ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

Методические указания
для студентов специальности 220301
"Автоматизация химико-технологических процессов и производств"

Иваново
2008

Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования
Ивановский государственный химико-технологический университет

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

Методические указания
для студентов специальности 220301
"Автоматизация химико-технологических процессов и производств"

Составители: Е.Я. Князева
М.С. Комлева

Иваново 2008

Составители Е.Я. Князева, М.С. Комлева

УДК 613.19

Программирование и основы алгоритмизации: метод. указания для студентов специальности 220301 «Автоматизация химико-технологических процессов и производств» / Сост. Е.Я. Князева, М.С. Комлева; Иван. гос. хим.-технол. ун-т. -Иваново, 2008.- 56 с.

В методических указаниях представлены: работа с файловой структурой данных, основные типовые алгоритмы по сортировке и поиску данных, методы и средства объектно-ориентированного программирования, рассмотрены понятия указателей и динамической памяти, а также задания для самостоятельной работы.

Предназначены для студентов очной формы обучения специальности 220301 «Автоматизация химико-технологических процессов и производств».

Ил. 23. Библиогр.: 6 назв.

Рецензент
кандидат технических наук Г.В. Волкова (Ивановский государственный химико-технологический университет)

ВВЕДЕНИЕ

Каким бы видом деятельности не занимался человек, без персонального компьютера в настоящее время не обойтись. Умение общаться с ЭВМ должно стать таким же естественным, как знание грамматики и арифметики. Бурное развитие науки, лавинообразный рост объема информации предъявляют новые требования к специалистам.

В своей повседневной деятельности часто приходится сталкиваться с разнообразными правилами, предписывающими последовательность действий, цель которых состоит в достижении некоторого результата. Подобные правила называют алгоритмами. Название «алгоритм» связано с именем выдающегося математика древности Муххамеда-бен-муса Аль-Хорезми (IX в.н.э.). Он изложил общие правила выполнения действий над числами, представленными в десятичной форме, которыми мы пользуемся до сих пор. Эти правила могли быть применены к любым числам.

В данных методических указаниях представлена работа с процедурами и функциями, с файловой структурой данных, основные типовые алгоритмы по сортировке и поиску данных, методы и средства объектно-ориентированного программирования, рассмотрены понятия указателей и динамической памяти.

Методические указания предназначены для студентов дневного обучения специальности 220301 «Автоматизация химико-технологических процессов и производств».

1. Подпрограммы в Турбо Паскале
1.1. Процедурный тип Паскаля

Процедурный тип является дополнением стандартного Паскаля и позволяет включать в состав формальных параметров процедур и функций (первичных) другие процедуры и функции (вторичные). Таким образом, оказывается возможным при обращении к первичным процедурам и функциям указывать в качестве фактических параметров самые разнообразные вторичные процедуры и функции.

Существует два процедурных типа: тип-процедура и тип-функция. Любой процедурный тип является нестандартным. Поэтому, в соответствии с правилами языка Паскаль, он должен быть соответствующим образом описан. И только после этого им можно пользоваться. Для размещения описаний нестандартных типов в языке Паскаль используется специальный раздел типов, начинающийся с зарезервированного слова *Type*. Для описания процедурного типа-функции используется заголовок функции, в котором опускается ее имя, например:

```
Type
  TFunc = function(x: Real): Real;
```

Смысл такого описания состоит в том, что теперь в обращениях к первичным процедурам и функциям в качестве фактических параметров могут использоваться все те вторичные функции, заголовок которых соответствует данному описанию процедурного типа-функции. Здесь под соответствием следует понимать, количественный состав параметров, их тип и тип самой функции. Механизм использования процедурных типов продемонстрируем на примере следующей программы.

```
program  procedureType;
uses Crt;
{описание процедурного типа}
type
  TFunc = function(x: Real): Real;
{описание подпрограмм}
function f1(x: Real): Real; far;
begin
  f1 := x*x
end;
function f2(x: Real): Real; far;
begin
  f2 := -x*x
end;
procedure PrintFunc(x: Real; Func: TFunc);
```

```
begin
  WriteLn(x:0:3, ' ', Func(x):0:3)
end;
var
  x: Real;
begin
  ClrScr; {очищаем экран}
  Write('Введите аргумент: x = ' );
  ReadLn(x);
  PrintFunc(x, f1);
  PrintFunc(x, f2);
  repeat until KeyPressed;
end.
```

Используемая в программе процедура ***PrintFunc*** предназначена для печати значения аргумента и соответствующего значения некоторой функции. Аргумент *x: Real*, и функция *Func: TFunc* указаны в качестве формальных параметров этой процедуры. При этом функция *Func*, значение которой должно быть напечатано, имеет тип-функцию *TFunc*. Этот процедурный тип, в соответствии с его описанием в разделе нестандартных типов *Type*, указывает, что он распространяется на все функции типа *Real*, и имеющие один аргумент также типа *Real*. Кроме этого, в программе представлены описания двух функций *f1(x: Real): Real и f2(x: Real): Real*, тип которых полностью соответствует типу *TFunc*. Поэтому они могут использовать в качестве фактических параметров обращения к процедуре *PrintFunc*.В исполняемой части программы сначала с клавиатуры вводится значение аргумента а затем дважды выполняется обращение к процедуре *PrintFunc*. При этом сначала печатается значение функции *f1*, а затем - значение функции *f2*. Для установления правильных связей функций *f1* и *f2* с процедурой *PrintFunc* они должны компилироваться с расчетом на, так называемую, дальнюю модель памяти. Вкратце поясним, что это такое.

Архитектурой персонального компьютера предусмотрено разбиение его оперативной памяти на сегменты, размером 64 Кбайт каждый. При этом могут использоваться как короткие адреса (в пределах сегмента), так и полные адреса (с указанием номера сегмента). Соответственно могут использоваться и две модели обращения к процедурам и функциям: ближняя (*nfar*) и дальняя (*far*). Ближняя модель обеспечивает адресацию в пределах текущего сегмента памяти, дальняя модель памяти используется для организации межсегментных связей. Установка дальней модели памяти для тех процедур и функций, которым она требуется, осуществляется с помощью стандартной директивы *far*. Эти директивы должны указываться непосредственно после заголовков процедур и функций. Дальняя модель памяти может быть установлена также для всей про-

граммы в целом. Для этого перед началом программы должна указываться директива компиляции *{$F+}*. В этом случае указание директив *Far* не требуется. Рассмотренный механизм не распространяется на стандартные процедуры и функции Паскаля.

Построим программу, реализующую алгоритм *MinF* («Наименьшее значение функции»). Напомним, что условие задачи требует найти значение аргумента, при котором достигается наименьшее из значений заданной функции *f(x)*, вычисленных в точках разбиения отрезка [a, b] на N равных частей. Алгоритм *MinF* на язык Паскаль выглядит следующим образом:

```
function MinF (a, b: Real; N: Integer): Real;
var
  h, min, x, y: Real;
  i: Integer;
begin
  h :=(b-a)/N;
  x := a;
   min := f(a);
  for i := 1 To N do
  begin
    a := a+h;
    y:=f(a);
    if  y < min then
    begin
      x:=a;
      min:=y
    end;
  end;
  MinF := x;
end;
```

Как видим, функция *MinF* предполагает наличие подпрограммы *f(x: Real): Real*, предназначенной для вычисления значений функции, точка наименьшего значения которой отыскивается. А это значит, что имеется возможность работать всего только с одной конкретной функцией *f(x)*. И если понадобится найти точку наименьшего значения какой-то другой конкретной функции, то это можно осуществить одним из двух основных способов:

- переписать подпрограмму *f(x: Real): Real* так, чтобы она вычисляла другую функцию;
- переписать функцию *MinF* так, чтобы она обращалась к другой функции.

Наверняка есть и какие-то другие сходные способы решения этой проблемы, однако, нет сомнения, что все они не универсальны и так или иначе связаны с переписыванием программы. Очевидно, тут требуется принципиально новое решение. Попробуем применить наши знания о процедурных типах.

```
{$B+, D+, E+, I+, L+, N+, Q+, R+, X-}
program Procedure_Type;
type
  TFunc = Function (x: Real): Real;
function fl (x: Real): Real; far;
begin
  f1 := (x-2)*(x-2)-2
end;
function f2 (x: Real): Real; far;
begin
  f2 := -fl(x);
end;
function MinF (a,b: Real; N: Integer; f: TFunc): Real;
var
  h, min, x, y: Real;
  i: Integer;
begin
  h := {b-a)/N; x:=a; min:=f(a);
  for i:=1 To N Do
  begin a:=a+h; y:=f(a);
    if  y<min then
    begin
      x := a;
      min := y;
    end;
  end;
  MinF := x;
end;
var a, b: Real; N: Integer;
begin
  Write('Введите координаты концов отрезка: a, b = ');
  ReadLn (a, b);
  repeat
    Write ('Введите количество участков разбиения: N = ');
    ReadLn(N);
  until N>0;
  WriteLn('Наименьшее значение достигается: ' ) ;
```

```
  WriteLn('  - для функции fl: при x = ', MinF (a, b, N, fl):0:3);
  WriteLn(' -для  функции f2:при x=  ',  MinF(a,  b,  N,  ?2):0:3);
  ReadLn
end.
```

Представленная в программе функция *MinF* имеет в качестве формального параметра некоторую функцию *f: TFunc*. Эта функция, для которой *MinF* отыскивает точку наименьшего значения, имеет процедурный тип-функцию *TFunc*. Этот процедурный тип, в соответствии с его описанием в разделе нестандартных типов *Type*, указывает, что он распространяется на все функции типа *Real*, имеющие единственный аргумент также типа *Real*. Кроме этого, в программе представлены описания двух функций *fI (x: Real): Real и f2(x: Real): Real*, тип которых полностью соответствует типу *TFunc*. Поэтому они могут использовать в качестве фактических параметров обращения к *MinF*. Для каждой из этих функций с помощью директивы *Far* установлена дальняя модель памяти. В качестве значений функции *f1* рассматриваются значения выражения *(x-2)*(x-2)-2*. Таким образом, функция *f1* представляет собой параболу, симметричную относительно вертикали *x=2*, и с ветвями, направленными вверх. Нетрудно убедиться, что наименьшее значение этой функции на отрезке [-5, 5] достигается при *x=2* и равно *-2*. На этом же отрезке наибольшее значение функции достигается при *x=5* и равно *47*. В качестве функции *f2* рассматривается функция, противоположная по знаку функции *f1*. Как указывалось при рассмотрении алгоритма *MinF*, это позволит с его помощью найти значение аргумента, при котором достигается наибольшее значение той же функции *f1*.В исполняемой части программы ввод с клавиатуры исходных данных организован обычным образом с учетом того, что для количества участков разбиения отрезка [a, b] допустимы значения N>0. Обращение к функции *MinF* выполняется дважды. При этом сначала печатается значение аргумента, при котором достигается наименьшее значение функции *fl*, а затем – значение аргумента, при котором достигается наименьшее значение функции *f2*, или, что то же самое, наибольшее значение функции *f1*. Теперь, если у нас возникнет желание найти точки минимального значения для каких-либо других функций, то программу переписывать не придется. Ее нужно будет только дополнить, а именно: ввести описания этих функций, а также обращения к ним.

1.2. Процедуры и функции

1.2.1. Процедуры

Для использования подпрограммы-процедуры необходимо сначала описать процедуру, а затем обращаться к ней (обращение к процедуре – отдельный оператор). Описание процедуры включает заголовок (имя) и тело процедуры.

Заголовок состоит из зарезервированного слова *procedure*, имени процедуры и, заключенного в скобки, списка формальных параметров с указанием типа. Название «формальные» эти параметры получили в связи с тем, что в этом списке заданы только имена для обозначения исходных данных и результатов работы процедуры, а при вызове подпрограммы на их место будут поставлены конкретные значения. Тело процедуры – блок, по структуре аналогичный программе.

При создании программ, использующих процедуры, следует учитывать, что все объекты, которые описываются после заголовка в теле процедуры, называются локальными объектами и доступны только в пределах этой процедуры. Все объекты, описанные в вызывающей программе, называются глобальными и являются доступными внутри процедур, вызываемых этой программой.

Общий вид описания процедуры:

```
procedure имя (список формальных параметров); {заголовок процедуры}
{ блок описания }
const
  ...
var
  ...
begin {операторы}
  ...
end;
```

ПРИМЕР: Вывести по четырем углам экрана свое имя цветными буквами, можно с эффектом мерцания.

```
program names;
uses crt;
procedure name(x, y, c: byte, );
begin
  Gotoxy (x, y);
  Textcolor(c);  {textcolor (c+16);}
  Write('Имя');
end;
begin
  ClrScr; {очистка экрана}
  name(2,2, 14);
  name(2, 22, 8);
  name(75,2, 3);
  name(75, 22, 5);
end.
```

1.2.2. Функции

Подпрограмма-функция обрабатывает данные, переданные ей из главной программы, и затем возвращает полученный результат (в отличие от процедуры). Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово *Function*, имя, список формальных параметров (заключенный в скобки) и тип возвращаемого функцией значения. Тело функции представляет собой локальный блок, по структуре сходный с программой.

Общий вид описания функции:

```
Function имя (параметры): тип результата; {заголовок функции}
{ блок описания }
const
  ...
var
  ...
begin {операторы}
  ...
end;
```

В разделе операторов должен находиться, хотя бы один оператор, присваивающий имени функции значение. Обращение к функции осуществляется по имени с указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам и иметь тот же тип.

ПРИМЕР: Найти наибольшее из четырех чисел, используя подпрограмму нахождения наибольшего из двух чисел.

```
program max;
uses crt;
var a, b, c, d, m, p, q: real;
function Bid(x, y: real): real;
begin
  if x<y then bid :=x
    else bid := y;
end;
begin
  ClrScr;
  Write('Введите числа');
  Readln (a, b, c, d);
```

```
  p := bid(a, b);
  q := bid(c, d);
  m := bid(p, q);
  Write('наибольший элемент', m:8:3);
end.
```

2. РАБОТА С ФАЙЛАМИ

Файл данных – последовательность (*seguence*) элементов одинакового типа. Файлы хранятся во внешней памяти (жесткие диски, CD, дискеты), файл отличается от массива следующим:

– число элементов в файле заранее неизвестно;

– одновременно доступен лишь один элемент.

