ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ В ПРИМЕРАХ И ЗАДАЧАХ

ОГЛАВЛЕНИЕ

Глава 1. ОСНОВНЫЕ ТИПЫ АЛГОРИТМОВ

1.1. Понятие алгоритма

Решить задачу — это значит получить результат, отвечающий целям данной задачи. Процесс решения представляет собой совокупность вполне определенных действий над исходными данными. В большинстве случаев эта совокупность действий может быть задана в виде инструкции настолько подробно, что ее исполнение становится механическим процессом. Такая инструкция называется *алгоритмом.*

Понятие алгоритма имеет строгое математическое определение, но мы остановимся на интуитивно-содержательном определении: алгоритмом называется последовательность точных и понятных действий (предписаний, правил, инструкций, команд) над данными, приводящая к получению результата решения любой конкретной задачи из некоторого класса однотипных задач.

Основные свойства любого алгоритма: *детерминированность,* означающая, что применение алгоритма к одним и тем же исходным данным должно приводить к одному и тому же результату; *массовость,* позволяющая получать результат при различных исходных данных; *результативность,* обеспечивающая получение результата через конечное число шагов.

Существует несколько способов записи алгоритмов, отличающихся друг от друга наглядностью, компактностью, степенью формализации и другими показателями. Наибольшее распространение получили способы: *словесный , в виде блок-схем, в виде программ для ЭВМ.*

Независимо от того, как представлен алгоритм (в виде схемы, текста или программы), его *структура* должна быть достаточно жесткой и включать определенные элементы.

Из теории известно, что любой алгоритм может быть реализован в виде комбинации трех базовых алгоритмических конструкций: *следование, ветвление, повторение (цикл).*

Конструкция следование образуется из последовательности команд, выполняемых друг за другом в естественной последовательности.

Ветвление осуществляет выбор одного из двух или более возможных действий в зависимости от условия.

Повторение (цикл) обеспечивает многократное выполнение одних и тех же действий для различных значений входящих в них данных.

1.2. Этапы разработки алгоритма

Разработка любого алгоритма состоит из нескольких тесно взаимосвязанных этапов. На каждом из них решаются свои специфические вопросы, определяющие в конечном счете общий результат алгоритмизации решаемой задачи.

Этап *анализа постановки задачи* — это основа решения задачи. Именно на этом этапе определяются составляющие, необходимые для разработки алгоритма.

Главная цель анализа — это понять задачу. Если совершена ошибка на этом этапе, т.е. задача неверно понята, то вся последующая работа становится бессмысленной.

Что значит понять задачу? Во-первых, необходимо в постановке задачи выделить и осмыслить все исходные данные и требуемые результаты. Во-вторых, необходимо сформулировать постановку задачи в виде одного короткого конкретного предложения. Чтобы предложение получилось коротким, необходимо в исходной постановке задачи «выбросить» все уточняющие детали.

“Сборка” алгоритма может происходить двумя путями. Во-первых, базовые конструкции могут соединяться в последовательность, образуя конструкцию следования. Это возможно, так как каждая базовая конструкция имеет один вход и один выход. Во-вторых, одна базовая конструкция может вкладываться в другую конструкцию, образуя “вложенные” конструкции. Это также возможно, так как внутри составных команд могут быть снова составные конструкции.

Таким образом, при построении алгоритм может развиваться как “вширь”, так и “вглубь” включением одних конструкций в другие. Такое конструирование обычно и происходит на практике. Алгоритм строится в несколько шагов: сначала он формулируется в самых общих чертах, а затем уточняется путем детализации более крупных через более мелкие. На каждом шаге алгоритм расширяется по мере добавления и уточнения деталей. Уровень команд (инструкций) постепенно понижается. Весь процесс заканчивается тогда, когда будет достигнут уровень операторов алгоритмического языка.

Такой метод разработки алгоритмов называется *методом пошаговой детализации* или проектированием *сверху—вниз* (от системы команд высокого уровня к системе команд низкого уровня).

Указанный метод разработки алгоритма позволяет получить алгоритм с ясно выраженной структурой, что облегчает понимание и доказательство его правильности.

Рассмотрим примеры реализации основных типов алгоритмов: алгоритмы линейной, разветвляющейся и циклической структуры.

1.3. Реализация линейных алгоритмов

Линейными называют алгоритмы, в которых последовательность выполнения инструкций (операторов) совпадает с порядком их записи и не зависит от конкретных значений исходных данных.

К линейным относятся алгоритмы вычислений по определенным формулам. Раздел операторов таких алгоритмов (и соответствующих программ) состоит из трех

частей: ввод исходных данных, вычисления по заданным формулам, вывод требуемых результатов.

Для реализации подобных программ в языке Турбо Паскаль используются оператор присваивания, составной оператор, оператор вызова процедур ввода данных (READ или READLN) и вывода данных (WRITE или WRITELN).

Оператор присваивания

Оператор присваивания — один из основных операторов Турбо Паскаль. Он предназначен для изменения значения переменной или, другими словами, для присваивания переменной нового значения. Общая форма оператора:

<переменная> := <*выражение*>

Здесь и далее угловыми скобками выделены объекты языка, участвующие в определении конструкции.

Выражение в программировании служит для задания действий, которые необходимы для определения нового значения. Например, a + b –1, A*sin(T), (x > 0) And (x < 1), ‘Решение’ + ' уравнения’ и т.п.

При выполнении оператора вычисляется указанное справа «выражение», полученное значение присваивается указанной «переменной», старое значение переменной теряется.

Содержание записи операторов присваивания:

A := 27.4 — вещественной переменной A присваивается значение 27.4;

X :=A*sin(T)—вещественной переменной X присваивается значение выражения A*sin(T) при текущих значениях переменных A и T;

ARR (5) := 0 — пятому элементу массива ARR присваивается значение 0;

F := 'Пример'—символьной переменной F присваивается символьное значение «Пример».

В операторе присваивания слева и справа от символа := может фигурировать имя одной и той же переменной, например

I := I + 1 — значение целой переменной I увеличивается на 1.

Ввод и вывод данных

Для ввода с клавиатуры используются операторы:

READ(b_1, b_2, ..., b_n);
READLN(b_1, b_2, ..., b_n);
READLN;

где b_1, b_2, ..., b_n — имена переменных, значения которых вводятся.

Оператор READ(b_1, b_2, ..., b_n) осуществляет ввод данных с клавиатуры целого, вещественного, символьного и строкового типов. Типы вводимых данных должны соответствовать типам переменных в списке оператора ввода.

Оператор READLN(b_1, b_2, ..., b_n) осуществляет ввод данных с клавиатуры и после выбора значения последней переменной обеспечивает переход к началу новой строки. При вводе значений целого и действительного типов операторы READ и READLN игнорируют пробелы между значениями.

Оператор READLN обеспечивает пропуск одной строки и переход к началу новой строки.

Для вывода информации используются операторы:

WRITE (b_1, b_2, ..., b_n);

WRITELN(b_1, b_2, ..., b_n);

WRITELN;

где b_1, b_2, ..., b_n — *выражения,* значения которых выводятся. Обратите внимание на то, что выводить можно только значения целого, вещественного, символьного, логического и строкового типов.

Оператор WRITE(b_1, b_2, ..., b_n) выполняет вывод значений соответствующих выражений на экран, размещая выводимые значения в одной строке.

Оператор WRITELN(b_1, b_2, ..., b_n) выполняет вывод значений выражений на экран и после вывода последнего значения осуществляет переход к началу новой строки.

Оператор WRITELN обеспечивает переход к началу новой строки.

Значения выражений в списке вывода могут принадлежать к целому, вещественному, символьному или логическому типам. Значения величин вещественного типа выводятся в нормализованном виде (мантисса числа с порядком), а целого типа — в обычной форме. Операторы вывода допускают использование указания о ширине поля, отводимого под значение.

