}
    PutImage(sx,sy, p^,xorPut); {вывод шарика}
    Delay(3600); { пауза }
    PutImage(sx,sy,p^,xorPut); {стирание шарика}
    {ограничения на движение шарика в пределах поля фона:}
    if (sx<50)or(sx>maxx-50-2*r) then xm:=-xm;
    if (sy<50)or(sy>maxy-50-2*r) then ym:=-ym;
    inc(sx, xm); {следующая точка появления}
    inc(sy, ym); {шарика на экране}
  until KeyPressed;   {пока не нажата клавиша}
  FreeMem(p,size); {освобождение памяти буфера}
  CloseGraph      {закрытие режима графики}
end.
```

Имитировать движение на экране можно с помощью управления видеостраницами. По умолчанию действия в графическом режиме производятся с 0-ой страницей. Если направить вывод на ненулевую страницу (если есть), то на экране ничего не отобразится. Переход на скрытую страницу делает изображение видимым. Для этого используем процедуры:

`SetVisualPage(n:word);` – устанавливает видимой на экране страницу с номером `n`;

`SetActivePage(n:word);` – перенаправляет все графические операции на страницу с номером `n`.

Пример 5**.** Программа выводит на экран пульсирующий эллипс.

```
program Prog11_5;
  uses graph,crt;
  var j:byte; gd,gm:integer;
procedure form(k:byte);   {рисование кадров 0..3}
  const r:array[0..3] of integer=(20,40,60,80);
  var r1,rr:integer; {радиусы эллипсов в кадрах}
begin
  r1:=r[k];        {максимальный радиус }
  rr:=0;           {радиус вложенного эллипса}
  repeat
    ellipse(getmaxx div 2,getmaxy div
    2,0,360,r1,rr); {рисование эллипса с центром в середине
    экрана}
    inc(rr,5);     {изменение радиуса вложенного эллипса}
  until rr>r1;     {пока не достигнет максимального радиуса}
end;
```

```
procedure smena;              {процедура смены кадров }
  const ms=460;
  var i:byte;
begin
  repeat
    for i:=0 to 3 do begin
      setvisualpage(i);  {смена видеостраниц}
      delay(ms)          {с задержкой между кадрами }
    end;
    for i:=3 downto 0 do begin
      setvisualpage(i); {смена видеостраниц в обратном
      порядке}
      delay(ms);
    end;
  until keypressed; {цикл до нажатия клавиши}
end;
begin
  gd:=0;
  initgraph(gd,gm,''); {инициализация графики}
  setgraphmode(VGALo); {установка графического режима
                        VGA с 4 видеостраницами}
  for j:=3 downto 0 do begin {цикл заполнения страниц}
    setvisualpage(succ(j) mod 4); {видимые страницы,
                                   начиная с 0}
    setactivepage(j);             {активные страницы,
                                   начиная с 3}
    form(j) {рисование кадра}
  end;
  smena;
  closegraph
end.
```

Иллюзию движения можно создать, используя смену палитр. Программная палитра может состоять из 16 цветов, которые обозначаются числами от 0 до 15. Каждому программному цвету присваивается аппаратный цвет из полной палитры (256 цветов). В модуле Graph определен тип для описания палитры:

```
PaletteType = record
  size:byte;
  colors: array[0..15] of shortint
end;
```

и процедуры установки соответствия программному цвету аппаратного, смены палитр.

Пример 6**.** Пример имитации движения с помощью смены палитр.