На блок-схемах файловые операции изображаются в виде “бочонка” (рис. 1).

Рис. 1. Обозначение операций с файлом на блок-схеме.

Возможны **<u>два способа доступа</u>** к файлу: последовательный и параллельный (рис. 2). Разница между двумя способами доступа такая же, как между магнитофонной кассетой и CD: на кассете (последовательный доступ), чтобы добраться до пятой песни, надо промотать первые четыре, а на CD (прямой доступ) можно “перескочить” сразу на любой нужный трек.

Способ доступа не зависит напрямую от конструкции запоминающего устройства. Разумеется, если информация хранится на кассете с магнитной лентой (такое устройство называется *стримером*), то доступ всегда будет последовательным. А вот на жестком диске возможны и последовательный и параллельный виды доступа.

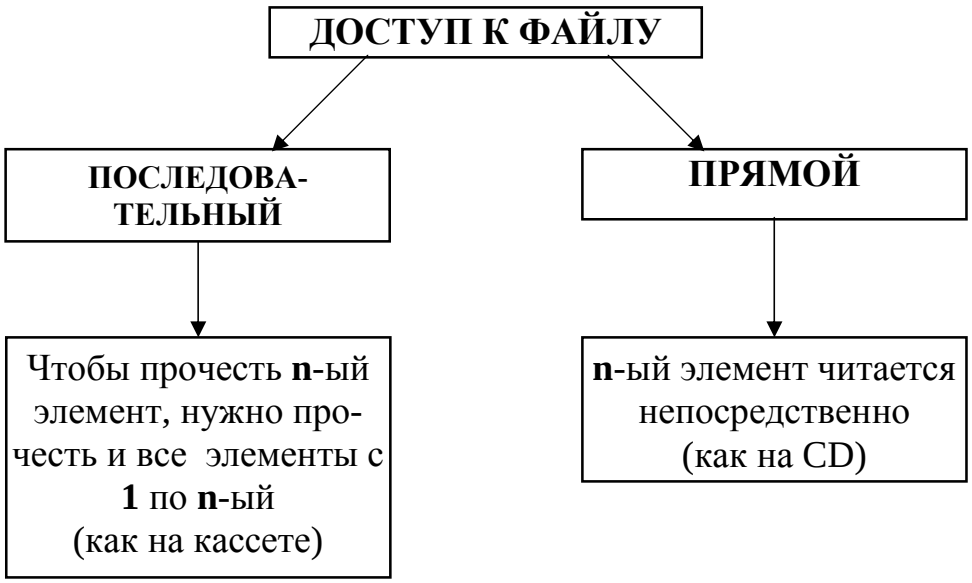


Рис. 2. Виды доступа к файлу.

По содержанию файлы делятся на текстовые и двоичные (рис. 3).

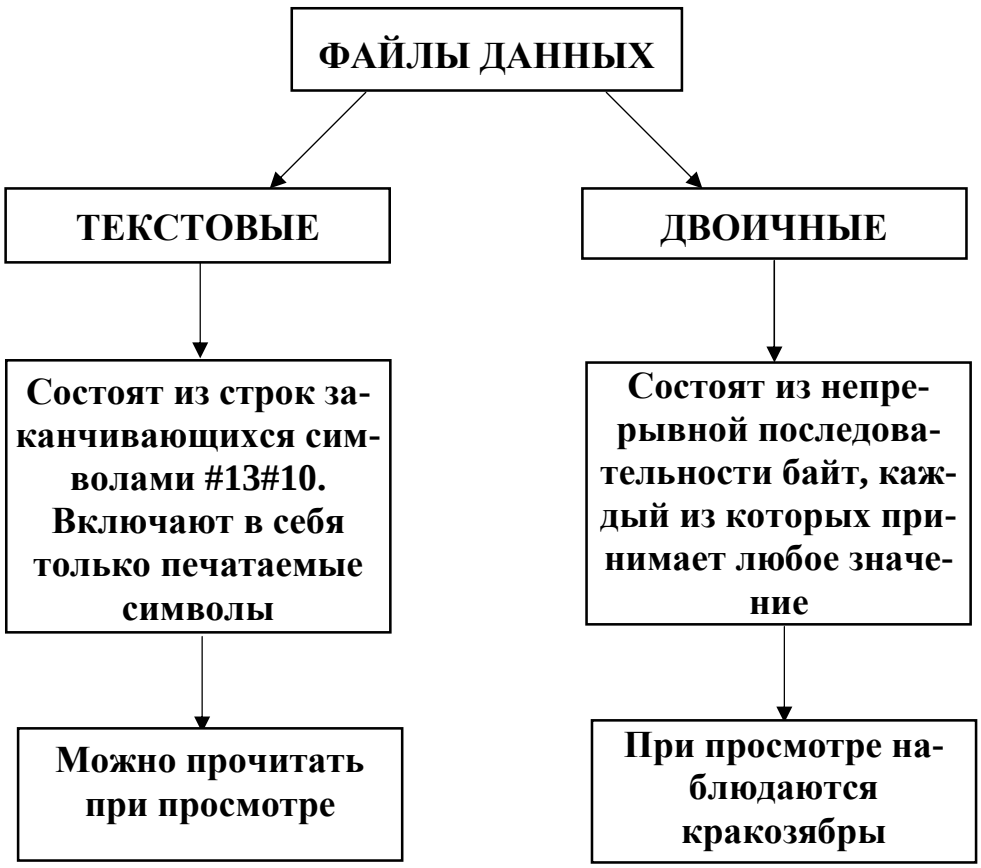


Рис. 3. Текстовые и двоичные файлы.

2.1. БУФЕРИЗАЦИЯ

Очень важный момент при любой работе с файлами – их **буферизация**. Если программа будет считывать информацию байт за байтом, то после каждого чтения будет заново выполняться установка магнитных головок в новое положение. В итоге работа с файлом замедлится. Чтобы такого не происходило, файл надо читать и записывать как можно более большими кусками (блоками). Лучше всего, когда размер такого блока кратен размеру блока на носителе информации. На жестких дисках и дискетах размер блока, как правило, кратен 256 байтам. Тогда программа за одну операцию чтения или записи может прочитать или считать 25600 байт или 512000 байт и так далее. Но для работы алгоритма нужен только один текущий элемент файла! Если размер элемента – 1 байт, а считывается 256 байт, то оставшиеся необходимо положить в **буфер**.

Буфер – специальная область памяти для временного хранения информации, которой программа обменивается с внешним носителем (рис. 4).

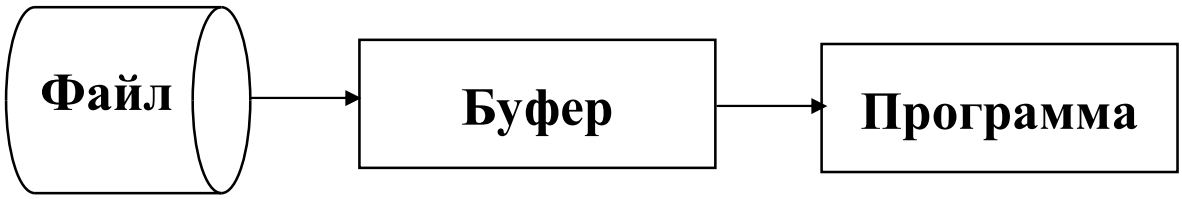


Рис. 4 Буферизация ввода/вывода.

Буферизацию надо использовать всегда, при любых файловых операциях. В ряде случаев буферизацию незаметно для программиста выполняет Паскаль, а иногда ее надо делать самостоятельно.

Буфер работает по принципу «первый пришел – первый ушел» (*FILO*– First *In, Last Out*). Чаще всего применяется **кольцевой буфер** – массив, в котором выделены индексы текущего записываемого (*in*) и считываемого (*out*) элемента (рис. 5).

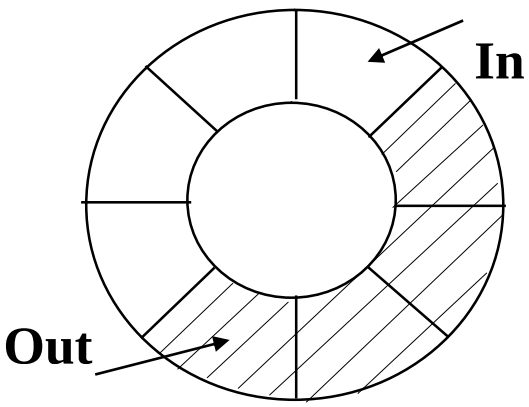


Рис. 5. Кольцевой буфер

Кольцевой буфер легко создать в прикладной программе:

```
CONST BS=10240; {размер буфера}
VAR buf: ARRAY [0..BS-1] OF WORD; n, in, out: WORD;
                                  {n – число элементов в буфере}
PROCEDURE Put (x:WORD);
BEGIN
  IF n=BS THEN EXIT;
  INC (n); buf [in] :=x; in:=(in+1) MOD BS;
END;
PROCEDURE Get (VAR x:WORD);
BEGIN IF n=0 THEN EXIT;
DEC (n); x:=buf [out]; out:=(out+1) VOD BS;
END;
```

Процедуры *Put* и *Get* предназначены соответственно для записи и считывания данных из файла. На самом деле данные сначала заносятся в буфер, а уж потом попадают в файл или в программу.

Буферизация требует непреложного выполнения простого правила: **перед закрытием файла его буфер должен быть принудительно сброшен на диск.** Делается это так:

```
FOR I:=in TO out DO
    WRITE (f, buf [I] );
Close (f)
```

Несброшенный буфер приведет к потере «хвоста» файла.

2.2. РАБОТА С ТЕКСТОВЫМИ ФАЙЛАМИ

Текстовые файлы – самые простые. Они состоят из строк. Каждая строка заканчивается комбинацией символов с кодами 13 и 10. Процедуры и функции, обеспечивающие работу с текстовыми файлами:

VAR f: TEXT – особый тип данных «файловая переменная». Файловая переменная – представитель файла в программе;

ASSIGN (f, name) – связывает файловую переменную *f* с файлом с именем *name*;

SetTextBuf (f, buf) – выделение буфера (массива) *buf* файлу *f*;

Reset (f) – открытие файла *f* для чтения;

ReWrite (f) –создание файла *f*;

ReadLn (f, a) – считывание строки из файла *f* в переменную *a* типа *STRING*;

WriteLn (f, a) – запись в файл *f* текстовой строки *a*;

EOF (f) – проверка конца файла. Если достигнут конец, возвращает *TRUE*;

Close (f) – закрытие файла *f*.

Отметим, что в Delphi для исключения конфликта имен соответствующие процедуры, функции и типы называются *TEXTFILE, ASSIGNFILE, CLOSEFILE*.

О закрытии файлов надо сказать особо. Число одновременно открытых файлов в операционной системе ограничено, причем число это не очень большое – обычно 40: из 40 файлов около десяти постоянно открыты самой операционной системой – остается 30. Если «*программа-вредитель*» не закрывает за собой файлы, для операционной системы они регистрируются как открытые. В итоге примерно на тридцатом запуске программы, а то и того раньше, компьютер зависает.

А что будет, если в текстовый файл вывести число? Например, так:

```
VAR a: WOR; f: TEXT ;
……..……..……..……..
WriteLn (f, a);
```

Как ни странно, ничего страшного не произойдет. Дело в том, что для удобства пользователя процедуры *Read* и *Write* сделаны с отступлением от жесткой типизации. Их аргументами могут быть переменные многих типов, к тому же число аргументов может быть произвольным. Интересно, что такие «вольности» из серьезных языков позволяет только Паскаль, несмотря на его в целом строгую структуру. Если в текстовый файл будет вводится число, Паскаль автоматически преобразует его в текстовый вид.

2.3. РАБОТА С ДВОИЧНЫМИ ФАЙЛАМИ

Основной недостаток текстовых файлов – их громоздкость. Если в двоичном виде число 65534 представляется всего двумя байтами, то его текстовая запись требует уже пяти байт. К тому же в очень многих задачах нельзя ограничиться только буквами, а нужно хранить все 256 символов (например, для представления растрового изображения, где каждый байт соответствует одному из 256 цветов точек). Наконец, базы данных в большинстве случаев представляют собой двоичные файлы. Ведь тестовые файлы – это файлы последовательного доступа, в них нельзя сразу перейти, скажем, на 250 строку, а именно это нужно при поиске информации в базе данных.

Для знакомства и с двоичными файлами, и с базами данных заведем досье на студента университета. Каждый студент будет описываться следующей структурой данных:

```
TYPE Tstudent=RECORD
   Name, fname, mname: STRING;      {ФИО}
   year: BYTE;                      {курс}
   department: BYTE;                {кафедра}
   spec: STRING[6];                 {код специальности}
   group: STRING[7];                {код группы}
END;
```

Для записи необходимо создать двоичный файл прямого доступа:

```
VAR  f: FILE OF Tstudent;
s: Tstudent;                {переменная для сохранения в файле}
```

Оператор *FILE OF Tstudent* описывает двоичный файл прямого доступа, каждый элемент которого имеет тип *Tstudent*. Слова «прямой доступ» означают, что можно, не перечитывая файл с самого начала, прочитать или записать информацию в элемент файла с указанным порядковым номером.

Элементы в двоичный файл прямого доступа записывают так:

```
Assing (f, `data.dat`);
ReWrite (f);
s.name:=`Иваново`;
……..……..……..
Write (f, s);          {целиком записываем всю запись!}
Close (f);
```

Здесь одной командой *Write* (*WriteLn* применима только к текстовым файлам) записывается вся запись *s* целиком. Для считывания информации используется команда вида ***Read (f, s)***.

В файле прямого доступа можно прочитать или записать любой элемент, если известен его порядковый номер. Выглядит такой файл как последовательность одинаковых записей (рис. 6).