Общий вид записи операторов для вывода значений целого типа:

WRITE (b:m);

WRITELN(b:m);

а для вывода действительного типа:

WRITE(b:m:n);

WRITELN(b:m:n);

где b — элемент списка вывода; m — поле, отводимое под значение и задаваемое константой или выражением целого типа; n — часть поля, отводимого под дробную часть числа. Например, оператор WRITE (KOL:8,NOM:4) выделяет на строке под значения переменных KOL и NOM соответственно восемь и четыре позиции, а оператор WRITELH(SUM: 10: 5) выделяет под значение SUM десять позиций, из которых пять позиции отводится под дробную часть числа.

Примеры реализации линейного алгоритма

Пример 1.1. Разработать программу вычисления значения функции для произвольных значений ее аргументов:

$$Y=\sqrt{\frac{(1+\lg^2|x|)(x^2+a^2+56.78\cdot10^{-8}b^2)}{(1+e^{-ax})(1+e^{-\sin b})}}.$$

Результаты вычислений вывести с поясняющим текстом.

Решение.

Анализ постановки задачи

Требуется выполнить вычисления по заданной формуле, используя линейный алгоритм решения задачи. Прежде, чем вычислять функцию, необходимо определить значения аргументов *a,b,x.* После вычислений значения выводятся на экран. Другими словами, в алгоритме можно выделить три этапа: ввод исходных данных, вычисления, вывод результатов.

Имена объектов задачи (а следовательно, и объектов программы) фактически заданы: аргументы—A, B, X, функция—Y.

Все объекты — переменные вещественного типа.

Ввод исходных данных в задаче — ввод трех переменных. Его можно реализовать одним оператором, например в виде

WRITE(‘Введите аргументы A, B, X: ’);

READLN(A,B,X);

Детализируем этап вывода результатов. Форма вывода в постановке задачи не определена, поэтому принимаем макет печати результатов в виде:

Результаты решения задачи

Аргументы функции: A= **.** B= **.**, X= **.**

Значение функции: Y = *****.**

Здесь и далее символ "*" обозначает любую цифру, а символ "." – десятичную точку, разделяющую целую и дробную части.

Макет вывода результатов определен, а его реализация — последовательность операторов WRITELN - не представляет каких-либо затруднений.

Детализируем этап вычислений. Поскольку задана одна формула, то вычисления можно реализовать одним оператором присваивания. Однако при этом рекомендуется избегать сложных выражений. С этой целью можно использовать разные приемы, например, дополнительные обозначения: $t=1+\lg^2|x|$, $z=x^2+a^2+56.78 \times 10^{-8} b^2$, $v=1+e^{-ax}$, $w=1+e^{-\sin b}$.

В результате для вычисления функции получим выражение:

$$Y = SQR-(t -z/v/w).$$

Этот прием плох тем, что в программе появляются вспомогательные объекты, загромождающие программу.

Более эффективен другой прием, основывающийся на возможности использовать в операторе присваивания слева и справа от символа присваивания имена одной и той же переменной. В соответствии с этим все вычисления можно представить совокупностью следующих операторов присваивания:

```
Y:= LN(ABS(X))/LN(10);
Y:=1+Y*Y;
Y:=Y*(X*X+A*A+56.78E-8*B*B);
Y:=Y/(1+EXP(-A*X);
Y:=Y/(1.+EXP(-SIN(B));
Y:=SQR(Y);
```

Поскольку все этапы решения максимально детализированы, их можно объединить в программу.

Текст программы

```
PROGRAM Prim11;
{Программа вычисления функции}
{A,B,X —Аргументы}
{Y—функции}
VAR
   A,B,X,Y : REAL;
```

```
BEGIN
    WRITE(‘Введите аргументы A, B, X: ’);
    READLN(A,B,X);
  {Этап вычислений}
   Y := LN(ABS(X))/LN(10);
   Y :=1+Y*Y;
   Y:=Y*(X*X+A*A+56.78E-8*B*B);
   Y:=Y/(1+EXP(-A*X));
   Y:=Y/(1+EXP(-SIN(B)));
   Y:=SQR(Y);
  {Этап вывода результатов}
   WRITELN(‘Результаты решения задачи');
   WRITELN (‘Аргументы функции:');
   WRITELN (‘A= ‘,A:5:2,’  B = ‘,B:5:2,’  X= ‘,X:5:2);
   WRITELN (‘Значение функции — Y= ‘,Y:8:2);
END.
```

В приведенном тексте программы содержатся поясняющие комментарии. *Комментарий* в Турбо Паскале – это произвольная последовательность любых символов, обрамленная фигурными скобками. Комментарии не производят никаких алгоритмических действий.

Пример 1.2. Вычислить высоты треугольника со сторонами a, b, c, используя формулы:

$$h_a = \frac{2}{a}\sqrt{p(p-a)(p-b)(p-c)},$$

$$h_b = \frac{2}{b}\sqrt{p(p-a)(p-b)(p-c)},$$

$$h_c = \frac{2}{c}\sqrt{p(p-a)(p-b)(p-c)},$$

где $p = (a+b+c)/2$.

Решение.

При решении данной задачи для исключения повторений следует вычислять не по приведенным выше формулам непосредственно, а используя промежуточную переменную

$$t = 2\sqrt{p(p-a)(p-b)(p-c)},$$

тогда $h_a = t/a$, $h_b = t/b$, $h_c = t/c$.

Текст программы

```
PROGRAM PRIM12;
VAR
  a, b , c, t, p: REAL;
BEGIN
  WRITE(‘Введите длины сторон a, b, c’);
  READLN(a, b, c);
  p:= (a+b+c)/2;
  t:= 2*SQRT(p*(p-a)*(p-b)*(p-c));
 WRITELN(‘Высота ha=’,t/a);
```

```
  WRITELN('Высота hb=',t/b);
  WRITELN('Высота hc=',t/c);
END.
```

Контрольные вопросы

1. Что называется алгоритмом?
2. Для чего необходимы выражения?
3. Какова последовательность действий при выполнении оператора присваивания?
4. Какие имеются средства в языке Турбо Паскаль для управления размещением данных на строке?
5. Как организовать вывод значений, сопровождая выводимое значение наименованием переменной?
6. Как организовать пропуск одной, двух строк при выводе?

Задачи

Эти задачи предназначены для приобретения навыков реализации линейных алгоритмов. Вычислить значения заданных функций.

1
$$S = \ln\left(x + \sqrt{x^2 + 1}\right)$$
$$Z = ae^{-ax}\sin(bx)$$

2
$$Y = \frac{2\cos(x - \frac{\pi}{6})}{0{,}5 + \sin^2 y}$$
$$Z = 1 + \frac{b^2}{3 + \frac{b^2}{5}}$$

3
$$Y = ax^{\frac{3}{2}}$$
$$Z = \frac{6a}{\sqrt{x}(4 + 9a^2 x)}$$

4
$$Y = \sin^3(x^2 + a^2) - \sqrt{\frac{x}{b}}$$
$$Z = \frac{x^2}{a} + \cos(x + b)^3$$

5
$$Y = x^2(x + 1) - \sin^2(x + a)$$
$$Z = \sqrt{\frac{xb}{a}} + \cos^2(x + b)^3$$

6
$$Z = \frac{-t\sqrt{t^2 + 8a^2}}{4a}$$
$$W = (x^2 + y^2 - ax)^2$$

7
$$Y = at^3$$
$$Z = \frac{1}{27a^2}\left((4 + 9a^2 x)^{\frac{3}{2}} - 8\right)$$