```
program Prog11_6;
  uses graph, crt;      {подключены graph и crt}
  var palette:PaletteType; {переменная  для палитры }
  i,j,maxc,gd,gm:integer; {счетчики; максимальный цвет}
begin
  gd:=detect;
  initgraph(gd,gm,''); { инициализация графики}
  palette.size:=GetPaletteSize;    {размер текущей
  палитры}
  maxc :=pred(palette.size); {программный цвет }
  for i:=0 to maxc do begin    {рисование вложенных}
                               {разноцветных прямоугольников}
    SetFillStyle(SolidFill,i);
    bar(i*10,i*10,GetMaxX-i*10,GetMaxY-i*10)
  end;
  for i:=0 to 63-maxc do begin {цикл по аппаратным
  цветам }
    {Сдвиг программных цветов относительно аппаратных:}
    for j:=0 to maxc do palette.colors[j]:=j+i;
    SetAllPalette(palette);{назначение новой палитры}
    delay(10000) {пауза в 10000 мс}
  end;
  readln;              {пауза до нажатия ввода}
  GetDefaultPalette(palette);    {берется исходная
  палитра}
  SetAllPalette(palette); {и восстанавливается}
  CloseGraph           {закрытие графики}
end.
```

Контрольные вопросы:

1. В чем заключается механизм отображения информации на экран дисплея в текстовом режиме?
2. Какие существуют возможности размещения информации на экране?
3. Какие есть способы выделения некоторой части информации на экране?
4. Как использовать символы псевдографики для представления информации в требуемой форме?
5. Как реализуется перемещение символов по экрану?
6. Как используются функции опроса клавиатуры для управления информацией на экране?
7. В чем заключается механизм отображения информации на экран дисплея в графическом режиме?
8. В чем отличие способов создания изображения в текстовом и графическом режимах?
9. Какие существуют возможности размещения информации на экране?

10. Как осуществить преобразование декартовых координат на плоскости в координаты экрана?
11. Как реализуется в программе эффект движущегося по экрану изображения?
12. Как осуществляется управление цветом изображения в различных графических режимах?

Лабораторная работа № 11

Задание 1. Ипользуя текстовый режим работы, выполнить задание, соответетствующее варианту.

Варианты заданий:

1. Дана строка символов. Разработать программу вывода на экран этой строки так, чтобы каждый символ появлялся на экране справа и перемещался в горизонтальном направлении влево при нажатии клавиши ← до первой незанятой позиции. После того как очередной символ займет свое место, должен начинаться вывод следующего символа и т.д.

2. Написать процедуру, которая выдает на экран (в текстовом режиме) таблицу из m строк и n столбцов шириной p позиций каждый. Привести пример использования разработанной процедуры.

3. Разработать программу конфиденциального ввода информации (цвет символов совпадает с цветом фона). При нажатии «парольной» клавиши введенная или вводимая информация становится видимой.

4. Написать программу рисования в текстовом режиме вертикальных и горизонтальных линий по нажатию соответствующих клавиш движения курсора. Для перемещения курсора в начальное положение использовать клавиши движения курсора.

5. Разработать программу, которая случайным образом устанавливает курсор в какой-либо строке экрана на диагонали и выводит номер строки. Спустя некоторое время это сообщение исчезает и появляется следующий случайный номер.

6. Разработать программу передвижения прямоугольника по экрану при нажатии клавиш перемещения курсора.

7. Разработать программу ввода нескольких строк текста. Каждая строка вводится в отдельном окне. Предусмотреть эффект сжимания предыдущей строки в точку при выводе строки в новом окне.

8. Разработать программу вывода на экран меню следующего вида:

Вариант 1	Вариант 2	Вариант 3

отображающего процесс выбора любого из перечисленных вариантов, осуществляемый с помощью ← и →. При этом вариант, на который падает выбор, должен выделяться повышенной яркостью или цветом. При нажатии клавиши <ВВОД> или ↑ на месте соответствующего варианта должно появляться другое окно. Используйте данную форму меню в конкретной задаче.

9. Разработать программу вывода на экран меню следующего вида:

Выбор ⟶ Вариант 1
Вариант 2
Вариант 3
Вариант 4

Процесс выбора одного из вариантов осуществлять с помощью клавиш ↓ и ↑. При этом к строке, на которую падает выбор, должна быть направлена стрелка от слова "Выбор". Используйте данную форму меню в конкретной задаче.

10. Разработать программу выдачи горизонтального меню с выпадающими вертикальными подменю:

Выбор 1	Выбор 2	Выбор 3
	Вариант 1	
	Вариант 2	
	Вариант 3	

Подменю активизируется клавишей <ВВОД>. Выбор режима осуществляется клавишами перемещения курсора. Используйте данную форму меню для вывода информации о знаках тригонометрических функций в соответствующих четвертях координатной плоскости.