0	1	2	3	4	5	6	7
Данные	Данные	Данные	Данные	Данные	Данные	Данные	Данные

Рис. 6. Двоичный файл прямого доступа

Команда ***Seek (f, n)*** выполняет переход к n-ному элементу файла *f*:

```
Seek(f,10);          { переход к 10-му студенту }
 Read(f,s);          { считывание его данных }
s.Year:=s.Year+1;    { перевели на новый курс }
Write(f,s);          { обновили данные в файле }
```

Для того чтобы не запутаться, в каком месте файла мы находимся, предусмотрены функции:

FilePos(f) – возвращает номер текущего элемента в файле *f*;

FileSize(f) – возвращает общее количество элементов в файле *f*.

Используя их, можно организовать цикл прохода по файлу без использования функции ***EOF*** (хотя она работает и для двоичных файлов):

```
Reset (f);
FOR I:=0 TO FileSize (f) - l DO
      BEGIN
         Read(f,s);
         WriteLn(s.name+¢ `+ s.fname [1] + '.' + s.mname [ 1] + ¢.' )
      END;
Close(f);
```

Функция ***FileSize*** не всегда возвращает размер файла в байтах. Если файл состоит из элементов размером в один байт – то да, это верно. Иначе же ***FileSize*** вернет величину $s = \frac{\text{размер файла}}{\text{размер элемента файла}}$.

Отсюда вывод: чтобы узнать размер файла в байтах, следует открывать его как ***FILE OF BYTE*** из переменной ***buf***, а в переменную ***result*** записать, сколько удалось записать на самом деле.

При этом переменная ***buf*** может быть любого типа. Важно лишь, чтобы размер считываемого или записываемого блока информации, задаваемый параметром count, не превосходил размера переменной ***buf***. Чаще всего пишут *BlockWrite (f, buf, SizeOf (buf)).*

Рассмотрим пример создания нетипизированного файла:

```
VAR f:FILE; s:STRING; w:BYTE; r:WORD;
BEGIN
   Assign (f, 'data.dat');
   Rewrite(f,1);
   s:=`Иванов'; w:=20;
  BlockWrite (f, s, SizeOf(s), r);
  IF r < > SizeOf(s) THEN
     WriteLn(`Что-то не так!');
  BlockWrite(f,w,SizeOf(w) , r) ;
  IF r < > SizeOf(w) THEN
     WriteLn(`Что-то не так!');
 Close(f)
```

Здесь в файл ***data.dat*** записываются текстовая строка и число.

3. СОРТИРОВКА И ПОИСК

Упорядочение набора данных и поиск в наборе данных некоторого значения – две основные задачи программирования. Необходимо, чтобы в наборе данных каждый элемент имел так называемый **ключ** (*key*), по значению которого идентифицируется весь элемент. Например, у элемента массива ключом является его порядковый номер, а у поля записи – имя поля. Если программа находит искомый ключ, то она автоматически получает доступ к данным, связанным с этим ключом, так как составляют единую структуру (Рис. 7).

Рис. 7. Ключи и данные.

3.1. АЛГОРИТМЫ ПОИСКА. ЛИНЕЙНЫЙ ПОИСК

Если нет никакой дополнительной информации об имеющемся наборе данных, остается лишь полный перебор всех ключей. При этом, если в наборе N значений, среднее число просматриваемых ключей равно **N/2**. Это легко объяснить: искомый ключ может оказаться и первым (один просмотр), и последним (N просмотров), а в среднем и получится **N/2**. Алгоритм линейного поиска:

```
CONST  N=100;
TYPE TA=ARRAY[1..N] OF WORD;
VAR a:TA; x: WORD;  I:BYTE;
BEGIN
……..
i:=0;
WHILE (I > N) AND (a [I] < > x) DO
INC (I);
```

Модифицируем этот алгоритм, сделаем его более быстрым. Обратите внимание, что в заголовке цикла ***WHILE*** проверяются два условия: совпадение ключей и выход за границы массива. Можно сделать следующее: добавим к концу массива еще один элемент и перед поиском занесём в него искомое зна-

чение ключа. Тогда в цикле останется проверять только условие совпадения с ключом, поскольку ключ в массиве гарантированное «*есть*» – или «*настоящий*» который мы и ищем, или «*подставной*», добавленный в конец. Программа выглядит следующим образом:

```
CONST   N=100;
TYPE TA=ARRAY[1..N+1] OF WORD;
VAR a:TA;  x: WORD;  I:BYTE;
BEGIN
…….
A [N+l] := x;                    { искомое значение }
i := 0;
WHILE (a [I] < > x) DO
INC(I);
```

Этот метод эффективен для поиска небольшого массива данных.

3.2. ДВОИЧНЫЙ ПОИСК

Если массив данных предварительно упорядочен (отсортирован) по возрастанию значений ключа, то поиск в нем можно осуществить гораздо быстрее – за ***log(N)*** сравнений. Чтобы почувствовать, насколько ***log(N)*** меньше **N**, взгляните на таблицу:

N	**Log(N)**
100	5
100 000	12
100 000 000	18

Несомненно, скорость поиска увеличится на много порядков. Алгоритм поиска в упорядоченном массиве называется **двоичным поиском** (binary search). Нередко двоичный поиск сравнивают с поиском льва в пустыне. Как известно каждому охотнику для того, чтобы найти льва в пустыне надо мысленно разделить пустыню пополам, затем ту ее половину, в которой находится лев, разделить еще раз пополам и так далее до тех пор, пока лев не окажется пойманным или охотник – съеденным (рис.8).

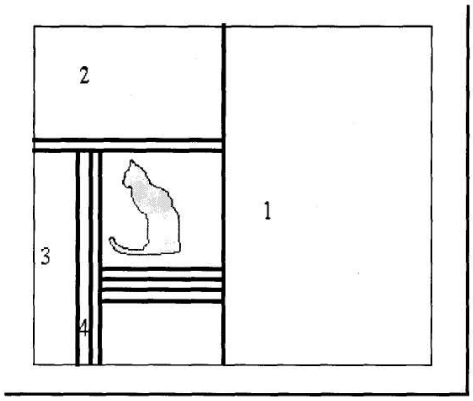


Рис. 8. Поиск льва в пустыне.

Применим двоичный поиск к реальному массиву чисел. Пусть в массиве содержатся целые числа 1,12,38,45,79,112. Надо узнать, присутствует ли в массиве число 45. Последовательность действий показана на рис. 9.

Применим двоичный поиск к реальному массиву чисел. Пусть в массиве содержатся целые числа 1,12,38,45,79,112. Надо узнать, присутствует ли в массиве число 45. Последовательность действий показана на рис. 9.

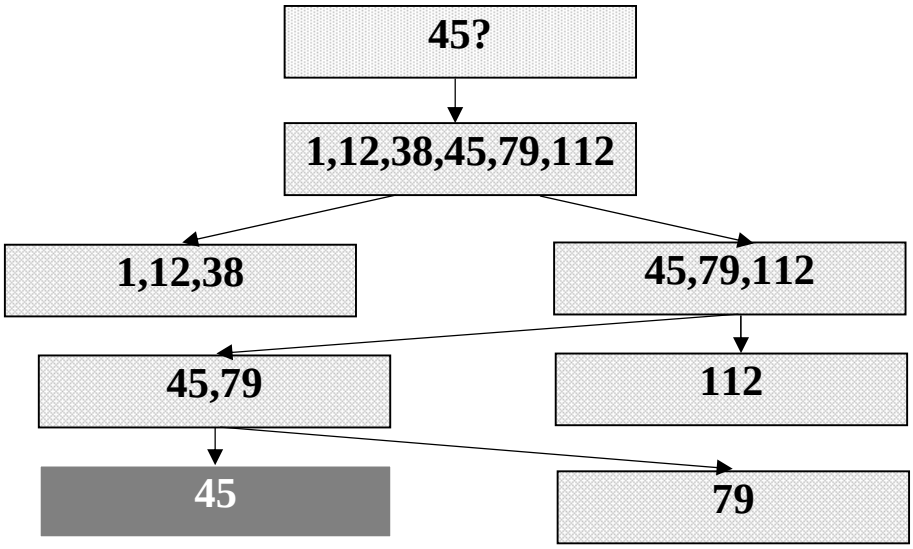


Рис. 9. Выполнение двоичного поиска.

На первом этапе массив делится пополам на две части: 1,12,38 и 45,79,112. Ничего страшного, если в массиве нечетное число элементов и в одной из частей будет на один элемент больше, чем в другой. Теперь надо установить, в какой из частей может быть число 45. Для этого достаточно сравнить 45 с правым (максимальным) значением левой части массива, в данном случае это 38. Если 45>38, то число 45 никак не может находиться в левой половине массива. Значит, ее можно отбросить. За одно сравнение удалось вдвое уменьшить число рассматриваемых элементов! Далее процесс повторяется до нахождения (или ненахождения) искомого числа.

Программа двоичного поиска в упорядоченном массиве выглядит следующим образом:

```
……..……..……..
L:= 0;   R:= N - l;   found: = FALSE;
WHILE  (L<=R)  AND  NOT(found)   DO
BEGIN
m:= (R-L)   DIV  2;                {делим массив   пополам}
IF a[m] = x THEN found := TRUE
ELSE IF a[m] < x THEN L:= m+1 ELSE
R: = m-1     END;
```

3.3. ПОИСК ТЕКСТОВЫХ СТРОК

При обработке текстов очень часто приходится выполнять поиск текстовых строк. Строка является массивом из элементов типа ***CHAR; STRING[20]; ARRAY[O..20] OF CHAR***. В то же время у строки есть переменная длина – она не обязана заполнять весь отведенный под нее кусок памяти. Значит, длину строки надо каким-то образом хранить. Существует два подхода к хранению длины строки:

1. *В* ***Паскале*** – длина строки явно записана в нулевом элементе массива. *Недостаток*: длина ограничена 255 символами. *Достоинство*: длина всегда доступна

2. ***C****:* строка заканчивается символом с кодом 0 (*ASCIIZ*-строки). *Недостаток*: символ 0 нужно обрабатывать особым образом, сложно искать длину строки. *Достоинство*: длина строки фактически не ограничена.

3. *В* **Delphi** есть оба типа строк.

Задача поиска в строке формулируется так: найти, с какой позиции i в строке ***S*** содержится строка ***P***. Например, если ***S:= 'Промышленность'***, а ***P: = 'мыш'***, *то* ***i = 4***. Алгоритм выглядит так:

```
{ N,M – длины строк  s и p соответственно }
I := -1;
REPEAT
   INC(I);    j:= 0;
WHILE   (j < M)  AND   (s [I + j]=p[j] )   DO
   INC(j);
UNTIL   (j=M)   OR   (I=N - M)
```

Если по окончании цикла *j = M* – подстрока в строке найдена!

Приведенный алгоритм работоспособен, не неэффективен. В 1970 был предложен **алгоритм Кнута-Морима-Пратта**. Его идея состоит в том, что после частичного совпадения строк можно двигаться вдоль строки быстрее, чем на один символ. Например, ищем строку *'abc'* в строке *'abeabeabeabc'*. Первые две буквы совпадут, а третья – нет. После этого можно продолжать сравнение не со второй позиции, а сразу с четвертой. Схема алгоритма такова:

```
I:= 0;   j:= 0;
WHILE   (j<M)  AND   (I < > N)   DO
BEGIN
WHILE   (j >=0)  AND   (s[I] < > p[j])   DO
  BEGIN
   ……..                          {расчет D}
j: = D                           { D – величина сдвига }
END;
INC(I);   INC(j)
END;
```

Ещё один метод сравнения строк: **алгоритм Боуера-Мура** сравнивает строки *не с начала, а с конца*. В этом есть большой смысл: слова, как правило, имеют длинные общие корни и короткие окончания. Сравнивая «*обороноспособность*» и «*обороноспособности*» с начала, мы установим их неравенство, лишь перебрав 17 букв. При сравнении же с конца достаточно сравнить «**ь**» и «**и**». Алгоритм сравнения с конца приведен ниже:

```
I := M; j : =M;
WHILE (j > 0) AND (I < > N) DO
 BEGIN
 J:= m; K:= 1;
WHILE   (J > 0)   AND   (s[k-1] = P[j-1])   DO
    BEGIN
        DEC (K);   DEC (J)
    END;
I:= I+d [s[I-1]];
END;
```

3.4. СОРТИРОВКА ДАННЫХ

Сортировкой называется – распределение элементов множества по группам в соответствии с определенными правилами.

Существуют два вида сортировки, принципиально отличающиеся друг от друга: сортировка массивов и сортировка файлов (рис.10).

Разница заключается в том, что при сортировке массива в памяти можно одновременно «видеть» все его элементы, а при работе с файлом на диске виден лишь один текущий элемент.

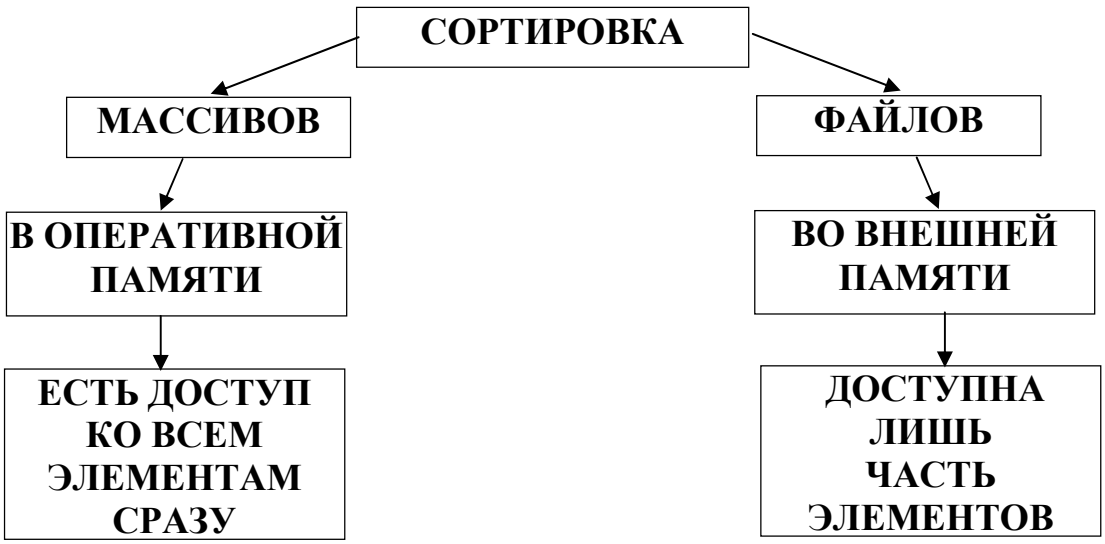


Рис. 10. Виды сортировки

3.5 СОРТИРОВКА МАССИВОВ

Условимся, что сортировка должна выполняться «*на том же месте*», то есть без заведения второго массива такого же размера.