8
$$Y = e^{-bt}\sin(at + b) + \sqrt{|bt + a|}$$
$$S = b\sin(at^2\cos 2t) - 1$$

16
$$Z = c^2\cos x + \sqrt{c^4\cos^2 2x}$$
$$W = (x^2 + y^2)^2 - 2c(x^2 - y^2)$$

17
$$Z = \frac{4a(x + a)}{2a + x} + \sin(\frac{x}{2a})$$
$$W = \pi a^2\frac{3x + 2a}{x}$$

9 $S = x^3 \operatorname{tg}(x+b)^2 + a$

$Z = \dfrac{bx^2 - a}{e^{bx} - 1}$

10 $Y = \dfrac{at^3}{1+t^2}$

$Z = \dfrac{a \sin^2 x}{\cos x}$

11 $Y = at(t^2 - 1)^{\frac{3}{2}}$

$Z = (e^{tx} + \pi)t^2 + 1$

12 $Y = at\dfrac{t^2 - 1}{t^2 + 1}$

$Z = 2a^2 - 0{,}5\pi a^2$

13 $Z = (x - a)^2(x^2 + y^2) - \ln x$

$W = \dfrac{\sqrt{x^2 - a^2}}{2}$

14 $Z = \dfrac{\sin x}{\sqrt{1 + t^2 \sin^2 x}}$

$W = e^{-tx}\sqrt{x+1} + e^{tx}\sqrt{x+1.5}$

15 $y = \dfrac{a^2 x - e^{-x}\cos ax}{ax - e^{-x}\sin ax}$

$Z = \ln(a+x) + e^{2x}\ln(a^2 + x^2)$

18 $Y = (a+x)^3 \cos x + \ln\cos x$

$Z = x^{\frac{2}{3}} + e^{(a-x)^2}$

19 $Y = \dfrac{ae^{\frac{x}{a}} + e^{-\frac{x}{a}}}{2}$

$Z = \cos\left(a + \dfrac{x^2}{2a}\right)^{\frac{3}{2}}$

20 $Y = \dfrac{ax\sqrt{bx} + ax^2}{(b-x)^2}$

$Z = (a+b) + ba\sin^2\left(\dfrac{b+a}{x}\right)$

21 $Y = e^{\cos x + \ln^2(x+a)}$

$Z = \sqrt{-bx^2 + abx - c}$

22 $Z = \dfrac{x^2}{a^2} + \dfrac{y^2}{b^2} + \dfrac{xy}{ab}$

$W = -\pi + \operatorname{arctg}\left(\dfrac{x-y}{ab+xy}\right)$

23 $e = \operatorname{arctg}(x^t + a) - ax$

$Z = e^{-ax} + \ln^2(tx - a)^2$

24 $y = \left|ax^2 - x\sin^3 a\right|$

$Z = \sqrt{\operatorname{arctg}\dfrac{a^2 + x^2}{ax}}$

1.4. Реализация разветвляющихся алгоритмов

Для описания даже простых вычислительных процессов оказывается недостаточно лишь одних линейных алгоритмов. Например, алгоритм вычисления функции, заданной формулой

$$Y = \begin{cases} \ln x, \text{ если } x \le 1, \\ 1 - x, \text{ если } x < 1 \end{cases}$$

уже не является линейным, т. к. в нем должна быть заложена операция выбора одной из формул в зависимости от заданного значения аргумента X. Такого рода алгоритмы *называются разветвляющимися, или ветвящимися.*

В разветвляющихся алгоритмах принцип линейного автоматического перехода от предписания к предписанию (от оператора к оператору) в порядке их естественной записи не является всеобщим, так как возникает потребность выполнения «произвольного» предписания, определяемого некоторыми условиями.

Итак, *разветвляющимися* называются алгоритмы, в которых последовательность выполнения некоторых предписаний определяется выполнением (или невыполнением) определенных условий.

Для реализации разветвлений в Турбо Паскале используются условные *операторы и оператор выбора.* Кроме них может оказаться необходимым *оператор*

перехода.

Условный оператор

Одна из форм условного оператора имеет вид:

IF <*условие*> THEN <*оператор1*> ELSE <*оператор 2*>

Здесь IF *(если),* THEN *(то, тогда),* ELSE *(иначе)* —ключевые слова языка; <*условие*> — логическое выражение; <*оператор1 и оператор2*> — один исполняемый оператор.

Смысл условного оператора: если условие при текущих значениях входящих в него переменных истинно, то выполняется <*оператор1*> (<*оператор2*> не выполняется), если же условие ложно, то выполняется <*оператор2*> (<*оператор1*> не выполняется). После выполнения <*оператора1*> или <*оператора2*> управление передается очередной строке программы, если в <*операторах1 и 2*> не предусмотрено программное изменение последовательности выполнения строк.

Примеры записи условного оператора:

- для нахождения большего из двух значений:

```
IF A > B THEN X:=A ELSE X:=B;
```

- для нахождения большего из двух значений с фиксацией его имени:

```
IF A > B THEN
BEGIN
X:=A;
F:=’A’
            END
ELSE
BEGIN
X:=B;
F:=’B’
            END;
```

Здесь F переменная символьного типа.

В конструкции условного оператора ветвь «иначе» не обязательна, она может отсутствовать. В этом случае имеет место сокращенная форма условного оператора:

IF <*условие*> THEN <*оператор*>

Смысл оператора: если <*условие*> истинно, то выполняется указанный оператор. Если условие ложно, то оператор не выполняется, управление просто переходит к очередной строке программы.

Очевидно, что один условный оператор в полной форме можно заменить двумя в сокращенной форме; например для нахождения большего из двух значений с фиксацией его имени

```
IF A > B THEN
BEGIN
  X:=A;
```

```
  F:='A'
END;
IF A <= B THEN
BEGIN
  X:=B;
  F:='B'
END;
```

В условном операторе после ключевых слов THEN и ELSE можно использовать другой условный оператор. Получающиеся конструкции называются *вложенными условными операторами.*

Примеры вложенных условных операторов:

```
IF X > 0 THEN
IF X <= 1 THEN Y:=1 +X
          ELSE Y:=X*X;
```

Здесь возможна неоднозначность исполнения программы с ветвью, следующей за ELSE. Чтобы ее исключить принимается, что ELSE «закрывает» последний «незакрытый» THEN.

Примеры использования условного оператора

Пример 1.3. Вычислить функцию:

$$y = \begin{cases} 1 - e^{-x} & \text{при } x < 0, \\ 1 + p & \text{при } x = 0, \\ 1/(x+p) & \text{при } 0 < x < 1, \\ 1 + p\sin x & \text{при } x \geq 1. \end{cases}$$

Решение.

Фрагмент программы

```
IF X<0 THEN Y:= 1 + EXP(-X);
IF X=0 THEN Y:=1+P;
IF (X > 0) AND (X < 1) THEN Y:=1/(X+P);
IF X>=1 THEN Y:=1+P*SIN(X);
```

Пример 1.4. Вычислить значения Y и Z**:**

$$Y = \begin{cases} ae^{-b} & \text{при } ab < 1, \\ b\sin^2 a^3 & \text{при } ab \geq 1, \end{cases}$$

$$Z = \begin{cases} a + \cos b & \text{при } ab < 1, \\ 1/(|b| + e^{-a}) & \text{при } ab \geq 1. \end{cases}$$

Решение.

Фрагмент программы

```
IF A*B < 1 THEN
BEGIN
   Y:=A*EXP(-B);
   Z=A+COS(B)
END
ELSE
BEGIN
  Y=B*SQR(SIN(A*A*A));
  Z=1/(ABS(B)+EXP(-A))
END;
```

Пример 1.5. Найти наибольшее из трех вещественных чисел (A, B, C) с фиксацией его имени.