11. Разработать программу вывода на экран меню следующего вида

В этом меню приведен полный список вариантов. Название

каждого варианта оформляется в отдельном окне. Полностью видно только одно окно, на которое падает выбор. При нажатии клавиш ↓ и ↑ порядок окон меняется. Разработанная программа должна отображать этот процесс выбора варианта для конкретной задачи.

12. Разработать программу выдачи информации в виде двухуровневого гипертекста. На 1-ом уровне в тексте цветом выделены некоторые слова, к которым можно при нажатии клавиши <ВВОД> получить пояснения в виде текста на 2-ом уровне. Возвращение к тексту предыдущего уровня выполнять нажатием клавиши <ESC>.

13. Разработать программу, выдающую на экран меню следующего вида:

Заголовок
F1 - Вариант 1
F2 - Вариант 2
F3 - Вариант 3
F4 - Вариант 4

Выбор варианта осуществляется соответствующими функциональными клавишами. Приведите пример использования данной формы меню в конкретной задаче.

14. Разработать заставку к программе, которая содержит основные сведения о программе (автор, название, год разработки). Предусмотреть эффект сжимания всего изображения в точку в центре экрана.

15. Разработать программу ввода и вывода матрицы по строкам. Ввод каждого числа осуществляется в заданную позицию экрана таким образом, чтобы на экране вводимые числа располагались по строкам и столбцам в виде матрицы. Размерность матрицы и максимально возможное количество позиций в числе вводятся с клавиатуры. При выводе матрицы предусмотреть выравнивание столбцов по правому краю (для вещественных чисел оставить 2 знака после запятой).

Задание 2. Ипользуя графический режим работы, вывести на экран график функции в декартовой системе координат. Нанести обозначения и разметку осей.

Варианты заданий:

1. $y = x + \sin x$
2. $y = |\sin x|$

3. $y=\cos 3x$
4. $y = e^x$
5. $y=\sin x+\sin x$
6. $y=4x^2+5x-7$
7. $y=x^3-6x+9$
8. $y=\sqrt{|x|}$
9. $y=2+3\cos x$
10. $y=\cos x+\sin x$
11. $y=2\sin x+1$
12. $y=5x^2-2x$
13. $y = \frac{1}{x}$
14. $y=\ln x+3$
15. $y = \frac{1}{x^2}$

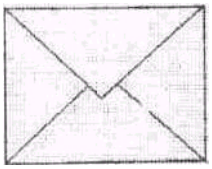
Задание 3. Ипользуя графический режим, нарисовать картинку, имитируя движение в соответствии с вариантом.

Варианты заданий:

1. Дом на берегу моря, на море – движущийся парусник. Когда он причалит к берегу, в окне дома загорается свет.
2. Рождественская елка в огнях и игрушках. Гирлянды огней зажигаются и гаснут.
3. Море с бегущей волной, корабль и маяк. Маяк подает сигналы.
4. Летящие над полем (лесом, морем) и машущие крыльями птицы.
5. Перекресток со светофором для пешеходов. Пешеход движется на зеленый свет.
6. Аквариум с плавающей рыбкой.
7. Перемещение по экрану стилизованного изображения человека (окружность – голова, треугольник – туловище, четыре ломаные линии – руки и ноги). Перемещение человека, изображенного на экране, должно осуществляться при нажатии соответствующих клавиш управления курсором.
8. Построение последовательно следующих рисунков:

При нажатии некоторой клавиши меняется форма рта.

9. Нарисовать почтовый конверт:



Предусмотреть, чтобы по нажатию какой-либо клавиши конверт открывался, в него влетал листок, затем конверт закрывается.

10. Циферблат механических часов с движущимися секундной, минутной и часовой стрелками (при этом используйте процедуру `Delay`).