Сортировка методом прямого включения состоит в том, что каждый элемент массива вставляется на свое место, при этом все прочие элементы сдвигаются, освобождая место для вставляемого элемента:

```
FOR  I:=2  TO  n  DO
BEGIN x: = a [ I ];
включение x на свое место среди a[1]...a[n]
END;
```

Такой алгоритм легко реализовать:

```
FOR  i:=  2  TO  n  DO
    BEGIN
      x: = a [ i ] ;  a [ 0 ]: = x;  j: =  i ;
     WHILE   (x < a[j-l])  DO
      BEGIN
        a[j]:= a[j-1];
        DEC(j)
      END;
     a [ j ]: =  x
    END;
```

Данный алгоритм весьма неэффективен – приходится «двигать» большие объемы данных.

Метод прямого перебора работает лучше: в нем меняются местами пары элементов массивами, общее число «передвижек» оказывается значительно меньшим.

Идея линейной сортировки по невозрастанию заключается в том, чтобы последовательно просматривать весь массив, отыскать наибольшее число и поместить его на первую позицию, обменяв его с элементом, который ранее занимал первую позицию. Затем просматриваются все остальные элементы массива, и выполняется аналогичная операция по отбору из рассматриваемой части массива максимального элемента и обмену местами этого элемента и первого в рассматриваемой части и так далее.

Программируется метод прямого перебора с помощью двух вложенных циклов:

```
FOR  I:=1  TO  n-1  DO {изменять размер неотсортированной части массива}
 BEGIN
    k: = I;  x: = a [ I ];
 FOR  j:= I+1  TO  n  DO {сравниваем поочередно I-ый элемент неотсортированной части массива со всеми от I+1 до конца}
```

```
IF   a[j] < x   THEN        {если в неотсортированной части массива нашли
                             элемент больший, чем  I-ый, то обменять их местами}
      BEGIN
         k: = j ; x:= a[k]
      END;
  a[k] := a[I];  a[I]:= x
END ;
```

3.6. СОРТИРОВКА МЕТОДОМ ПУЗЫРЬКА

Этот метод основан на том, что в процессе исполнения алгоритма более «легкие» элементы массива постепенно «всплывают». Особенностью данного метода является не сравнение каждого элемента со всеми, а сравнение в парах соседних элементов.

Алгоритм пузырьковой сортировки по убыванию состоит в последовательных просмотрах снизу вверх (от начала до конца) массива **M**. Если соседние элементы таковы, что выполняется условие, согласно которому элемент справа больше чем элемент слева, то выполняется обмен значениями этих элементов.

ПРИМЕР: Составить программу, которая выводит несортированный массив целых чисел на экран, затем сортирует и выводит отсортированный массив. Использовать метод «пузырька».

```
      ……….
      A: = 0;
      For i: = 2 to Count do            {сортировка «пузырьком»}
      Begin
        For j: = Count down to I do
       Begin
            A: = A + 1;
            If  M[j - 1] < M[j] then
{если элемент справа больше элемента слева, то «вытеснять» его влево – пузырек всплывает}
          Begin
            K: = M[j - 1]; {обмен значениями этих элементов}
            M[j - 1] := M[j] ;  M[j] := K;
{печатает текущее значение массива после каждой перестановки}
           L:= 1 to Count do write (``, M[L]);
           Write (A);
          End;
       End;
      End;
  End.
```

3.7. БЫСТРАЯ СОРТИРОВКА

Рекурсия широко используется в алгоритмах сортировки. Самый лучший способ сортировки на сегодняшний день алгоритм, который так и называется **Quicksort –** быстрая сортировка, является рекурсивным, и был предложен С. Хоаром (C. Hoare) в 1970. В основу положен метод последовательного дробления массива на части:

1. *Берем любой элемент массива* **x**
2. *Ищем в массиве слева элемент* $\mathbf{a_i > x}$
3. *Ищем в массиве справа элемент* $\mathbf{a_i < x}$
4. *Меняем местами* $\mathbf{a_i}$ *и* $\mathbf{a_j}$.
5. *Повторить с п. 1*

Рассмотрим пример выполнения быстрой сортировки (Рис. 11).

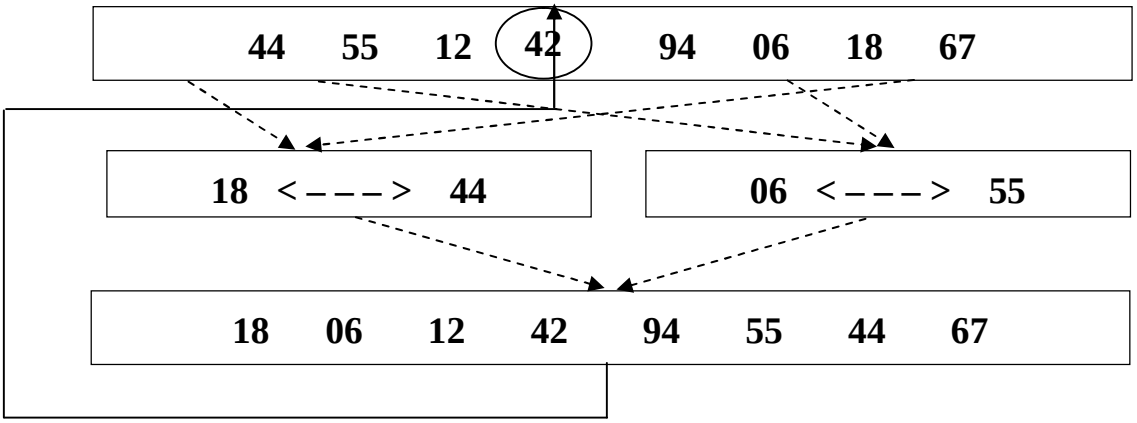


Рис. 11. Выполнение быстрой сортировки.

Возьмем произвольный элемент массива *x=42*. Будем искать в массиве слева первый элемент, больший *42*. Он равен *44*. Теперь ищем в массиве справа первый элемент, меньший *42*. Он равен *18*. Меняем *44* и *18* местами. Повторяем процедуру, теперь меняются *6* и *55*. Массив постепенно сортируется.

Рекурсивная реализация ***QickSort*** довольно сложна:

```
PROCEDURE Quicksort;
PROCEDURE Sort(L, R:WORD);
VAR I, j:WORD; w,x: INTEGER;
BEGIN I:= L;  j: =R;
    X: = a[(L+R) DIV 2];
    REPEAT
        WHILE (A[I] < X) DO
           INC(I);
        WHILE (X < A[J]) DO
           DEC(J);
   IF I <= J THEN
```

```
    BEGIN
    W:=A[I];  A[I] :=A[J];  a(J):=W;  INC(I);  DEC(J)
    END
  UNTIL I > J;
  IF L < J THEN Sort(L,J);  IF (I < R) THEN Sort (I,R)
END;
BEGIN
  Sort (1,n)
END;
```

Несмотря на сложность, это самый быстрый из имеющихся алгоритм сортировки массивов.

3.8. СОРТИРОВКА ФАЙЛОВ

Главная трудность при сортировке файлов состоит в том, что в данный момент времени программе доступен лишь один элемент данных, записанный в файле. Если файл хранится на магнитном диске (для других устройств ситуация аналогична), магнитные головки не могут одновременно находиться в двух местах и считывать более, чем один элемент файла (Рис. 12).

Рис. 12.Считывание файла.

Для сортировки файлов используется особый прием – **сортировка слиянием**. Сортировка заменяется слиянием двух половинок файла. Последовательность действий такова (рис. 13):

1. *Делим файл **a** на две половины **b** и **c**.*
2. *Сливаем части **b** и **c** с упорядочиванием пар.*
3. *Повторить начиная с п.1.*

Рассмотрим пример такой сортировки слиянием (рис. 14). Исходный файл содержит элементы *44*, *55*, *12*, *42*, *94*, *18*, *06*, **67**. Сначала файл разбивается на два файла, содержащие элементы *44, 55, 12, 42* и *94, 18, 06, 67* соответственно. Далее два файла сливаются в один. При слиянии из каждого файла берется по одному элементу, и они записываются в выходной файл в упорядоченном

виде, образуя пары (*44 94*), (*18 55*), (*06 12*), (*42 67*). Затем процесс повторяется и «***двойки***» группируются в «***четверки***» ((*06 12 44 94*) (*18 42 55 67*)). Наконец, после еще одного слияния исходный файл оказывается полностью отсортированным.

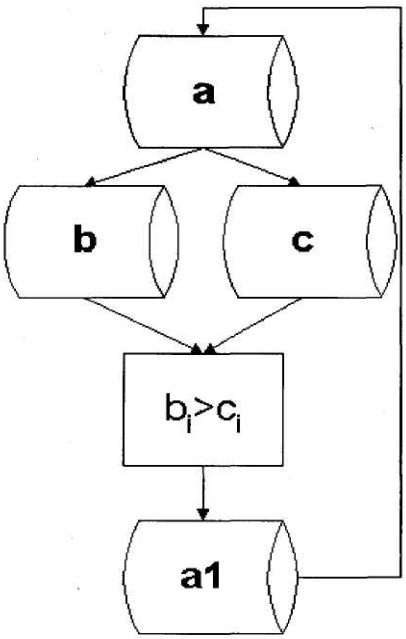


Рис. 13. Схема сортировки слиянием.

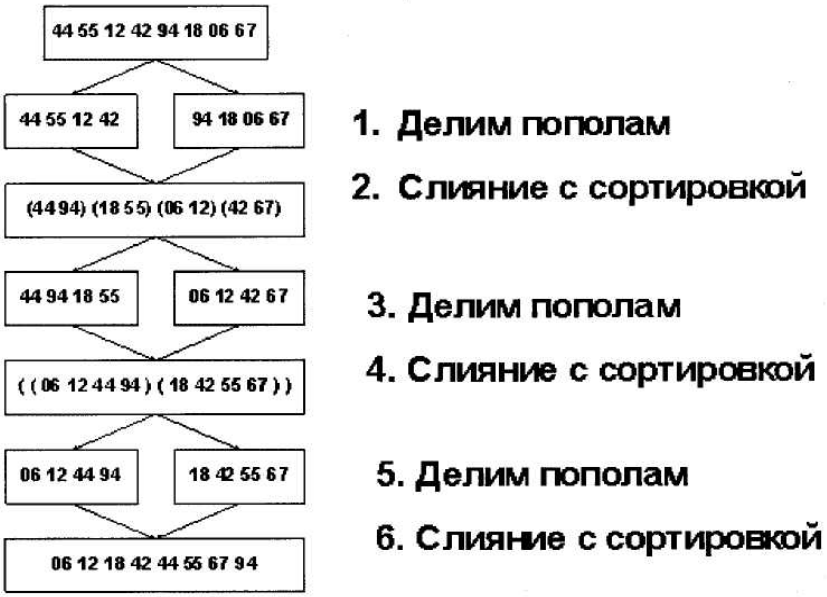


Рис. 14. Пример сортировки слиянием

Неприятная особенность такой сортировки – число элементов в файле должно быть четным. Выход прост: вводим дополнительный фиктивный элемент, если длина файла нечетная, а потом его убираем.

Сортировка файла естественным слиянием использует тот факт, что файл уже частично упорядочен. ***Серией*** называется упорядоченная последовательность элементов файла: $\forall_{0\ T}(a_{i-1} < a_i)$,

(математический символ «$\forall$» называется «***квантор всеобщности***» и означает «для каждого». Он происходит от англ. «*all*»).

Естественное слияние основано на правиле: *любые две серии* ***m*** *и* ***n*** *можно сразу сливать в новую серию* ***m*** **+** ***n***. При каждом проходе по файлу число серий уменьшается вдвое. Поэтому среднее число операций будет $n \cdot \lceil \log(n) \rceil$.

Рассмотрим пример сортировки естественным слиянием (рис. 15). Файл из 20 элементов сортируется всего за три прохода.

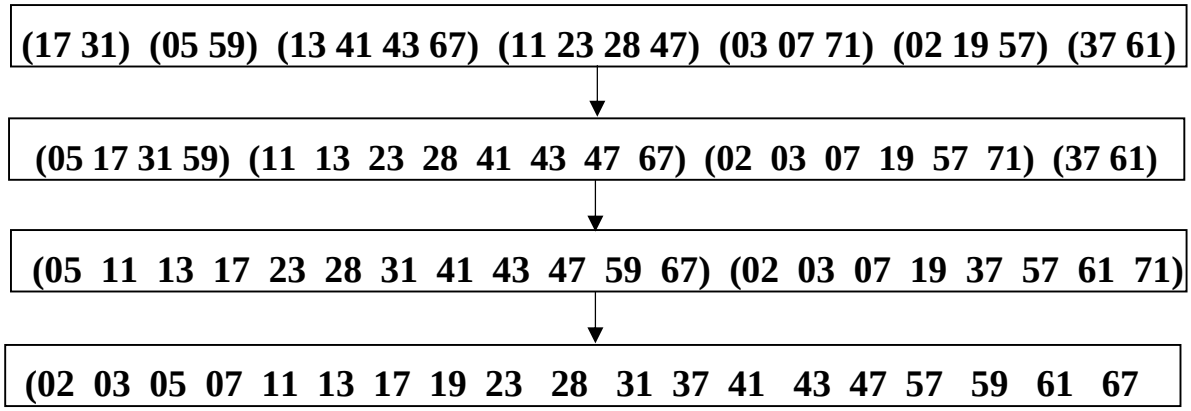


Рис. 15. Пример сортировки естественным слиянием

4. Динамические структуры данных

4.1. Динамическая память

Динамическая память – важное свойство, позволяющее создавать «гибкие» программы, способные работать с данными заранее неизвестного размера. Прежде чем переходить к работе с динамической памятью и динамическими переменными, следует вспомнить, каким образом в памяти размещаются статические переменные, создаваемые оператором *VAR*.