Решение.

Фрагмент программы

```
IF A > B THEN
 BEGIN
  X:=A;
  F:='A'
 END
ELSE
 BEGIN
  X:=B;
  F:='B'
 END;
IF C > X THEN
 BEGIN
  X:=C;
  F:='C'
 END;
```

Оператор перехода

Оператор перехода (безусловного перехода) используется для изменения естественной последовательности выполнения строк программы. Общая форма оператора:

GOTO <*метка*>

Смысл оператора: управление программой передается оператору с указанной меткой. Например:

GOTO Metka;

GOTO 200;

Метка назначается пользователем и представляет собой правильный идентификатор или целое число без знака, содержащее не более четырех цифр. Используемые в программе метки должны быть описаны в разделе описания меток.

Операторы перехода при их бессистемном использовании затрудняют чтение программы, делают ее запутанной и трудной для отладки, поэтому говорят, что

квалификация программиста обратно пропорциональна количеству операторов перехода в его программах.

С другой стороны, оператор перехода - часто неизбежный оператор при программировании разветвляющихся участков программ.

Учитывая эти два противоречивых высказывания, можно сделать вывод: следует рационально использовать оператор перехода, исключив его использование, там, где можно использовать другие средства.

Оператор выбора

Оператор выбора (CASE) обеспечивает выполнение одного оператора (простого или сложного) из нескольких возможных. Выбор оператора определяется значением выражения (селектора), которое располагается между ключевыми словами CASE и OF. Значение выражения должно совпадать с константами, стоящими перед операторами. Выражение может принадлежать любому простому типу, кроме вещественного.

Выбор оператора определяется совпадением значения селектора и константы, стоящей перед оператором.

Пример 1.6. Написать программу для вывода дней недели.

```
PROGRAM Prim16;
  VAR
    NUMBER : BYTE;
 BEGIN
   READ (NUMBER);
   CASE NUMBER OF
 1: WRITELN('ПОНЕДЕЛЬНИК');
 2: WRITELN('ВТОРНИК');
 3: WRITELN('СРЕДА');
 4: WRITELN('ЧЕТВЕРГ');
 5: WRITELN('ПЯТНИЦА');
 6: WRITELN('СУББОТА');
 7: WRITELN('ВОСКРЕСЕНЬЕ');
    END
 END.
```

Примеры реализации разветвляющегося алгоритма

Пример 1.7. Разработать программу для вычисления функции

$$y = \frac{1}{1 - a^2 + b^2}$$

для произвольных значений аргументов.

Решение.

Анализ постановки задачи

На первый взгляд кажется, что решение задачи сводится к линейному

алгоритму, т.е. вычислению функции, заданной формулой. Однако это мнение ошибочное, так как возможны значения аргументов *a* и *b,* обращающие знаменатель дроби в нуль. Следовательно, в таких случаях задача не имеет решения. Поэтому в алгоритме (и программе) необходимо предусмотреть две ветви:

- первая, когда знаменатель обращается в нуль и об этом необходимо вывести сообщение;

- вторая, если знаменатель отличен от нуля, то вычисляется функция и выводятся результаты вычисления.

Если ограничиться выводом простейших сообщений, то решение практически очевидно:

— ввести аргументы функции (A и B);

— вычислить знаменатель;

— если знаменатель равен 0, то выводится “сообщение”;

— если знаменатель не равен 0, то выводится результат.

Все инструкции алгоритма максимально детализированы, поэтому составим текст программы.

Нельзя считать хорошей программу для ЭВМ, если в результате ее исполнения на экран ПК выводится одна малозначащая фраза. Поэтому дополним вывод результатов заголовком, значениями аргументов и расширенным сообщением для первой ветви.

Вариант программы будет иметь вид:

```
PROGRAM Prim17;
{Программа вычисления функции }
  VAR
     A,B,Y : REAL;
     {A, B — Аргументы }
     {Y—Функция}
 BEGIN
   {Этап ввода исходных данных}
  WRITE(‘Введите A=’);
  READLN(A);
  WRITE(‘Введите B=’);
  READLN(B);
{Вычисления и вывод}
 Y:=1-A*A+B*B;
 IF Y=0 THEN WRITELN(‘Нет решения!’);
 IF Y<>0 THEN WRITELN(‘Функция =’, l/Y:7:4);
END.
```

Пример 1.8. Определить, попадает ли точка с координатами *x*, *y* в круг радиуса *r* с центром в начале координат (уравнение окружности $r^2 = x^2 + +y^2$).

Решение.

Вариант программы будет иметь вид:

```
PROGRAM Prim18;
```

```
VAR
  x, y, r : REAL;
BEGIN
 WRITE('Введите радиус окружности R=');
 READLN(R);
 WRITE('Введите координаты точки X и Y');
 READLN(X, Y);
 IF X*X+Y*Y<=R*R
      THEN WRITELN('Точка лежит внутри круга')
      ELSE WRITELN('Точка лежит вне круга');
END.
```

Пример 1.9. Составить программу для упорядочивания трех чисел *a*, *b*, *c* по возрастанию таким образом, чтобы имени *a* соответствовало наименьшее число, имени *b* – среднее, а имени *c* – наибольшее.

Решение.

Вариант программы будет иметь вид:

```
PROGRAM Prim19;
  LABEL 10, 20, 30;
  VAR A, B, C, H : REAL;
  BEGIN
    WRITE('Введите три числа a, b, c:');
     READ(a,b,c);
     IF A<=B THEN GOTO 10
                ELSE BEGIN
                          H:=A;
                          A:=B;
                          B:=H
                        END;
   10: IF A<=C THEN GOTO 20
                ELSE BEGIN
                          H:=A;
                          A:=C;
                          C:=H
                        END;
   20: IF B<=C THEN GOTO 30
                ELSE BEGIN
                          H:=B;
                          B:=C;
                          C:=H
                        END;
    30: WRITELN('A=',A:5:2,'  B=',B:5:2,'  C=',C:5:2);
 END.
```

Контрольные вопросы

1. Что такое вычислительный процесс разветвляющейся структуры?
2. Какие управляющие конструкции в Турбо Паскале используются для организации разветвления?
3. Какова последовательность действий при выполнении условного оператора?
4. Какие действия выполняются оператором перехода goto?
5. Почему не рекомендуется использование в программах оператора goto?
6. Какие особенности существуют при написании вложенных операторов If?
7. Какой оператор позволяет выполнить одно из нескольких действий в зависимости от результата вычисления выражения?

Задачи

Эти задания предназначены для приобретения навыков организации разветвлений.

1. На плоскости расположена окружность радиуса R с центром в начале координат. Ввести заданные координаты точки, определить, лежит ли она на окружности. Решить задачу при $R = 2$ для точек с координатами (0, 2), (-1.5, 0.7), (1, 1), (3, 0).

Указание. Считать, что точка с координатами x, y лежит на окружности радиуса R, если $x^2 + y^2 — R^2 < 10^{-3}$.

2. Вычислить площадь треугольника со сторонами a, b, c по формуле Герона, проверив условие корректности исходных данных (длины всех сторон положительны, сумма длин любых двух сторон больше длины третьей).

3. Для заданных a и b получить

$$C=\begin{cases} \max(a,b), \text{ если } a > 0, \\ \min(a,b), \text{ если } a \le 0. \end{cases}$$

4. Для заданных a, b, c вычислить z =max(min(a,b),c).

5. Заданы площади круга R и квадрата S. Определить, поместится ли квадрат в круге. Задачу решить при: 1) R = 70, S =36.74; 2) R=0.86, S= 0.74.

Указание. Для решения задачи выразить диагональ квадрата и диаметр круга через заданные площади фигур. Квадрат поместится в круге, если диагональ квадрата не превышает диаметр окружности.