11. Изменение масштаба изображения (рисунок и текст) в прямоугольном окне на экране при нажатии фиксированных клавиш.

12. Перемещение упругого шарика в замкнутом пространстве прямоугольной формы. Начальное направление и скорость движения шарика должны задаваться произвольно.

13. Разделить экран на 2 части: в одной предусмотреть выдачу позитивного изображения (белое на черном), а в другой – негативного (черного на белом). Написать процедуру движения по такому экрану некоторого геометрического объекта.

14. Выделение цветом, отличным от фона и цвета изображения прямоугольного фрагмента рисунка на экране. Задание размеров фрагмента с помощью стрелки-указателя, фиксируя концы одной из диагоналей прямоугольника. Для управления стрелкой-указателем по экрану использовать клавиши перемещения курсора и ввода.

15. Многоугольник любой формы. Вершины многоугольника должны задаваться путем перемещения текущего указателя в форме стрелки с помощью клавиш управления курсором и нажатия клавиш <ВВОД>. Замкнутое пространство внутри многоугольника следует заполнить штриховкой.

Л И Т Е Р А Т У Р А

1. Абрамов, В.Г. Введение в язык Паскаль. – М.: Наука, 1988. – 318 с.
2. Бондарев, В.М. Основы программирования. – Харьков: Фолио, 1997. – 368 с.
3. Гуденко, Д.А. Сборник задач по программированию. – СПб.: Питер, 2003. – 475 с.
4. Давыдов, В.Г. Программирование и основы алгоритмизации. – М.: Высш. школа, 2003. – 448 с.
5. Долинский, М.С. Алгоритмизация и программирование на Turbo Pascal: от простых до олимпиадных задач. – СПб.: Питер, 2005. – 237 с.
6. Епанешников, А. М. Программирование в среде Turbo Pascal 7.0. – М.: Диалог-МИФИ, 1996. – 288 с.
7. Зуев, Е.А. Программирование на языке TURBO PASCAL 6.0, 7.0. – М.: Радио и связь, 1993. – 384 с.
8. Лукин, С.Н. Турбо Паскаль 7.0: самоучитель для начинающих. – М.: Диалог-МИФИ, 2000. – 384 с.
9. Немнюгин, С.А. Turbo Pascal: практикум. – СПб.: Питер, 2000. – 256 с.
10. Офицеров, Д.В., Долгий, А.Б., Старых, В.А. Программирование на персональных ЭВМ: практикум. – Мн.: Выш. школа, 1993. – 256 с.
11. Павловская, Т.А. Паскаль: программирование на языке высокого уровня. – СПб.: Питер, 2004. – 393 с.
12. Пильщиков, В.Н. Сборник упражений по языку Паскаль. – М.: Наука, 1989. – 160 с.
13. Поляков, Д.Б., Круглов, И.Ю. Программирование в среде ТУРБО ПАСКАЛЬ. – М.: Изд. МАИ, 1992. – 576 с.
14. Сурков, Д.А. Программирование в среде Borland Pascal для Windows: справочное пособие. – Мн.: Выш. школа, 1996. – 432 с.
15. Фаронов, В.В. Турбо Паскаль 7.0: Начальный курс. – М.: Нолидж, 2000. – 576 с.
16. Фаронов, В.В. Турбо Паскаль 7.0: Практика программирования. – М.: Нолидж, 2000. – 416 с.