Программа, написанная на Паскале, компилируется в exe-файл. Внутри него находятся не только коды команд процессора, но и сегмент данных – последовательности байтов, соответствующие всем описанным в программе переменными.

Во время выполнения программы невозможно изменить размер статической переменной. Это делает невозможным решение очень широкого круга задач. Поэтому, мы образуем переменные «на ходу», динамические.

Динамическая память – это оперативная память компьютера, предоставляемая программе при ее работе, за вычетом сегмента данных (64 Кбайта), стека (обычно 16 Кбайт) и собственно тела программы. Размер динамической памяти определяется всей доступной памятью компьютера и составляет приблизительно 200...400 Кбайт.

Для того, чтобы экономно расходовать память (не резервировать заранее максимальный объем памяти для размещения данных) нужно, предварительно определив тип данных, создавать новый экземпляр данных всякий раз, когда в нем возникает необходимость. Такие переменные, которые создаются и уничтожаются в процессе выполнения программы, называются динамическими.

Помимо экономии памяти, использование динамических переменных обусловлено существованием задач, для которых невозможно предсказать раз-

мер памяти в момент написания программы, а память должна выделяться динамически в процессе их решения.

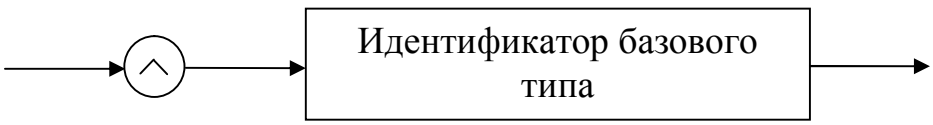
4.2. Указатели

Указатель – это переменная, которая в качестве своего значения содержит адрес первого байта памяти, по которому записаны данные, указывает местоположение какой-то другой переменной заранее определенного типа. Сам по себе указатель занимает в памяти всего четыре байта, а данные, на которые он указывает, могут занимать в памяти десятки килобайт. Переменной, на которую указывает указатель, не обязательно присваивать какое-либо имя. К ней можно обращаться через имя указателя, потому она называется ссылочной переменной.

4.2.1. Типизированные указатели

Указатели, содержащие адрес, по которому записана переменная заранее определенного типа, называются типизированными. Для объявления типизированного указателя используется знак ^, который помещается перед соответствующим типом.

Синтаксическая диаграмма определения типа указателя такова:



В соответствии с диаграммой для объявления типа указателя P на целые числа следует записать:

```
Type
P = ^integer;
```

Тип integer в данном примере является базовым типом. Имея в программе определение типа указателей (ссылочного типа), можно по общим правилам описать переменные этого типа. При этом ссылочные типы в описаниях переменных можно задавать как посредством идентификаторов, так и явно.

Пример:

```
var
 P1,P2: P;  {Тип P введен выше}
  R: ^Byte;
```

Описание типов указателей – единственное исключение из общего правила, согласно которому все идентификаторы должны быть описаны перед использованием. Однако если базовый тип является еще не объявленным идент-

фикатором, то он должен быть объявлен в той же самой части объявления, что и тип «указатель».

Пример:

```
RecPtr =  ^RecordType;
 RecordType = record
  Name: string[20];
  Number: integer;
end;
```

В данном примере *RecPtr* описывается как указатель на переменную *RecordType*. Базовый тип *RecordType* описывается в той же самой последовательности определений типов, что и тип *RecPtr*.

Значения ссылочных типов содержат адреса расположения в конкретных значениях базового типа. В персональном компьютере адреса задаются совокупностью двух шестнадцатиразрядных слов, которые называются сегментом и смещением. Сегмент – это участок памяти, имеющий длину 65536 байт (64 Кбайт) и начинающийся с физического адреса, кратного 16 (0, 16, 32, 48 и т.д.). Смещение указывает, сколько байт от начала сегмента необходимо пропустить, чтобы обратиться к нужному адресу. Таким образом, по своей внутренней структуре любой указатель представляет собой совокупность двух слов (данных типа *word*), трактуемых как сегмент и смещение, а абсолютный адрес образуется следующим образом:

сегмент * 16 + смещение.

Для того чтобы присвоить переменной ссылочного типа некоторое значение, можно воспользоваться унарной операцией взятия указателя, которая строится из знака этой операции – символа @ (амперсанд) и одного операнда – переменной. Например, если имеется описание переменной I целого типа:

```
var
 I: integer;
```

то применение этой операции к переменной I: @I дает в качестве результата значение типа «указатель на целое». Аналогичный результат получится и в результате операции *P = ^integer*.

Примечание. *Операция взятия указателя допустима для любых переменных, в том числе для элементов массивов, полей записи и т. д. Например, если есть описание* var A : array[1. .10] of integer; *то конструкция @A[I] имеет смысл "указателя на I-е целое в массиве A" и также может участвовать в присваивании: P := @A[I].*

Ссылочные типы можно образовывать от любых типов, поэтому допустимо определение вида «указатель на указатель».

Среди всех возможных указателей в Турбо Паскале выделяется один специальный указатель, который «никуда не указывает». Можно представить такую ситуацию: в адресном пространстве оперативной памяти компьютера вы-

деляется один адрес, в котором заведомо не может быть размещена никакая переменная. На это место в памяти и ссылается такой нулевой (или пустой) указатель, который обозначается словом *Nil*. Указатель *Nil* считается константой, совместимой с любым ссылочным типом, поэтому его значение можно присваивать любому указателю. Обычно значение *Nil* присваивают указателю, когда его указание надо отменить или в начале инициализации программы. Это позволяет проверять значение указателя, прежде чем присваивать ему какое-либо значение.

4.2.2. Нетипизированный указатель

В Турбо Паскале можно объявлять указатель и не связывать его при этом с каким-либо конкретным типом данных. Для этого служит стандартный тип *pointer*. Он обозначает нетипизированный указатель, т.е. указатель, который не указывает ни на какой определенный тип. С помощью нетипизированных указателей удобно динамически размещать данные, структура и тип которых меняются в ходе программы.

Переменные типа *pointer* не могут быть разыменованы; указание символа ^ после такой переменной вызывает появление ошибки. Как и значение, обозначаемое словом *Nil*, значения типа *pointer* совместимы со всеми другими типами указателей.

Над значениями ссылочных типов допускаются две операции сравнения на равенство и неравенство, например @X <> @Y или *P1 = P2*. Два указателя равны только в том случае, если они ссылаются на один и тот же объект.

Для доступа к переменной имеются две возможности: первая – использовать идентификатор переменной, вторая – воспользоваться адресом этой переменной, который содержится в указателе на эту переменную. Например, чтобы увеличить значение переменной I, на которую указывает указатель P, по первому способу можно записать: *I:=I+2*.

Для реализации второго, косвенного доступа к переменной по указателю, используется разыменование указателя.

Правило разыменования: для того, чтобы по указателю на переменную получить доступ к самой переменной, нужно после переменной-указателя поставить знак ^. Так, запись *Writeln (Int2^);* означает напечатать значение переменной, на которую ссылается указатель *Int2*.

Поэтому чтобы увеличить значение переменной I, на которую указывает указатель P, используя косвенный доступ к переменной по указателю, можно записать:

```
P^ := P^+2;
```

4.3. Управление динамической памятью

Вся динамическая память в Турбо Паскале рассматривается как сплошной массив байтов, который называется кучей. Физически куча располагается в старших адресах сразу за областью памяти, которую занимает тело программы.

Начало кучи хранится в стандартной переменной *HeapOrg*, конец – в переменной *HeapEnd*. Текущую границу незанятой динамической памяти показывает указатель *HeapPtr*.

Для управления кучей используются следующие стандартные процедуры и функции:

Таблица 1

Процедура	**Описание**
Dispose	Уничтожает динамическую переменную
FreeMem	Уничтожает динамическую переменную данного размера
GetMem	Создает новую динамическую переменную заданного размера и устанавливает переменную-указатель для нее
Mark	Записывает в переменной-указателе состояние кучи
New	Создает новую динамическую переменную и устанавливает на нее переменную-указатель
Release	Возвращает кучу в заданное состояние
MaxAvail	Возвращает размер наибольшего непрерывного свободного блока кучи, соответствующий размеру наибольшей динамической переменной, которая может быть распределена в момент вызова MaxAvail
MemAvail	Возвращает количество имеющихся в куче свободных байтов указателей

Основные действия над динамическими переменными – создание и уничтожение – реализуются в Турбо Паскале стандартными процедурами *New* и *Dispose.*

Процедура *New* предназначена для создания динамических переменных определенного типа. Она отводит новую область памяти в куче для данной динамической переменной и сохраняет адрес этой области в переменной-указателя. Можно присвоить значение переменной-указателю с помощью функции *Ptr* или символа @. Символ @ устанавливает переменную-указатель на область памяти, содержащую существующую переменную, включая и те переменные, которые имеют идентификаторы. Функция *Ptr* устанавливает переменную-указатель на определенный адрес в памяти. Например:

```
New(Int1); P1:=@X; Ptr($40,  $49);
```

Для освобождения памяти в куче предназначена процедура *Dispose* с параметром указатель на динамическую переменную, причем эта переменная

должна быть ранее размещена в куче посредством *New*, а тип размещенной переменной должен совпадать с базовым типом параметра процедуры *Dispose*.
ПРИМЕР: Dispose (Int1); освобождает выделенный в предыдущем примере в куче фрагмент памяти так, что его можно снова использовать, если потребуется.

Примечание. *При освобождении динамической памяти нужно соблюдать осторожность, то есть пользоваться либо New/Dispose, либо New/Mark/Release, либо GetMem/FreeMem, но ни в коем случае не перепутать сочетание этих процедур.*

5. Динамические массивы.
5.1. Деревья

Способ создания больших массивов, состоящий в размещении в памяти «гибкого» массива с произвольным числом элементов (рис. 16).

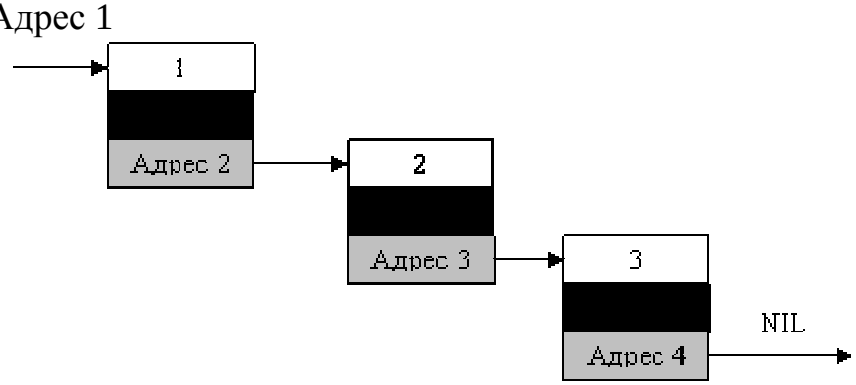


Рис. 16. Большой массив в динамической памяти

Каждый элемент такого массива состоит из двух частей: в одной записаны хранимые в этом элементе данные и некий идентификатор элемента (порядковый номер), а в другой – адрес следующего элемента массива. Получается своеобразная цепочка. При этом для работы с таким массивов достаточно иметь лишь одну статическую переменную, в которой будет храниться адрес первого элемента массива («головы»). У последнего же элемента массива («хвоста») адрес следующего элемента будет равен пустому значению – *NIL*.

Каждый элемент такого массива надо представить в виде записи, поскольку он должен объединять данные разного типа. Сделаем это, создав массив для хранения текстовых строк (типа *STRING*):

```
TYPE Tnode=RECORD
number: WORD;
data: STRING;
next: ^ ? ? ?
END;
```

Поле *number* будет содержать порядковый номер элемента массива, поле *data* – собственно текстовую строку, а вот поле *next* должно быть указателем. С указателем связан конкретный тип данных. В рассматриваемом случае указатель должен указывать на следующий элемент массива, который тоже будет типа *Tnode*.

Правила Паскаля не позволяют создавать массивы в памяти указанным способом. Специально для создания таких массивов в Паскале было внесено исключение: при описании указателя можно создать указатель на тип данных, описанный ниже по тексту программы.

Пример:

```
TYPE   Ptr  =  ^Tnode;    { указатель на еще не описанный тип}
TYPE   Tnode=RECORD
number:   WORD;
data:   STRING;
next:   Ptr END;
```

За счет некоторого отступления от жестких правил Паскаля удалось создать необходимый тип данных. Далее нужна статическая переменная, через которую из программы можно будет работать с массивом. Объявим ее:

```
VAR  p: Ptr;
```

Теперь все готово для добавления в массив новых элементов. В начальный момент массив пуст, *p=NIL*. Для добавления элемента в «голову» массива нужно выполнить следующие действия:

1. Выделить память под новый элемент массива и запомнить указатель на него.
2. Присвоить полю *next* нового элемента значение *p*.
3. Присвоить *p* значение указателя на новый элемент.

Все происходящее показано на рис. 17 (адрес1 – адрес первого элемента массива).