6. Для задачи 5 определить, поместится ли круг в квадрате. Задачу решить при: 1) R = 3.2, S= 3.5; 2) R = 3.2, S = 4**;** 3) R = 6, S = 9.

В задачах 7 —10 определить значение функции y при заданном аргументе x.

7. $$y = \begin{cases} 1, & \text{если } |x| > 1 \\ |x|, & \text{если } |x| \le 1 \end{cases}$$

8. $$y = \begin{cases} 1, & \text{если } |x| \ge 1 \\ -\sqrt{1 - x^2}, & \text{если } |x| < 1. \end{cases}$$

9. $$y = \begin{cases} 0, & \text{если } x \le -1 \\ 1 + x, & \text{если } -1 < x \le 0 \\ 1, & \text{если } x > 0. \end{cases}$$

10. $y = \begin{cases} 1, & \text{если } x \le -1 \\ x^2, & \text{если } -1 < x < 1 \\ 1, & \text{если } x \ge 1 \end{cases}$

В задачах 11 — 15 определить принадлежность точки (x,y) заданной области. Координаты трех точек задать самостоятельно.

11. $y \ge 0$ и $x^2 + y^2 \le 1$.
12. $y \ge |x|$ и $y \le 1 - x^2$.
13. $y \ge 0$, $y^2 \ge 4 - x^2$ и $y^2 \le 9 - x^2$.
14. $x \ge 0$, $y \ge 0$ и $x^2 + y^2 \le 4$.
15. $x \ge 0$, $y \le 0$ и $y^2 \le 9 - x^2$.

1.5. Реализация циклических алгоритмов

Циклические алгоритмы

Часто при решении задач приходится многократно выполнять одни и те же действия при различных значениях входящих в них величин. Такие многократно повторяющиеся участки вычислительного процесса называются *циклами*.

Соответственно циклический алгоритм — это алгоритм, содержащий циклы.

Использование циклов позволяет существенно сократить схему алгоритма и длину соответствующей ему программы.

Для организации любого цикла необходимы блоки, выполняющие следующие функции:

1. Задание начального значения переменной, изменяющейся в цикле.
2. Изменение переменной перед каждым новым повторением цикла.
3. Проверку условия окончания цикла и выход из него, если цикл закончен.
4. Переход к началу цикла, если цикл не закончен.

Отметим, что возможен «досрочный» выход из цикла с помощью условного оператора и оператора перехода, а также процедур Break или Exit.

Различают два типа циклов: *циклы с известным числом повторений (арифметические) и циклы с неизвестным числом повторений (итерационные).* Подчеркнем, что число повторений определяется на момент разработки алгоритма.

Цикл, содержащий внутри себя один или несколько других циклов, называется *вложенным*. Цикл, охватывающий другие циклы, называется внешним, а остальные - внутренними. Правила организации для внешних и внутренних циклов такие же, как и для простого цикла. Параметры этих циклов изменяются не одновременно: при одном значении параметра внешнего цикла параметр внутреннего цикла принимает по очереди все свои значения. В качестве параметров для этих циклов должны использоваться переменные с разными именами.

Реализация циклических алгоритмов

Для реализации циклов в программах на языке Турбо Паскаль можно использовать условные операторы в сочетании с оператором перехода, но наибольший эффект дают специальные операторы— *операторы циклов.*

Прежде всего рассмотрим циклы с заданным (известным) числом повторений.

Цикл с известным числом повторений

Цикл с известным числом повторений в Турбо Паскаль называется также *циклом с параметром*. Общая форма конструкции, образующей цикл, имеет вид:

- при увеличении значения параметра

FOR <*переменная*> = Выр.1 TO Выр.2 <оператор>

- при уменьшении значения параметра

FOR <*переменная*> = Выр.1 DOWNTO Выр.2 <оператор>

Здесь FOR *(для),* TO *(до)*, DOWNTO — ключевые слова языка Турбо Паскаль. <*Переменная*> называется *параметром цикла,* или управляющей переменной цикла. В качестве нее можно использовать любую переменную порядкового типа.

*Выр.1, Выр.*2 — выражения, определяющие соответственно начальное и конечное значения параметра цикла. Они могут быть записаны константами или выражениями, совпадающими по типу с параметром цикла. *Выр.1, Выр.2* вычисляются до входа в тело цикла.

Первая строка конструкции *(оператор* FOR) обычно называется *заголовком цикла.* Оператор, следующий за ним, образует *тело цикла.* Это может быть любой исполнимый оператор языка, в том числе и составной .

Примеры реализации циклического алгоритма

Пример 1.10. Разработать программу для вычисления суммы *K* членов ряда, определяемых общей формулой

$C_i = (-1)^{i+1} \frac{i!}{x^i}$ для аргумента $x > 0$.

Решение.

Анализ постановки задачи

В формуле для членов ряда символом «!» обозначена функция, называемая факториалом и определяемая в виде:

$$K!=1\cdot 2\cdot 3\cdot \ldots \cdot K.$$

Исходными данными для решения задачи, очевидно, являются число членов ряда *K* и значение аргумента *X*.

Обозначим вычисляемую сумму через *S,* счетчик ряда — *I,* очередной член ряда — *C*.

Вычисление суммы ряда — это цикл с известным (заданным) числом повторений (от 1 до *K)*, в котором не только вычисляются текущие значения членов ряда, но и накапливается их сумма путем прибавления полученного значения члена ряда к сумме предыдущих. В нашем примере формулой для накопления суммы нескольких слагаемых является формула $S_i := S_{i-1} + C_i$. Таким образом, значению суммы на *i*-м шаге присваивается значение частичной суммы на предыдущем шаге плюс слагаемое C_i. Поскольку надобности в запоминании значений всех промежуточных сумм и членов ряда нет, в качестве *S* нужно использовать простую переменную и накопление суммы вести по формуле *S*:=*S*+*C*.

До ввода в цикл вычисления суммы *S* его надо подготовить, т.е. присвоить *S* нулевое значение (“обнулить”) — *S* = 0, а перед накапливанием вычислить очередное слагаемое *C* — очередной член ряда.

Будет совершенно нерационально использовать для вычисления очередного члена ряда общую формулу, указанную в постановке задачи. Действительно, например, для 5-го номера получим:

$C_5 = +\frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}{x \cdot x \cdot x \cdot x \cdot x}$, а для шестого – $C_6 = -\frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6}{x \cdot x \cdot x \cdot x \cdot x \cdot x}$.

Отсюда следует, что для вычисления очередного слагаемого можно использовать оператор присваивания:

C := -*C***I*/*X*.

Начальное значение переменной *C* надо определить до входа в цикл: *C* = -1.

После выхода из цикла необходимо вывести полученные результаты. Форма вывода результатов в постановке задачи не определена, выбираем ее сами, например, в виде:

Результаты вычислений
Число членов ряда: * * *
Аргумент ряда: **.**
Сумма ряда: ****.***

Этот макет реализуется необходимой последовательностью операторов вывода. Таким образом мы достаточно детализировали алгоритм решения задачи.