П Р И Л О Ж Е Н И Я

Приложение 1

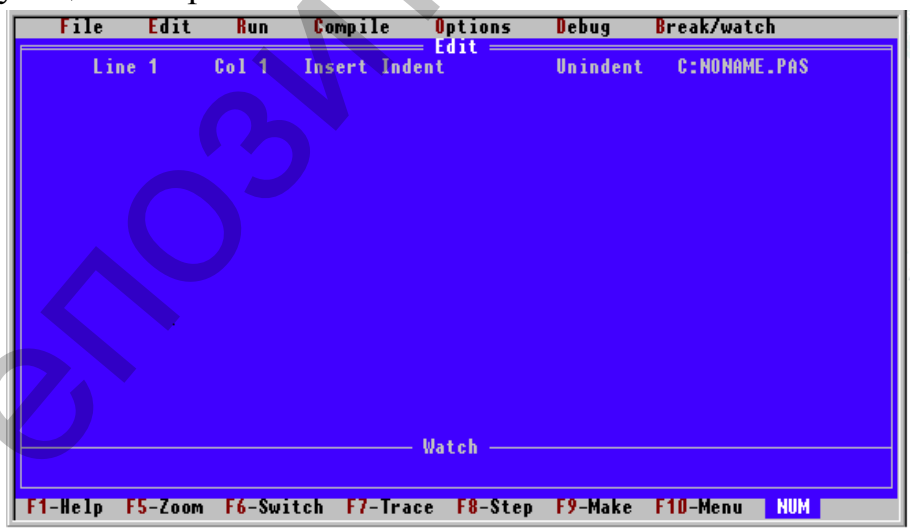
Краткое описание системы программирования Турбо Паскаль

Базовыми компонентами системы программирования Турбо Паскаль являются компилятор языка Паскаль, средства создания и редактирования исходных текстов программ и средства их отладки. Все эти компоненты объединены в единую интегрированную среду разработчика. Базовый набор для разработки программ, не использующих графический режим, состоит из файла TURBO.EXE, который хранит ядро системы, и файла TURBO.TPL, содержащего резидентную библиотеку модулей.

Чтобы инициировать работу Турбо Паскаля, надо запустить программу из файла TURBO.EXE на выполнение. Это можно сделать с помощью DOS-ориентированных оболочек управления файлами (Windows Commander, Norton Commander, Far и т.п.) либо команды системного меню ***Пуск | Выполнить***. В этом случае среда Turbo Pascal будет открыта в окне Windows.

Рассмотрим кратко основные элементы интегрированной среды программирования.

При запуске Турбо Паскаля, на экране появляется следующее изображение:



Экран Турбо Паскаля содержит 4 основные части, сверху вниз:

- главное меню (File, Edit, Run, Compile, Options, Debug, Break/Watch);
- окно редактирования (Edit) со служебной информацией редактора в верхней строке;
- окно просмотра (Watch) для отладки программ;

– строку подсказки о назначении функциональных клавиш (F1..F10).

Рассмотрим коротко назначение основных пунктов меню Турбо Паскаля.

File	
New	удаление текущей программы из оперативной памяти, очистка экрана и создание нового файла с именем NONAMEXX.pas
Open [F3]	загрузка файла с диска в новое окно редактора. При этом в диалоговом окне на выбор предлагаются имена файлов с маской *.pas. Если ввести имя, которого нет в текущем каталоге, то будет создан новый файл.
Save [F2]	сохранение на диске текущего редактируемого файла. Если имя файла было NONAMEXX.pas, то среда запросит новое имя.
Save as	запись содержимого окна на диск под другим именем. В поле ввода диалогового окна необходимо написать новое имя, которому автоматически будет дано расширение .pas.
Change dir	изменяет текущий каталог пользователя.
Print	печатает содержимое текущего окна на принтере.
Exit [Alt+X]	завершает работу с Турбо Паскалем. Синоним команды – сочетание клавиш.

Edit	
Undo [Alt + Back Space]	отмена предыдущего выполненного действия в редакторе, так называемый «откат».
Redo	отменяет действие команды Undo.
Cut [Shift + Del]	удаляет выделенный блок текста из окна редактора и переносит его в буфер обмена Clipboard .
Copy [Ctrl + Ins]	копирует выделенный блок из окна редактора в буфер обмена.
Paste [Shift + Ins]	копирование содержимого буфера обмена в окно редактора, в то место, где в данный момент находится курсор.
Clear [Ctrl + Del]	удаляет из окна редактора выделенный блок без помещения его в буфер обмена
Run	
Run [Ctrl + F9]	запускает компиляцию (трансляцию), компоновку (сборку, линковку) и выполнение программы
Step over [F8]	выполняет те же действия, что и [F7], но без

	трассировки процедур и функций
Trace Into [F7]	запускает режим пошагового (построчного) выполнения программы (трассировка). Многократно нажимая клавишу [F7], строка за строкой, мы можем выполнить всю программу. Очередная строка при этом подсвечивается голубым цветом.