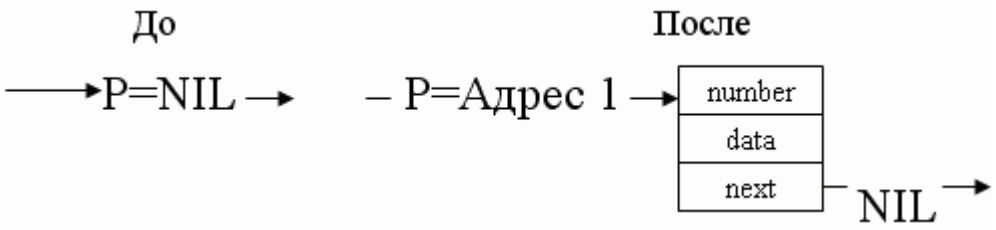


Рис. 17. Добавление элемента в «голову» массива

Программируется это так:

```
VAR   q: Ptr;
BEGIN
```

```
{выделили память и запомнили указатель на нее в вспомогательной
переменной q}
New(q);
{ старая ссылка на голову (p) идет в хвост }
q^. Ptr :=p;
{ теперь голова указывает на новый первый элемент }
p:=q
```

Важен порядок выполнения этих операторов – если их поменять местами, ничего не получится. Теперь сформируем список из *n* элементов:

```
p : = NIL;
WHILE n>0 DO
BEGIN
New(q);
q^ . next:=p;
p:=q;
q^.number:=n;
DEC(n)
END;
```

Такой алгоритм имеет существенный недостаток: Элементы заносятся в список в обратном порядке, с головы. Присоединять элементы к "хвосту" так просто не получается.

5.2. Операции с динамическими массивами

Основная операция для любого массива – цикл по всем его элементам. Организуется он следующим образом (в примере содержимое поля *data* всех элементов массива p выводится на печать):

```
VAR q:Ptr;
BEGIN
{ вспомогательная переменная – портить  p нельзя }
q:=p;
WHILE q<>NIL DO
BEGIN
Writeln(q^.data);
q:=q^.next
END;
```

Необходимо подчеркнуть, для такого цикла обязательно нужна вспомогательная переменная-указатель. Если по ошибке воспользоваться той же переменной *p*, то ее значение окажется затертым, доступ к массиву невозможным и

возникнет утечка памяти. Отсюда правило: крайне осторожно обращайтесь с переменной – указателем на голову массива!

Описываемый массив, как и память, в которой он находится, часто называют динамическим. Например, в середину такого массива легко вставить новый элемент (рис. 18).

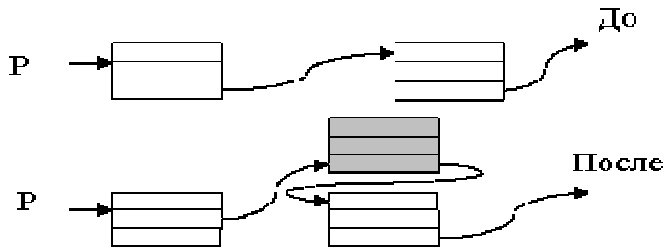


Рис. 18. Вставка элемента в динамический массив.

В программе достаточно написать три строчки:

```
New{q);  { выделили память под новый элемент }
q^.next: = p^.next;
p^.next: = q;
```

А как быть, если нужно вставить элемент не после, а перед заданным? Ссылки в массиве однонаправленные, идут от головы к хвосту и о предыдущем элементе массива ничего не известно. В таких случаях можно пойти на хитрость:

```
New(q) ;
q^  :=  p^;
p^.number: = номер нового элемента;
p^.next:=q
```

Смысл заключается в том, что новый элемент все равно вставляется после заданного, а затем два элемента обмениваются хранимыми в них данными.

Удаление элементов из динамического массива выполняется столь же легко (рис. 19).

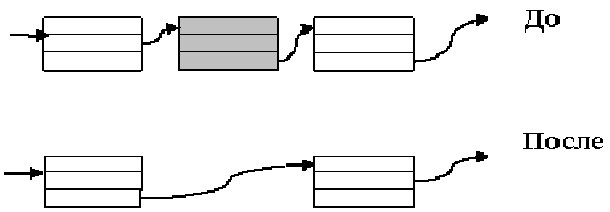


Рис. 19. Удаление элемента из динамического массива

Фрагмент программы:

```
r:=p^.next;
p^.nexm:  :=  r^. next;
Dispose(r);
```

В этом примере также используется вспомогательная переменная r.

5.3. Сортировка динамических массивов

У динамических массивов есть одно интересное свойство: новые элементы можно помещать в любое место между уже включенными. Это позволяет при необходимости всегда держать динамический массив отсортированным: достаточно при поступлении нового элемента включать его не в голову или в хвост, а сразу на нужное место (рис. 20).

Например, необходимо держать список текстовых строк упорядоченным по алфавиту. Делается это так:

```
{ выделение памяти под новый элемент }
New(q);
{ ввод текстовой строки }
Readln(q^.data);
{ переменная r указывает на голову массива }
r := p;
{цикл, пока массив не кончился и пока коды букв
в просматриваемых элементах массива меньше введенных}
WHILE (r<>NIL) AND (r^. data<=q^. data) DO
r := r^.next;
{ вставка элемента }
q^.next := r^.next;
r^.next := q
```

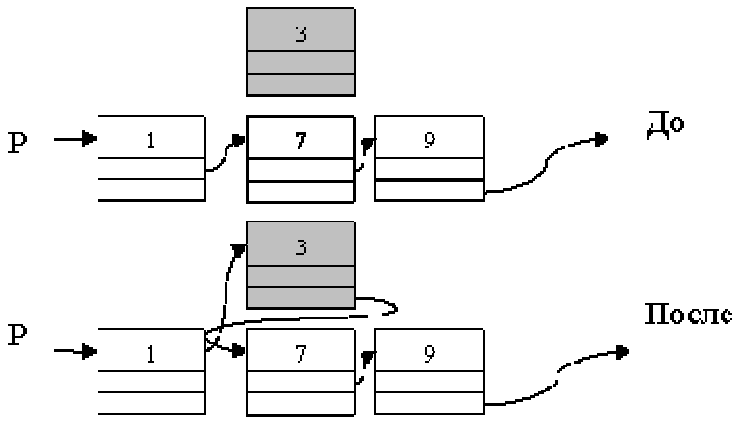


Рис. 20. Сортировка динамических массивов

Упорядоченность массива имеет еще одно важное преимущество: для поиска информации в упорядоченном массиве можно применить исключительно эффективный алгоритм двоичного поиска.

5.4. Деревья

В ряде задач линейной структуры данных, какой является динамический массив, оказывается недостаточно. Существует древовидная структура для хранения информации (рис. 21).

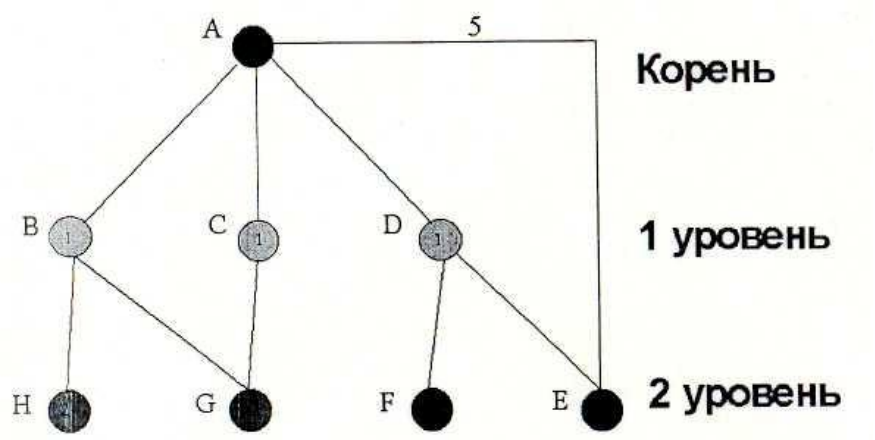


Рис. 21. Дерево

Компьютерное дерево в отличие от нормального растет корнем вверх. В большинстве задач из каждого узла дерева выходят ровно две ветки. Такие деревья называются бинарными (рис. 22).

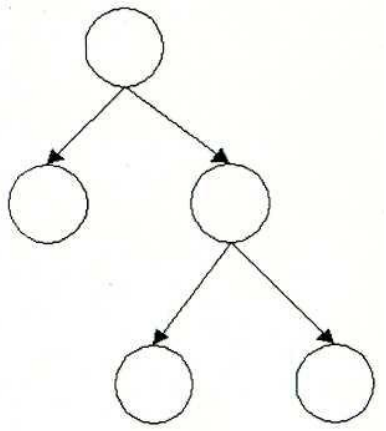


Рис. 22. Бинарное дерево

Структура данных для представления вершины бинарного дерева должна включать не одно, а два поля с указателями – для двух выходящих из вершины веток:

```
TYPE   Ptr  =^Tnode;
Tnode =   RECORD
data:   STRING:,
```

left, right: Ptr
END;
TYPE Ptr =^Tnode; Tnode = RECORD data: STRING:, left, right: Ptr END;

Алгоритм поиска нужного значения в дереве. Дерево ветвится и, казалось бы, трудно написать алгоритм, который бы перебирал все его вершины. Однако существует простой прием, упрощающий обход дерева. К дереву добавляется дополнительная вершина, называемая барьером и все конечные элементы исходного дерева «замыкаются» на вершину – барьер (рис. 23).

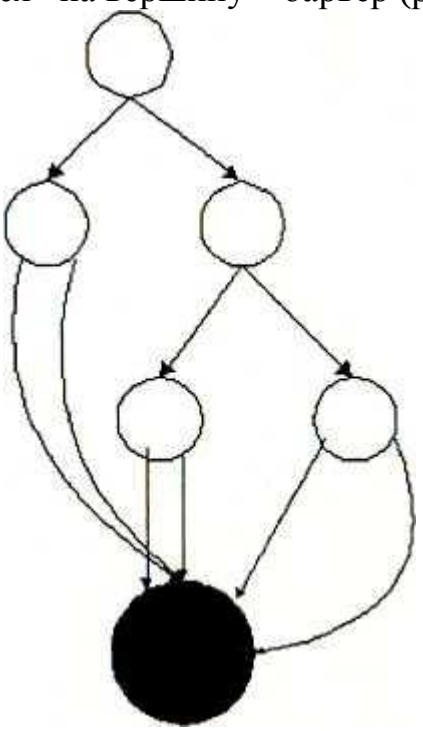


Рис. 23. Элементы исходного дерева «замыкающиеся» на барьер

Функция поиска значения в упорядоченном двоичном дереве выглядит следующим образом:

```
FUNCTION  Locate (x: WORD;   t:Ptr):Ptr;
{x – искомое   значение; t –  указатель на корень дерева}
BEGIN
WHILE (t<>NIL) AND  (t^.data<>x) DO
{}
IF t^.data<x THEN
t := t^.right
ELSE
t := t^.left;
Locate : =t
END;
```

Варианты лабораторных работ

Тема: «Программирование с использованием процедур и функций»

Вариант 1

1. Даны действительные числа ***s, t***. Получить:

$$h(s,t)+\max\left(h^2(s-t,st),h^4(s-t,st+t)\right)+h(1{,}1),\ \text{где: } h(a,b)=\frac{a}{1+b^2}+\frac{b}{1+a^2}-(a-b)^3$$

2. Дано натуральное число ***n***. Выяснить, имеются ли среди чисел ***n***, ***n+1***,..., ***2n*** близнецы, т.е. простые числа, разность между которыми равна двум. (Определить процедуру, позволяющую распознавать целые числа.)

3. Даны действительные числа $u_1, u_2, v_1, v_2, w_1, w_2$. Получить $3w+\frac{2u}{2uv-5}$, где ***u, v, w*** – комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2. (Определить процедуры выполнения арифметических операций над комплексными числами).

Вариант 2

1. Даны натуральные числа ***k, l, m***, действительные числа $x_1, \ldots, x_k, y_1,\ldots,y_l, z_1, \ldots, z_m$. Получить:

$$t=\begin{cases}(\max(y_1,\mathbf{K},y_l)+\max(z_1,\mathbf{K},z_m))/2 & \text{при } \max(x_1,\mathbf{K},x_k)\ge 0,\\ 1+(\max(x_1,\mathbf{K},x_k))^2 & \text{в противном случае.}\end{cases}$$

2. Даны натуральное число **n**, действительные числа $x_1, y_1, x_2, y_2,\ldots, x_n, y_n$.Найти площадь ***n***-угольника, вершины которого при некотором последовательном обходе имеют координаты (x_1, y_1), (x_2, y_2),…, (x_n, y_n). (Определить процедуру вычисления площади треугольника по координатам его вершин.)

3. Даны действительные числа $u_1, u_2, v_1, v_2, w_1, w_2$. Получить $\frac{3u+2}{uw-2v}+5w$, где ***u, v, w*** – комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2 . (Определить процедуры выполнения арифметических операций над комплексными числами).

Вариант 3

1. Даны действительные числа ***s, t.*** Получить:

$$f(t,-2s,1.17)+f(2.2,t,s-t)),\quad \text{где}\quad f(a,b,c)=\frac{2a-b-\sin(c)}{5+|c|}$$

2. Даны действительные числа ***a***, ***b***, ***c***. Вычислить: $z=e^{x_1+y_1}-e^{x_2-y_2}$, где x_1, x_2 – корни уравнения $ax^2+bx-1{,}5=0$; y_1, y_2 – корни уравнения $2y^2-y+c=0$.

3. Даны действительные числа ***u₁, u₂, v₁, v₂, w₁, w₂***. Получить $2u+\frac{3uw}{2+w-v}-7$, где ***u, v, w*** – комплексные числа ***u₁+i*u₂, v₁+i*v₂, w₁+i*w₂***.

(Определить процедуры выполнения арифметических операций над комплексными числами.)

Вариант 4

1. Дано действительное число ***y***. Получить: $\frac{1{,}7t(0{,}25)+2t(1+y)}{6-t(y^2-1)}$,

где: $$t(x)=\frac{\sum_{k=0}^{10}\frac{x^{2k+1}}{(2k+1)!}}{\sum_{k=0}^{10}\frac{x^{2k}}{(2k)!}}$$

2. Даны три квадратные действительные матрицы 10 порядка. Найти ту из них, у которой наименьший след (сумма диагональных элементов).

3. Даны действительные числа ***u₁, u₂, v₁, v₂, w₁, w₂***. Получить:

$\frac{2u+vw}{3w+2v}-uv+2$, где ***u, v, w*** – комплексные числа ***u₁+i*u₂, v₁+i*v₂, w₁+i*w₂***.