Текст программы

```
PROGRAM Prim110;
{Вычисление суммы ряда }
  VAR X, C, S : REAL;
     K, I     : INTEGER;
     {К—число членов ряда }
     {X — аргумент }
     {C—очередной член ряда }
     {I—номер члена ряда }
     {S - вычисляется сумма }
  BEGIN
  {Этап ввода исходных данных }
WRITE(‘Число членов ряда-‘);
READLN(K);
WRITE(‘Аргумент (больше 0)');
READLN( X);
{Этап вычислений }
{Подготовка цикла}
S:=0;
C:= -1;
FOR I:=1 TO K DO
BEGIN
{Вычисление значения члена ряда}
   C:= -C*I/X;
   {Накопление суммы}
   S:=S+C;
```

```
END;
{Этап вывода }
WRITELN('Результаты вычислений');
  WRITELN('Число членов ряда: ' ,K:3);
  WRITELN('Аргумент ряда: ', X:5:2);
WRITELN('Сумма ряда: ', S:7:3);
END.
```

Формальное исполнение программы

Принимаем K=4. Значение аргумента можно не конкретизировать, записывая каждый раз выражение для вычисления слагаемого. Результаты формального исполнения:

Подготовка: $S = 0$, $C = -1$

$i = 1 \quad c = +1/x , \quad s = +1/x;$

$i = 2 \quad c = -1 \cdot 2/x^2, \; s = +1/x - 1 \cdot 2/x^2;$

$i = 3 \quad c = +1 \cdot 2 \cdot 3/x^3,$

$s = +1/x - 1 \cdot 2/x^2 + 1 \cdot 2 \cdot 3/x^3;$

$i = 4 \quad c = -1 \cdot 2 \cdot 3 \cdot 4/x^4,$

$s = +1/x - 1 \cdot 2/x^2 + 1 \cdot 2 \cdot 3/x^3 - 1 \cdot 2 \cdot 3 \cdot 4/x^4.$

Результаты, полученные при формальном исполнении программы, свидетельствуют о правильности составленного алгоритма.

Пример 1.11. Вычислить факториал $p = n!$, ($n! = 1 \cdot 2 \cdot 3 \ldots n$) при $n = 6$.

Решение.

Вариант программы будет иметь вид:

```
PROGRAM Prim111;
  VAR
     N, P, I : INTEGER;
  BEGIN
N:=6;
P:=1;
For I:=1 TO N DO
   P:=P*I;
WRITELN('P=',P)
  End.
```

Пример 1.12. Найти и напечатать все делители натурального числа N.

Решение.

Перед составлением программ по обработке целых чисел необходимо ознакомиться с арифметическими операциями DIV и MOD.

Напомним, что операция DIV является операцией целочисленного деления и выделяет целую часть от деления одного целого числа на другое.

Операция MOD выделяет остаток от деления одного целого числа на другое.

Например:

```
A:=5 DIV 2;
```

WRITE(A);
На экране появится «2».
M:=5 MOD 2;
WRITE(M);
На экране появится «1».

Если даны 2 числа A и B, то по определению B является делителем числа A, если A делится на B без остатка.

Минимальное количество делителей у каждого целого числа равно двум — это единица и само число.

Число, имеющее только два делителя, называется простым.

В данном алгоритме делители определяются перебором всех чисел в диапазоне от 2 до N/2. Если заданное число N делится без остатка на какое-либо из этих чисел, то это число будет являться одним из делителей числа N.

Диапазон делителей ограничен сверху величиной N/2, так как ни одно число не делится без остатка на число, большее своей половины.

Вариант программы

```
PROGRAM Prim112;
  TYPE
      NATUR = 1 .. MAXINT;
   VAR
      N : NATUR;
       I : INTEGER;
   BEGIN
  WRITE('Введите число');
   READLN(N);
   WRITELN('Делители числа',N, ':');
    WRITE( 1);
    FOR I:=2 TO N DIV 2 DO
         IF N MOD I = 0 THEN WRITE( I:5);
    WRITELN(N); END.
```

В приведенной программе делители только выводятся на экран, но не запоминаются. В ряде случаев необходимо запоминать делители в оперативной памяти. Для этого создается одномерный массив, имеющий размерность, достаточную для запоминания всех делителей числа.

Пример 1.13. Ввести с клавиатуры границы диапазона трехзначных натуральных чисел, из которых напечатать только простые. Определить количество таких чисел и их сумму.

Решение.

Обозначим:

A - нижняя граница диапазона;

B - верхняя граница диапазона;

S - сумма простых чисел;

KOL - количество простых чисел;

FL - признак простого числа (0 - простое число, 1 - непростое).

В приведенном алгоритме простые числа определяются перебором всех чисел

данного диапазона и вычислением делителя каждого из них (во вложенном цикле FOR I:=...).

Если у числа только два делителя (единица и само число), то число простое (FL=0), если больше, то число непростое (FL=1).

Вариант программы

```
PROGRAM Prim113;
    VAR
        A, B, N : 100 ..999;
         FL, D, S, KOL : INTEGER;
    BEGIN
WRITE('Нижняя и верхняя границы диапазона трехзначных чисел');
     READLN(A, B);
     S:=0;
     KOL:=0;
     FOR N:=A TO B DO
      BEGIN
      FL:=0;
      FOR D:=2 TO N DIV 2 DO
      IF N MOD D=0 THEN BEGIN FL:=1; BREAK END;
       IF FL=0 THEN BEGIN KOL:=KOL+1; S:=S+N END;
       END;
     WRITELN('Сумма ',S,' и количество простых чисел ', KOL)
    END.
```

Задача 1.14. В произвольной последовательности *N* вещественных чисел определить максимальное количество подряд идущих нулей.

Решение.

Обозначим объекты задачи:

N - количество чисел последовательности,

S - счетчик подряд идущих нулей,

MAX - максимальное количество подряд идущих нулей,

X - переменная с очередным числом последовательности.

Возможный вариант программы

```
PROGRAM Prim114;
VAR
  N, I, S, MAX : INTEGER;
  X : REAL;
BEGIN
  WRITE('Количество чисел в последовательности');
  READLN(N);
  S:=0;
  MAX:=0;
  FOR I:=1 TO N DO
    BEGIN
      WRITE('Значение ', I:4, '-го числа последовательности ');
```

```
    READLN( X);
     IF X=0 THEN S:=S+1
            ELSE S:=0;
    IF S > MAX THEN MAX:=S
  END;
     WRITELN('Максимальное количество подряд идущих нулей = ',
MAX:4);
END.
```

Цикл с неизвестным числом повторений

Цикл с неизвестным числом повторений называется также итерационным циклом. В языке Турбо Паскаль имеются два оператора цикла - WHILE и REPEAT, которые используются для организации цикла с неизвестным числом повторений.

Общая форма конструкции, образующей цикл WHILE, имеет вид:

WHILE <выражение> DO <оператор>

Здесь WHILE (пока); DO — ключевые слова языка; <выражение> — логическое выражение, которое определяет условие продолжение цикла. Первая строка (оператор WHILE) конструкции называется заголовком цикла.

<Оператор> образует тело цикла, состоящее из простого или составного оператора.

Выполнение оператора WHILE происходит следующим образом:

1. Вычисляется выражение, указанное в заголовке цикла.
2. Если выражение истинно, то один раз выполняется тело цикла и вновь вычисляется выражение.
3. Пункт 2 повторяется до тех пор, пока выражение истинно.
4. Если выражение ложно, то тело цикла не выполняется, осуществляется выход из цикла, т.е. управление передается оператору, который следует за оператором WHILE.

Общий вид записи оператора REPEAT:

REPEAT <оператор> UNTIL <выражение>

где <оператор> - тело цикла, состоящее из простого или составного оператора; <выражение> - логическое выражение, определяющее окончание цикла.

Выполнение оператора WHILE происходит следующим образом:

1. Выполняется тело цикла.
2. Вычисляется выражение, указанное во фразе UNTIL.
3. Если выражение ложно, то повторяются пп. 1 и 2.
4. Если выражение истинно, то выходим из цикла, т.е. управление передается оператору, который следует за оператором REPEAT.

Из сказанного следует важное правило: операторы тела цикла WHILE и REPEAT должны изменять выражение, указанное в операторе. В противном случае будет иметь место бесконечный цикл, что противоречит определению алгоритма.