User Screen [ALT+F5]	отображает состояние экрана дисплея после завершения прогона программы. Повторное нажатие этих клавиш возвращает экран компьютера в исходное состояние.

Compile	
Compile [Alt + F9]	компилирует программу, которая находится в окне редактора (или содержащуюся в файле, имя которого указано в подкоманде Primary file команды Compile).
Destination	с помощью этой опции можно указать системе, куда поместить «EXE» – файл на диск DISK или в оперативную память MEMORY. Переключение из одного состояния в другое осуществляется нажатием клавиши ENTER.

Options	
Compiler	разворачивается дополнительное меню, с помощью которого можно управлять параметрами компилятора.
Environment	настройка среды Турбо Паскаля для удобства работы конкретного пользователя.
Directories	настройка рабочих каталогов (директорий).

Break/Watch	
Add Watch	эта опция позволяет поместить или добавить переменную или выражение в окно просмотра и узнать их текущее значение.
Delete Watch	эта опция удаляет из окна WATCH текущее просматриваемое выражение (режим удаления доступен в том случае, если окно просмотра WATCH является видимым). Удалить текущее выражение просмотра можно также с помощью клавиш DEL или CTRL-Y.
Watch	открывает окно просмотра.

Приложение 2

Типы данных в Паскале

Тип величины определяет возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу; внутреннюю форму представления данных в ЭВМ; операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

- **Классификация типов:**
  - Простые
    - Порядковые
      - Целый
      - Логический
      - Символьный
      - Перечисляемый
      - Интервальный
    - Вещественный
    - Указатели
  - Структурированные
    - Массивы
    - Строки
    - Множества
    - Записи
    - Файлы
- **Простые типы данных**

В таблице приведены простые стандартные типы данных Турбо Паскаль, объем памяти, необходимый для хранения одной переменной указанного типа, множество допустимых значений и применимые операции.

Таблица 2.1

Стандартные типы данных

Идентификатор	Длина (байт)	Диапазон значений	Операции*
Целые типы			
integer	2	-32768..32767	+, -, /, *, div, mod, >=, <=, =, <>, <, >
byte	1	0..255	+, -, /, *, div, mod, >=, <=, =, <>, <, >
word	2	0..65535	+, -, /, *, div, mod, >=, <=, =, <>, <, >
shortint	1	-128..127	+, -, /, *, div, mod,

* Описание операций приведено в приложении 3.

Идентификатор	Длина (байт)	Диапазон значений	Операции*
			>=, <=, =, <>, <, >
longint	4	-2147483648..2147483647	+, -, /, *, div, mod, >=, <=, =, <>, <, >
Вещественные типы			
real	6	$2{,}9x10^{-39}..1{,}7x10^{38}$	+, -, /, *, >=, <=, =, <>, <, >
single	4	$1{,}5x10^{-45}..3{,}4x10^{38}$	+, -, /, *, >=, <=, =, <>, <, >
double	8	$5x10^{-324}..1{,}7x10^{308}$	+, -, /, *, >=, <=, =, <>, <, >
extended	10	$3{,}4x10^{-4932}..1{,}1x10^{4932}$	+, -, /, *, >=, <=, =, <>, <, >
Логический тип			
boolean	1	true, false	not, and, or, xor, >=, <=, =, <>, <, >
Символьный тип			
char	1	все символы кода ASCII	+, >=, <=, =, <>, <, >

Перечисляемый тип представляет собой ограниченную упорядоченную последовательность скалярных констант, составляющих данный тип. Значение каждой константы задается ее именем. Имена отдельных констант отделяются друг от друга запятыми, а вся совокупность констант, составляющих данный перечисляемый тип, заключается в круглые скобки:

type имя_типа = (константа_1, константа_2, …, константа_N);

Пример.