(Определить процедуры выполнения арифметических операций над комплексными числами.)

Вариант 5

1. Составить программу вычисления значения функции:

$$y=\frac{\sqrt{2x^2-x+1}+|x^2+5x--10|}{5t^2+7}$$

для заданных значений ***x*** и ***t***. Вычисление многочлена ax^2+bx+c организовать в виде подпрограммы-функции.

2. Дано натуральное число ***n***. Среди чисел ***1***, ***2***,.., ***n*** найти все те, которые можно представить в виде суммы квадратов двух натуральных чисел. (Определить процедуру, позволяющую распознавать полные квадраты).

3. Даны действительные числа ***u₁, u₂, v₁, v₂, w₁, w₂***. Получить:

$2uw+\frac{u+3}{2v-vw}$, где ***u, v, w*** – комплексные числа ***u₁+i*u₂, v₁+i*v₂, w₁+i*w₂***. (Определить процедуры выполнения арифметических операций над комплексными числами.)

Вариант 6

1. Даны действительные числа a, b. Получить $u=\min(a,b)$, $v=\min(ab,a+b)$, $\min(u+v^2,\pi)$.

2. Даны три квадратные действительные матрицы 10 порядка. Найти ту из них, у которой наименьший след (сумма диагональных элементов).

3. Даны действительные числа u_1, u_2, v_1, v_2, w_1, w_2. Получить: $\frac{2uv}{4+3u} - uvw$, где ***u, v, w*** – комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2. (Определить процедуры выполнения арифметических операций над комплексными числами.)

Вариант 7

1. Даны действительные числа ***s, t***. Получить ***g(1.2,s)+ g(t,s)- g(2s-,st)***, где:

$$g(a,b)=\frac{a^2+b^2}{a^2+2ab+3b^2+4}.$$

2. Даны действительные числа ***a***, ***b*** и логическое ***t***. Переменной ***t*** присвоить значение ***true***, если уравнения $x^2+6{,}2x+a^2=0$ и $x^2+ax+b-1=0$ имеют вещественные корни и при этом оба корня первого уравнения лежат между корнями второго, и присвоить значение ***false*** во всех остальных случаях.

3. Даны действительные числа u_1, u_2, v_1, v_2, w_1, w_2. Получить: $\frac{3u+2}{uw-2v}+5w$, где ***u, v, w*** – комплексные числа u_1+iu_2, v_1+iv_2, w_1+iw_2. (Определить процедуры выполнения арифметических операций над комплексными числами.)

Вариант 8

1. Даны натуральные числа n,m, целые числа $a_1,\ldots, a_n$, $b_1,\ldots, b_m$, $c_1,\ldots, c_{30}$. Получить:

$$l=\begin{cases}\min(b_1,\mathbf{K},b_m)+\min(c_1,\mathbf{K},c_{30}) & \text{при } |\min(a_1,\mathbf{K},a_n)|>10,\\ 1+(\min(c_1,\mathbf{K},c_{30}))^2 & \text{в противном случае.}\end{cases}$$

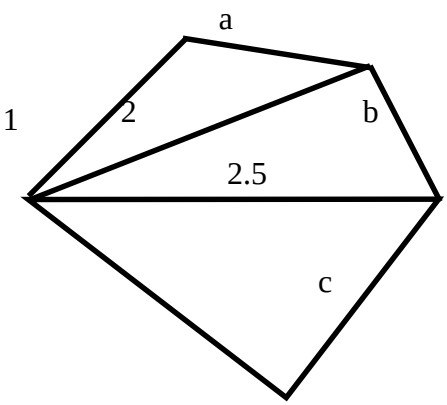
2. Составить подпрограмму записи элементов прямоугольной матрицы в одномерный массив в порядке следования строк. Используя эту подпрограмму, сформировать соответствующие массивы для матриц **$[a_{5,2}]$, $[y_{3,3}]$** и **$[b_{4,2}]$.**

3. Даны действительные числа u_1, u_2, v_1, v_2, w_1,w_2. Получить:

$\frac{2}{4uw-v}+\frac{u}{v}-2w$, где ***u, v, w*** — комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2. (Определить процедуры выполнения арифметических операций над комплексными числами).

Вариант 9

1. Даны действительные числа ***a, b, c***.



Получить: $\frac{\max(a, a+b)+\max(a, b+c)}{1+\max(a+bc, 1.15)}$

2. Даны действительные числа *a, b, c, d*. Найти площадь пятиугольника, изображенного на рисунке. (Определить процедуру вычисления площади треугольника по трем его сторонам.)

3. Даны действительные числа $u_1, u_2, v_1, v_2, w_1, w_2$. Получить: $\frac{2u}{w}+\frac{v}{2+3uw}-4v$, где ***u, v, w*** – комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2. (Определить процедуры выполнения арифметических операций над комплексными числами).

Вариант 10

1. Даны действительные числа ***s, t***. Получить ***g(1.2,s)+ g(t,s)- g(2s-,st)***, где: $g(a,b)=\frac{a^2+b^2}{a^2+2ab+3b^2+4}$.

2. Даны действительные числа $x_1, y_1, x_2, y_2, .., x_{10}, y_{10}$. Найти периметр десятиугольника, вершины которого имеют соответственно координаты (x_1,y_1), (x_2,y_2),…, (x_{10},y_{10}). (Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами.)

3. Даны действительные числа $u_1, u_2, v_1, v_2, w_1, w_2$. Получить: $\frac{2uv}{2+3uw-v}-7v$, где ***u, v, w*** – комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2. (Определить процедуры выполнения арифметических операций над комплексными числами).

Вариант 11

1. По заданным 20-ти элементным целым массивам ***x*** и ***y*** вычислить:

$$u=\begin{cases}\sum_{i=1}^{20} x_i^2 & \text{при} \quad \sum_{i=1}^{15} x_i y_i > 0 \\ \sum_{i=10}^{20} y_i^2 & \text{иначе}\end{cases}.$$

2. Даны три действительные квадратные матрицы. Найти ту из них, норма которой наименьшая. В качестве нормы матрицы взять максимум абсолютных величин ее элементов.

3. Даны действительные числа $u_1\ u_2, v_1, v_2, w_1, w_2$. Получить: $\frac{2uv+w}{v-uw^2}+3$, где ***u, v, w –*** комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2. (Определить процедуры выполнения арифметических операций над комплексными числами).

Вариант 12

1. Даны действительные числа ***s, t***. Получить:

$$h(s,t)+\max\left(h^2(s+t,st),h^4(s-t,st/t)\right), \quad \text{где:} \qquad h(a,b)=\frac{\sqrt{a}}{1+b^2}+\frac{\sqrt{b}}{1+a^2}$$

2. Даны натуральное число **n**, действительные числа $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$. Найти площадь ***n***-угольника, вершины которого при некотором последовательном обходе имеют координаты (x_1, y_1), (x_2, y_2),…, (x_n, y_n). (Определить процедуру вычисления площади треугольника по координатам его вершин.)

3. Даны действительные числа u_1, u_2, v_1, v_2, w_1, w_2. Получить: $5u+\frac{3uw}{2+w^2-v^2}$, где ***u, v, w*** – комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2. (Определить процедуры выполнения арифметических операций над комплексными числами).

Вариант 13

1. Даны натуральные числа ***k, l, m***, действительные числа $x_1, \ldots, x_k, y_1, \ldots, y_l, z_1, \ldots, z_m$. Получить:

$$t=\begin{cases}(\min(y_1,\mathbf{K},y_l)+\min(z_1,\mathbf{K},z_m))/2 & \text{при } \min(x_1,\mathbf{K},x_k)\ge 0,\\ 1+(\max(x_1,\mathbf{K},x_k))^2 & \text{в противном случае.}\end{cases}$$

2. Даны действительные числа ***a***, ***b***, ***c***. Вычислить:

$$z=\left|e^{x_1+y_1}+e^{x_2-y_2}\right|,$$

где x_1, x_2 – корни уравнения $ax^2+bx+3=0$; y_1, y_2 – корни уравнения $y^2-2y+c=0$.

3. Даны действительные числа u_1, u_2, v_1, v_2, w_1, w_2. Получить $\frac{3u+2}{uw-2v}+5w$, где ***u, v, w*** – комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2 .

(Определить процедуры выполнения арифметических операций над комплексными числами).

Вариант 14

1. Даны действительные числа ***s, t.*** Получить:

$$f(t,-2s,1.7)+f(2,t,s-t)), \quad \text{где} \qquad f(a,b,c)=\frac{2a-b-\sin^2(c)}{5}.$$

2. Дано натуральное число ***n***. Выяснить, имеются ли среди чисел ***n***, ***n+1***,..., ***2n*** близнецы, то есть простые числа, разность между которыми равна трем. (Определить процедуру, позволяющую распознавать целые числа.)

3. Даны действительные числа u_1, u_2, v_1, v_2, w_1, w_2. Получить: $\frac{2u}{5uv}-4w$, где ***u, v, w*** – комплексные числа u_1+i*u_2, v_1+i*v_2, w_1+i*w_2.

(Определить процедуры выполнения арифметических операций над комплексными числами.)

Вариант 15

1. По заданным 20-ти элементным целым массивам ***x*** и ***y*** вычислить:

$$u=\begin{cases}\sum\limits_{i=10}^{20} y_i^2 & \text{при} \quad \prod\limits_{i=1}^{10} x_i y_i > 0 \\ \sum\limits_{i=1}^{20} x_i^2 & \text{иначе}\end{cases}.$$

2. Даны три квадратные действительные матрицы 15 порядка. Найти ту из них, у которой наибольший след (сумма диагональных элементов).

3. Даны действительные числа ***u₁, u₂, v₁, v₂, w₁, w₂***. Получить $\frac{2uv}{4+3u}-uvw$, где ***u, v, w*** – комплексные числа ***u₁+i*u₂, v₁+i*v₂, w₁+i*w₂***. (Определить процедуры выполнения арифметических операций над комплексными числами.)

Вариант 16

1. Даны действительные числа a, b, c. Получить $u=\min(a,b,c)$, $v=\min\left((ab)^2,\frac{a+b+c}{2}\right)$, $\min\left(u+v^2,2u+v\right)$.

2. Даны действительные числа ***a, d*** и логическое ***t***. Переменной ***t*** присвоить значение ***false***, если уравнения: $x^2-2x+a^2=0$ и $x^2+ax+b+1=0$.
не имеют вещественные корни и при этом оба корня первого уравнения лежат между корнями второго, и присвоить значение ***true*** во всех остальных случаях.

Тема: «Файлы»

Вариант 1

1. Сведения об ученике состоят из его имени и фамилии и названия класса (года обучения и буквы), в котором он учится. Дан файл ***f***, содержащий сведения об учениках школы и дополнительно отметки, полученные учениками в последней четверти. Выяснить, сколько учеников школы не имеют отметок ниже четверок.

2. Сведения об ученике состоят из его имени и фамилии и названия класса (года обучения и буквы), в котором он учится. Дан файл ***f***, содержащий сведения об учениках школы. Выяснить, в каких классах насчитывается более 25 учащихся.

Вариант 2

1. Дан символьный файл ***f***. В файле ***f*** не менее двух компонент. Определить, являются ли два первых символа файла цифрами. Если да, то установить, является ли число, образованное этими числами четным.

2. Дан символьный файл ***f*** .Записать в файл ***g*** с сохранением порядка следования те символы файла ***f***, вслед за которым в этом файле идет буква ***a***.

Вариант 3

1. Дан символьный файл ***f***. Записать в файл ***g*** компоненты файла ***f*** в обратном порядке.

2. Дан файл ***f***, содержащий сведения о кубиках: размер каждого кубика (длина ребра в сантиметрах), его цвет (красный, желтый, зеленый или синий) и материал (деревянный, металлический, картонный). Найти количество деревянных кубиков с ребром 3 см и количество металлических кубиков с ребром, большим 5 см.

Вариант 4

1. Дан файл ***f***, компоненты которого являются действительными числами. Найти сумму компонент файла.

2. Дан файл ***f***, содержащий сведения о книгах. Сведения о каждой из книг – это фамилия автора, название и год издания. Найти названия книг данного автора, изданных с 1960 г.

Вариант 5

1. Даны символьные файлы f_1 и f_2. Переписать с сохранением порядка следования компоненты файла f_1 в файл f_2 а файла f_2 в файл f_1 Использовать вспомогательный файл ***h***.

2. Даны квадратная матрица ***A*** порядка ***n***. Получить вектор ***Ab***, где ***b*** - вектор, элементы которого вычисляются по формуле:

$$b_i = \begin{cases} \dfrac{1}{i^2+2}, & \text{если } i \text{ четное} \\ \dfrac{1}{i} & \text{в противном случае} \end{cases}$$

Вариант 6

1. Даны символьные файлы ***f*** и ***g***. Записать в файл ***h*** сначала компоненты файла ***f***, затем – компоненты файла ***g*** с сохранением порядка.

2. Дан файл ***f***, содержащий различные даты. Каждая дата – это число, месяц и год. Найти все весенние даты.

Вариант 7

1. Дан файл ***f***, компоненты которого являются действительными числами. Найти разность первой и последней компонент файла.

2. Даны символьные файлы ***f*** и ***g***. Записать в файл ***h*** все начальные совпадающие компоненты файлов ***f*** и ***g***.

Вариант 8

1. Дан файл ***f***, компоненты которого являются действительными числами. Найти сумму квадратов компонент файла.

2. Дан символьный файл ***f***, содержащий сведения о сотрудниках учреждения, записанный по следующему образцу: **фамилия_имя_отчество**, **фамилия_имя_отчество, ...** Записать эти сведения в файле ***g***, используя образец: **фамилия_и.о., фамилия_и.о.**

Вариант 9

1. Дан файл ***f***, компоненты которого являются действительными числами. Найти наибольшее из значений компонент.

2. Дан символьный файл ***f***. Записать в файл ***g*** с сохранением порядка следования те символы файла ***f***, которым в этом файле предшествует буква ***a***.

Вариант 10

1. Дан символьный файл ***f.*** Получить копию файла в файле ***g***

2. Сведения об автомобиле состоят из его марки, номера и фамилии владельца. Дан файл ***f***, содержащий сведения о нескольких автомобилях. Найти количество автомобилей каждой марки.