В тело итерационного цикла допускается включать другие циклы. Это может быть другой итерационный цикл или цикл с известным числом повторений. Другими словами, в итерационных циклах никаких ограничений на вложенные циклы не накладывается.

Примеры реализации итерационного цикла

Пример 1.15. Составить программу для вычисления функции, заданной рекурсивными формулами:

$Y_0=(1+X)/3$

$Y_{k+1}=Y_k+(X/Y_k \cdot Y_k - Y_k)/3$ при k=l, 2, 3.... Вычисления закончить при выполнении условия:

$$|Y_{k+1}-Y_k| < e,$$

где e — точность вычислений (малое положительное число). Аргумент функции X— произвольное число.

Решение.

Анализ постановки задачи

Исходные данные для решения: аргумент функции — X и точность вычисления — E.

То, что заданы рекурсивные формулы для вычислений, означает повторяемость (цикличность) вычислений: каждое новое значение функции вычисляется по ее предыдущему значению. Число вычислений неизвестно, но известно условие продолжения вычислений: пока $|Y_{k+1} - Y_k| \geq E$.

Введем обозначения объектов программы: Y0 — предыдущее (в том числе начальное) значение функции; Y —очередное (в том числе конечное) значение функции; Z — модуль разности значений.

Для решения задачи будем использовать цикл WHILE.

Как и любой другой, этот цикл требует подготовки. В первую очередь необходимо обеспечить истинность выражения Z≥E.

Значение E определим на этапе ввода исходных данных, а значение Z определим до входа в цикл, присвоив ему любое значение, которое больше E, например Z:=2*E.

Также до входа в цикл необходимо вычислить начальное значение функции Y0:=(1+X)/3.

Наконец, тело цикла. В нем вычисляется очередное значение функции Y:=Y0+(X/Y0/Y0-Y0)/3.

Здесь же необходимо обеспечить изменение значения выражения в заголовке цикла Z:=ABS(Y-Y0).

Наконец, необходимо подготовить следующую итерацию Y0:=Y.

Выход из цикла произойдет при достижении заданной точности вычислений. После этого можно выводить полученные результаты.

С целью большей информативности результатов дополним алгоритм вычислением числа выполненных итераций К:=К+1

Начальное значение К присвоим до цикла К:=1

Результаты вычислений выведем в следующем виде:

Результаты вычислений
Аргумент: **.**
Заданная точность: *.*****

Значение функции: ****.***

Число итераций: ***

Поскольку алгоритм уже достаточно детализирован, составим текст программы:

Текст программы

```
PROGRAM Prim115;
{Вычисление функции с заданной точностью}
  VAR
        X, Y0, Y, E, Z : REAL;
        K : INTEGER;
{X – Аргумент функции}
{Y0, Y – Начальное и конечное значения функции}
{E – Заданная точность}
{Z=ABS(Y-Y0)}
{K – число итераций}
BEGIN
{Этап ввода исходных данных}
  WRITE(‘Аргумент функции’);
  READLN(X);
  WRITE(‘Заданная точность вычислений’);
  READLN(E);
{Этап вычислений}
  Z:=2*E;
  Y0:=(1+X)/3;
  K:=1;
  WHILE Z>=E DO
    BEGIN
      Y:=Y0+(X/Y0/Y0-Y0)/3;
      Z:=ABS(Y-Y0);
      Y0:=Y;
      K:=K+1
    END;
{Этап вывода результатов}
  WRITELN(‘Результаты вычислений');
  WRITELN(‘Аргумент:’,X:5:2);
  WRITELN(‘Заданная точность:',E:7:5);
  WRITELN(‘Значение функции:’,Y:8:3);
  WRITELN(‘Число итераций:’,K:3);
END.
```

Пример 1.16. Вычислить сумму $S = \cos(x) + \frac{\cos(2x)}{2} + \cdots + \frac{\cos(nx)}{n}$ при x =0,5.

Суммирование прекращать, когда очередной член суммы будет меньше заданного e = 0.0001.

Решение.

В приведенной ниже программе используется цикл REPEAT:

```
PROGRAM Prim116;
  VAR
      X, Eps, S, A : REAL;
N : INTEGER;
BEGIN
 X := 0.5;
 Eps:=0.0001;
 S:=0;
 N:=1;
 REPEAT
   A:=cos(n*x)/n;
   S:=s+a;
   N:=N+1;
 UNTIL a<Eps;
 WRITELN( ‘Сумма равна ‘,S:8:5);
End .
```

Пример 1.17. Определить приближенное значение суммы элементов ряда:

$$S = 1 + x/1! + x^3/3! + x^5/5! + ...$$

Суммирование выполнять до тех пор, пока очередное слагаемое не станет меньше заданной величины E.

Решение.

Анализ условия задачи

Исходными данными для решения задачи являются значения X и E, где E — точность вычислений.

Вычисление суммы элементов ряда — цикл с неизвестным числом повторений.

Очевидно, что условие выполнения операторов тела цикла можно записать в виде

WHILE A>=E DO,

где A — очередной элемент ряда;

E — заданная точность вычислений.

В теле цикла следует:

— накапливать в переменной S значение предыдущего элемента ряда;

— получать новое значение элемента ряда.

Для вычисления нового элемента ряда необходимо определить закономерность изменения последующего элемента по сравнению с предыдущим.

Чтобы найти эту закономерность, попарно сравним соседние элементы и проанализируем выражение, на которое в каждом цикле нужно умножать предыдущий элемент. Результаты сравнения представлены в таблице.

Номер цикла	Предыдущий элемент	Выражение	Новый (последующий) элемент
1	$X/1$	$X^2/2\cdot3$	$X^3/1\cdot2\cdot3$
2	$X^3/1\cdot2\cdot3$	$X^2/4\cdot5$	$X^5/1\cdot2\cdot3\cdot4\cdot5$
3	$X^5/1\cdot2\cdot3\cdot4\cdot5$	$X^2/6\cdot7$	$X^7/1\cdot2\cdot3\cdot4\cdot5\cdot6\cdot7$

Из приведенной таблицы видно, что числитель выражения от цикла к циклу не меняется: X^2.

Знаменатель — произведение двух соседних натуральных чисел, то есть если первое число обозначим переменной К, то в общем виде выражение можно записать: $P=X\cdot X/(K\cdot(K+1))$, а значение К изменяется в каждом цикле на 2.

Итак,

```
 WHILE A>=E DO
Begin
  S:=S+A;
  P:=X*X/(К*(К+1)) ;
  A:=P*A ;
  К:=К+2 ;
END;
```

И последнее. До входа в цикл следует определить значения переменных S, A, К:

```
S:=1;
A:=X;
К:=2;
```

(см. условие примера)

Возможный вариант программы

```
PROGRAM Prim117;
VAR
  X, E, S, A, K, P: REAL;
{X – переменная для ввода и хранения аргумента
  E – переменная для ввода и хранения точности вычислений
  S – переменная для получения суммы элементов ряда
  A – переменная для получения очередного элемента ряда
  К – переменная для получения знаменателя
  P – переменная для вычисления значения выражения, на которое умножается
      предыдущий элемент}
{Блок ввода информации}
BEGIN
WRITE (‘Введите значение аргумента’);
READLN(X);
WRITE (‘Введите точность вычислений’);
READLN(E);
{Блок вычисления суммы}
```

```
S:=1;
A:=X;
K:=2;
WHILE A>=E DO
BEGIN
  S:=S+A;
  P:=X*X/(K*(K+1)) ;
  A:=A*P ;
  K:=K+2 ;
  END ;
WRITELN('При значении аргумента = ' , X);
WRITELN('Сумма элементов ряда = ', S)
END.
```

Пример 1.18. Вычислить сумму

$$S = 1 + \frac{x^2}{2} - \frac{x^4}{8} + \cdots + (-1)^{n-1}\frac{(2n-1)x^{2n}}{(2n)!}$$

для значений x, изменяющихся в пределах от 0.2 до 1 с шагом 0.2. Суммирование прекращать, когда очередной член суммы по абсолютной величине станет меньше $e = 0.0001$.