```
type
  colors =(red, white, blue);
```

Отрезок любого порядкового типа может быть определен как ***интервальный*** тип. Отрезок задается диапазоном от минимального до максимального значения констант, разделенных двумя точками:

```
type
имя_типа = <мин.зн. конст.>..<макс.зн. конст.>
```

В качестве констант могут быть использованы константы, принадлежащие к целому, символьному, логическому или перечисляемому типам.

Пример.

```
type
  digit = '0'..'9';
  hour= 0..24;
```

Интервальный тип необязательно описывать в разделе TYPE, а можно указывать непосредственно при объявлении переменной.

Пример.

```
var
  date : 1..31;
  month: 1. .12;
```

- **Структурированные типы данных**

Описание структурированных типов данных (массивов, множеств, строк, записей, файлов) приводится в соответствующих разделах данного издания.

Приложение 3

Операции и отношения

Таблица

Операции (арифметические, отношения, логические)

Арифметические операции			
Операция	**Действие**	**Тип операндов**	**Тип результата**
+	Сложение	I, R*	I, R
-	Вычитание	I, R	I, R
*****	Умножение	I, R	I, R
/	Деление	I, R	I, R
div	Целочисленное деление	I	I
mod	Остаток от деления	I	I

Логические операции			
Операция	**Действие**	**Тип операндов**	**Тип результата**
and	Логическое И	B**	I, R
or	Логическое ИЛИ	I, R	I, R
xor	Логическое исключающее ИЛИ	I, R	I, R
not	Отрицание	I, R	I, R

Операции отношения		
Операция	Название	Выражение
=	Равно	A=B
<>	Неравно	A<>B
>	Больше	A>B

* I – любой целочисленный тип, R – любой вещественный тип

* * B – тип boolean. Логические операции могут применятся также к целочисленным типам.

<	Меньше	A<B
>=	Больше или равно	A>=B
<=	Меньше или равно	A<=B

- **Приоритет операций**

Порядок вычисления выражения определяется старшинством (приоритетом) содержащихся в нем операций. В языке Паскаль принят следующий приоритет операций:

- унарная операция not, унарный минус -, взятие адреса @
- операции типа умножения: * / div mod and
- операции типа сложения: + - or xor
- операции отношения: = <> < > <= >= in

Порядок выполнения операций переопределить можно с помощью скобок.

Приложение 4

Процедуры и функции

Таблица

Основные стандартные процедуры и функции

Обращение	Тип параметра	Тип результата	Пояснение
abs (x)	I, R	I, R	Модуль x
sqr (x)	I, R	I, R	Квадрат аргумента
sqrt (x)	I, R	R	Корень квадратный
int (x)	I, R	R	Целая часть числа
frac (x)	R	R	Дробная часть числа
round (x)	R	I	Округляет x до ближайшего целого
trunc (x)	R	I	Целая часть аргумента
pi		R	π = 3.141592653...
sin (x)	I, R	R	Синус, угол в радианах
cos (x)	I, R	R	Косинус, угол в радианах
arctan (x)	R	R	Арктангенс (значение в радианах)
exp (x)	R	R	Экспонента
ln (x)	I, R	R	Логарифм натуральный
dec (x[, i])	I	I	Уменьшает значение x на i, а при отсутствии i - на 1
inc (x[, i])	I	I	Увеличивает значение x на i, а при отсутствии i - на 1
odd (x)	I	boolean	Возвращает true, если аргумент – нечетное число
random (x)	I, R	I, R	Псевдослучайное число, равномерно распределенное в диапазоне [0,x]
randomize			Инициация генератора псевдослучайных чисел
chr (x)	I	char	Символ по его коду
ord (x)	Порядковый	I	Порядковый номер значения выражения x
pred (x)	– “ –	тип параметра	Предыдущее значение порядкового типа
succ (x)	– “ –	– “ –	Возвращает следующее значение порядкового типа
low (x)	– “ –	– “ –	Наименьшее значение величин данного типа
high (x)	– “ –	– “ –	Наибольшее значение величин данного типа