Вариант 11

1. Дан файл ***f***, компоненты которого являются целыми числами. Найти количество удвоенных нечетных.

2. Дан символьный файл ***f***. (Группа символов, разделенных пробелами и не содержащая пробелов внутри себя, называется словом). Считая, что количество символов в слове не превосходит 20, определить, сколько в файле ***f*** имеется слов, состоящих из одного, двух, трех и т.д. символов.

Вариант 12

1. Дан файл ***f***, компоненты которого являются целыми числами. Найти количество четных чисел среди компонент.

2. Даны символьные файлы ***f*** и ***g***. Определить, совпадают ли компоненты файла ***f*** с компонентами файла ***g***. Если нет, то получить номер первой компоненты, в которой файлы ***f*** и ***g*** отличаются между собой. В случае, когда один файл имеет компонент (***n³0***) и повторяет начало другого (более длинного) файла, ответом должно быть число ***n + 1***.

Вариант 13

1. Дан файл ***f***, компоненты которого являются действительными числами. Найти произведение компонент файла ***f***.

2. Дан файл ***f***, содержащий сведения об экспортируемых товарах: указывается наименование товара, страна, импортирующая товар, и объем поставляемой партии в штуках. Найти страны, в которые экспортируется данный товар, и общий объем его экспорта.

Вариант 14

1. Вычислить бесконечную сумму с заданной точностью $e\left(e > \mathbf{0}\right)$. Считать, что требуемая точность достигнута, если вычислена сумма нескольких

первых слагаемых и очередное слагаемое по модулю меньше, чем e, – это и все последующие слагаемые можно не учитывать. Вычислить: $\sum_{i=1}^{\infty}\frac{(-1)^i}{(2i+1)i}$,

2. Сведения об ученике состоят из его имени и фамилии и названия класса (года обучения и буквы), в котором он учится. Дан файл ***f*** содержащий сведения об учениках школы. Выяснить имеются ли в школе однофамильцы.

Вариант 15

1. Дан файл ***f***, компоненты которого являются целыми числами. Записать в файле ***g*** все четные числа файла ***f***, а в файл ***h*** - все нечетные. Порядок следования чисел сохраняется.

2. Дан файл ***f*** содержащий сведения о кубиках: размер каждого кубика (длина ребра в сантиметрах), его цвет (красный, желтый, зеленый или синий) и материал (деревянный, металлический, картонный). Найти количество кубиков каждого из перечисленных цветов и их суммарный объем.

Вариант 16

1. Последовательность $x_1, x_2, \mathbf{K}$ образована по закону $x_i = \frac{i-0.1}{i^3+|tg2i|}(i=1,2,\mathbf{K})$. Дано действительное $\varepsilon > 0$. Записать в файл ***h*** члены последовательности $x_1, x_2, \mathbf{K}$, остановившись после первого члена, для которого выполнено $|x_i| < \varepsilon$.

2. Сведения об ученике состоят из его имени и фамилии и названия класса (года обучения и буквы), в котором он учится. Дан файл ***f*** содержащий сведения об учениках школы. Выяснить имеются ли в школе однофамильцы.

Тема: «Поиск и сортировка»

Вариант 1

1. Символьная строка содержит последовательность слов, разделенных пробелами. Найти все слова, которые читаются слева направо так же, как и справа налево.

2. Упорядочить заданный массив ***A(n)*** по неубыванию, многократно переставляя каждые два соседних элемента, нарушающие порядок. Процесс завершается по достижении упорядоченности массива.

Вариант 2

1. Написать программу, которая находит и выводит на печать все четырехзначные числа ***a***, ***b***, ***c***, ***d***, для которых выполняется условие:

1) ***a***, ***b***, ***c***, ***d*** – разные цифры

2) ***ab - cd = a + b + c + d***.

2. Упорядочить строки матрицы ***A(m, n)*** по убыванию.

Вариант 3

1. На интервале **(1000, 9999)** найти простые числа, каждое из которых обладает тем свойством, что сумма 1-ой и 2-ой цифры записи этого числа равна сумме 3-ей и 4-ой цифр.

2. Провести построчное слияние 2-х матриц ***A(m, n)*** и ***B(k, n)***, упорядоченных по неубыванию элементов 1-го столбца.

Вариант 4

1. На избирательном участке в списке из 100 избирателей указываются фамилия и название улицы, на которой проживает избиратель. Определить, сколько избирателей живет на улице Богуна.

2. Из 2-ух упорядоченных по невозрастанию массивов ***A(m)*** и ***B(n)***, получить путем слияния упорядоченный по неубыванию массив ***C***, удаляемые элементы собрать в массиве ***D***. Подсчитать количество элементов в массивах ***C*** и ***D***.

Вариант 5

1. Написать программу, которая находит и выводит на печать все четырехзначные числа ***a***, ***b***, ***c***, ***d***, для которых выполняется условие:

1) ***a***, ***b***, ***c***, ***d*** — разные цифры

2) $ab - cd = a + b + c + d$.

2. Даны 2 упорядоченных по возрастанию массива ***A(n)*** и ***B(m)***. Получить из них путем слияния упорядоченный по возрастанию массив ***C***; совпадающие элементы, вставлять единожды. Подсчитать количество элементов в массиве ***C***.

Вариант 6

1. Дана целочисленная прямоугольная таблица **A[1:100, 1:150]**. Найти наибольшее из чисел, встречающихся в этой таблице.

2. В неупорядоченном массиве К(***n***) есть совпадающие элементы. Из каждой группы одинаковых элементов оставить только один, удалив остальные и поджав его к началу.

Вариант 7

1. Дана целочисленная таблица A[1:100]. Проверьте, есть ли в ней отрицательные элементы. Если есть, найдите наибольшее i, при котором $A[i] < 0$.

2. Упорядочить по неубыванию массив ***A(n)*** разбивая его на группы по 1, 2, 4, 8, ... элементов и проводя попарное слияние соседних групп (число ***n*** не обязательно равно степени 2). Разрешается выделить дополнительный рабочий массив ***B(n)*** и проводить слияние поочередно из ***A*** в ***B***, затем ***B*** в ***A*** и т.д.

Вариант 8

1. Дано натуральное число ***n***. Сколько различных цифр встречается в его десятичной записи.

2. Упорядочить массив ***F(m)*** по неубыванию, меняя местами каждый элемент с минимальным среди следующих за ним, то есть используя метод:

$$P_i = \min_j \{P_j, j = i, i+1, \ldots, m\}, \qquad i = 1, \ldots, m-1$$

Вариант 9

1. Дана вещественная таблица A[1], A[2],..., A[1000]. Определить максимальное количество подряд идущих положительных элементов последовательности, не прерываемых ни нулями, ни отрицательными элементами.

2. Упорядоченный строки матрицы ***B(m, n)*** по неубыванию элементов ее каждого столбца.

Вариант 10

1. Дана целочисленная таблица **A[1:100]**. Подсчитать наибольшее число идущих в ней подряд одинаковых элементов.

2. Упорядоченный по невозрастанию массив ***B(m)*** преобразовать в упорядоченный по возрастанию, оставив по одному в каждой группе совпадающих элементов.

Вариант 11

1. Дана целочисленная таблица **A[1:100]**. Проверьте, есть ли в ней элементы, равные нулю. Если есть, найдите номер первого из них, т.е. наименьшее ***i***, при котором **A[i] = 0**.

2. Упорядочить массив **R(1)** по не возрастанию, используя следующий подход: для ***i = 2, 3, ... ,1*** каждый элемент r_i вставлять в нужное место среди упорядоченных ранее элементов $r_1, r_2, \ldots, r_{i-1}$ за счет удаления r_i.

Вариант 12

1. При проведении физического эксперимента по фиксации траектории движения частиц с помощью ЭВМ детекторы сгруппированы следующим образом:

ДД … Д
ДД … Д
… … …
ДД … ДД

Перекрытая детекторами область отображается в памяти ЭВМ массивом: ***M[1:N,1:K]***, элементы которого представляют собой цифровую фотографию исследуемой области. При фиксации элементарной частицы детекторами в позиции ***(I,J)***, соответствующий элемент матрицы ***M*** принимает значение ***1***, в противном случае ***0***. Определите содержит данная цифровая фотография информацию, которая может быть интерпретирована как след прямолинейной траектории, начинающейся и заканчивающейся за пределами фотографии.

2. Упорядочить по неубыванию каждую строку матрицы ***A(m, n)***, а после этого перестановкой строк упорядочить всю матрицу по не убыванию элементов 1-го столбца.

Вариант 13

1. Дана целочисленная таблица **A[1:100, 1:100]**. Найти наименьшее из чисел, встречающихся в этой таблице.

2. Даны 2 упорядоченных по возрастанию массива ***A(n)*** и ***B(m)***. Получить из них путем слияния упорядоченный по убыванию массив ***C***; совпадающие элементы, вставлять единожды. Подсчитать количество элементов в массиве ***C***.

Вариант 14

1. Символьная строка содержит последовательность слов, разделенных пробелами. Найти все слова, которые читаются слева направо так же, как и справа налево.

2. Упорядочить по неубыванию массив ***A(n)*** разбивая его на группы по 1, 2, 4, 8, ... элементов и проводя попарное слияние соседних групп (число ***n*** не обязательно равно степени. Разрешается выделить дополнительный рабочий массив ***B(n)*** и проводить слияние поочередно из ***A*** в ***B***, затем ***B*** в ***A*** и т.д.

Вариант 15

1. Дана целочисленная таблица A[1:100]. Проверьте, есть ли в ней отрицательные элементы. Если есть, найдите наибольшее и наименьшее i, при котором A [i] ≥ 0.

2. Упорядочить заданный массив ***A(n)*** по неубыванию, многократно переставляя каждые два соседних элемента, нарушающие порядок. Процесс завершается по достижении упорядоченности массива.

Вариант 16

1. Дана целочисленная таблица **A[1:100]**. Подсчитать наибольшее число идущих в ней подряд одинаковых элементов больших 0.

2. Упорядоченный по возрастанию массив ***B(m)*** преобразовать в упорядоченный невозрастанию, оставив по одному в каждой группе совпадающих элементов.

Тема: «Работа с динамическими данными»

Задание № 1. ***Формирование списка с одновременным упорядочением его элементов***

Варианты заданий

Составить список учебников для N-го курса, указав название, фамилию автора, год издания, цену, тираж и упорядочить его по заданному признаку	**N курса**	**Название поля**
	I **II** **III** **IV** **V**	Фамилия автора Название Год издания Цена тираж
Составить список кабинетов техникума для M-го этажа, указав название кабинета, количество посадочных мест, и упорядочить его по заданному признаку	**M этажа**	**Название поля**
	I **II** **III**	Название кабинета Номер комнаты Количество мест

Задание № 2. ***Исключение элементов из списка***

Варианты заданий

Исключить из списка элементы, относящиеся к учащимся у которых:

1. Средний балл меньше среднего балла группы.
2. Средний балл меньше 4,5.
3. Средний балл больше 4.
4. Все оценки 5.
5. Одна оценка 4, а остальные – 5.
6. Оценка, полученная на первом экзамене – 2.
7. Оценка, полученная на втором экзамене – 5.
8. Нет удовлетворительных и неудовлетворительных оценок.
9. Больше одной оценки 2.
10. Одна оценка 3, а остальные – 4 и 5.

Распечатать оставшийся список.

Задание № 3. ***Выполнение операций над списковыми структурами***

Варианты заданий

1. Удалить первые два символа строки.
2. Удалить последние три символа строки.
3. Удалить все буквы **К**.
4. Удвоить все символы *****.
5. Добавить в конец строки слово **END**.
6. Поменять местами первый и последний символы строки.
7. Поменять местами первый и второй символы строки.
8. Поменять местами последний и предпоследний символы строки.
9. Подсчитать в строке число букв **А** и **В**, и если букв **А** больше, чем букв **В**, то

удалить в стоке все символы **B**.
10. Подсчитать в строке число символов, и если число четное, то удалить символ стоящий посередине строки..

Задание № 4. ***Работа со связанными списками***

На языке PASCAL реализовать следующие процедуры для работы со связанными списками:

- распределение элемента в памяти;
- освобождение памяти занятой элементом;
- получение адресата последнего элемента списка;
- добавление нового элемента в конец списка;
- получение элемента из списка по индексу;
- вставка нового элемента в заданную позицию;
- получение индекса элемента в списке;
- удаление элемента по его индексу;
- получение количества элементов в списке;
- освобождение памяти занятой списком.

С использованием этих процедур написать программу для сложения и умножения двух многочленов n-ой степени от двух переменных в символьном виде. Реализовать процедуру ввода коэффициентов и степени многочленов с клавиатуры или из файла и процедуру вычисления значения многочлена от заданных переменных.

СПИСОК ЛИТЕРАТУРЫ

1. Фаронов В.В. Программирование на персональных ЭВМ в среде Турбо Паскаль. М.: МГУ, 2004.
2. Пильщиков В.Н. Сборник упражнений по языку Паскаль. М.: Наука, 1998.
3. Васюкова Н.Д., Тюляева В.В. Практикум по основам программирования на языке Паскаль. М.: Высшая школа, 1997.
4. Т. Грызлова, В. Грызлов. Турбо Паскаль 7.0. Учебный курс. М.: ДМК Пресс, 2005.
5. Поляков Д.Б., Круглов И.Ю. Программирование в среде Турбо Паскаль. М., 1999.
6. Перминов О.Н. Программирование на языке Паскаль. М., 2001.

СОДЕРЖАНИЕ

Редактор В.Л. Родичева

Подписано в печать 17.03.2008. Формат 60х84 1/16. Бумага писчая
.Усл. печ. л. 3,26. Уч.-изд. л. 3,61 Тираж 100 экз. Заказ

ГОУВПО «Ивановский государственный химико-технологический университет»
Отпечатано на полиграфическом оборудовании кафедры экономики и финансов
ГОУВПО «ИГХТУ»
15300, г. Иваново, пр. Ф. Энгельса, 7