Решение.

Задача сводится к организации вложенных циклов. Внешний цикл по счетчику обеспечивает изменение x. Во внутреннем цикле по условию осуществляется вычисление суммы.

Член суммы a_n целесообразно представить в виде двух сомножителей:

$$a_n = c_n (2n - 1).$$

Сомножитель $c_n = (-1)^{n-1}\frac{x^{2n}}{(2n)!}$ будем вычислять по рекуррентной формуле, выражая последующий член через предыдущий:

$$c_n = -c_{n-1}\frac{x^2}{(2n-1)2n}.$$

Число значений x на отрезке от 0.2 до 1 с шагом 0.2 равно 5. В программе для контроля при каждом значении x вычисляется также функция $\cos x + x \sin x$, которая приближенно может быть представлена в виде указанной суммы.

```
PROGRAM Prim118;
  VAR
       X, S, EPS,C,A : REAL;
       I, N : INTEGER;
 BEGIN
  X:=0.2;   Eps:=0.0001;
For I:=1 To 5 DO
   BEGIN
     S:=1;
     C:=-1;
```

```
      N:=1;
      REPEAT
        C:=-c*x*x/((2*n-1)*2*n);
        A:=c*(2*n-1);
        S:=s+a;
        N:=N+1
      UNTIL Abs(a)<Eps;
    WRITELN( 'x=',x, ' s=',s, ' f(x)=', cos(x) + x*sin(x));
     X:=X+0.2
   END
End.
```

Пример 1.19. Вычислить сумму ряда $S = \frac{1}{2 \cdot 1} + \frac{1}{2 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{2 \cdot 4} \cdots$. Суммирование осуществлять, пока разность между предыдущим и текущим членами остается больше 0,04. Кроме суммы вывести на экран значение последнего слагаемого и его номер.

Решение.

Обозначим объекты программы:

S- сумма;

N - количество членов ряда;

TEK - текущий член ряда;

PR - предыдущий член ряда.

Вычисление данной суммы будем осуществлять в цикле WHILE до тех пор, пока разность по модулю между текущим и предыдущим значениями членов ряда будет больше заданной величины, т.е.

WHILE ABS(TEK-PR)>0.04 DO

Так как выход из цикла осуществляется при нарушении данного условия, то для вывода правильного значения суммы, в которой разность по модулю между текущим и предыдущим слагаемыми еще остается больше 0,04, следует определить значение суммы, а также значение и номер слагаемого, предшествующего сумме ряда при выходе из цикла.

Приведем вариант программы

```
PROGRAM Prim119;
  VAR S, TEK, PR : REAL;
     N : INTEGER;
  BEGIN
    S:=0.5;
    TEK:=0.5;
    PR:=0;
    N:=1;
    WHILE ABS(TEK-PR)>0.04 DO
     BEGIN
      N:=N+1;
      PR:=TEK;
      TEK:=1/(2*N);
```

```
        S:=S+TEK;
      END;
   WRITELN('Сумма= ', S-TEK);
   WRITELN('Последнее слагаемое ',PR,' и его номер ',N-1);
END.
```

Пример 1.20. Найти максимальное количество слагаемых в сумме членов ряда $7+\frac{7}{2\cdot1}+\frac{7}{3\cdot2\cdot1}+\frac{7}{4\cdot3\cdot2\cdot1}+\cdots$, при котором эта сумма остается меньше 12. Кроме этого вывести на экран значение последнего слагаемого и его номер.

Решение.

Введем обозначения объектов:

S – сумма;

N - количество слагаемых в сумме;

A - очередное (текущее слагаемое).

Формула общего члена данного ряда имеет вид:

$$\frac{7}{n!},$$

где *n* - порядковый номер слагаемого.

Суммирование осуществляется в цикле WHILE до тех пор, пока сумма остается меньше 12, т.е.

WHILE S<12

При выводе результатов следует учесть, что после выхода из цикла значение суммы будет больше или равно 12. Поэтому следует «откатиться» к предыдущему значению суммы, т.е. максимальному значению суммы, еще остающейся меньше 12.

Предлагаем следующий вариант.

```
PROGRAM Prim120;
  VAR A, S : REAL;
       N : INTEGER;
  BEGIN
S:=0;
N:=0;
A:=7;
WHILE S<12 DO
 BEGIN
  N:=N+1;
  A:=A/N;
  S:=S+A;
 END;
WRITELN('Максимальная сумма <12= ', S-A);
WRITELN('Последнее слагаемое ',A*N,' и его номер ', N-1);
END.
```

Пример 1.21. В числовую переменную последовательно вводятся отличные от нуля целые числа. Количество чисел заранее неизвестно. Определить и вывести на

экран, какие (четные или нечетные) числа были введены последними и какова их сумма.

Решение.

В данном условии задачи обратите внимание на фразу «количество чисел заранее неизвестно». Это означает, что количество вводимых чисел может быть определено (подсчитано) только при завершении их ввода, а до начала ввода данных сведения об их количестве просто отсутствуют.

Кроме того, при составлении данной программы должны быть задействованы два счетчика сумм подряд идущих четных и нечетных чисел, которые при отсутствии подсчитываемых значений должны обнуляться.

Возможный вариант программы.

```
PROGRAM Prim121;
  VAR
 S1, S2, X : INTEGER;
 {S1 - сумма подряд идущих нечетных чисел,
  S2 - сумма подряд идущих четных чисел,
  X - переменная, содержащая введенное число}
BEGIN
 S1:=0;
 S2:=0;
 WRITE(‘Введите число ‘);
 READLN(X);
 WHILE X<>0 DO
      BEGIN
       IF X MOD 2<>0
       THEN
         BEGIN
           S1:=S1+X;
           S2:=0
         END
       ELSE
         BEGIN
           S2:=S2+X;
           S1:=0
         END;
       WRITE(‘Введите число’);
       READLN(X);
     END;
 IF S1<>0 THEN
WRITELN('Последними были введены нечетные числа. Их сумма = ‘, S1);
 IF S2<>0 THEN
WRITELN(‘Последними были введены четные числа. Их сумма =’, S2);
 END.
```

Пример 1.22. Найти сумму и количество цифр натурального числа N.

Решение.

Обозначим:

S - сумма цифр числа;

K - количество цифр числа;

Z - выделенная цифра числа;

CH - целое частное от деления.

Предложенный алгоритм базируется на выделении всех цифр числа, начиная с младшего разряда, методом последовательного деления и получения целых частных на основании данной системы счисления до тех пор, пока целое частное от деления не станет равным нулю.

Текст программы

```
PROGRAM Prim122;
   VAR
    S, K, Z, CH : INTEGER;
     N : 1.. MAXINT;
    BEGIN
     WRITE('Введите натуральное число ');
      READLN(N);
      S:=0;
      K:=0;
      CH:=N;
      WHILE CH<>0 DO
         BEGIN
           K:=K+1 ;
           Z:=CH MOD 10;
           S:=S+Z; CH:=CH DIV 10;
         END;
      WRITELN('Количество ',K,' и сумма цифр ', S)
    END.
```

Пример 1.23. В числовую переменную вводится число в десятичной системе счисления, не превышающее по модулю 10000. Необходимо вывести на экран представление этого числа в двоичной системе счисления.

Решение.

Обозначим объекты программы:

N - целое число;

K - количество цифр двоичного числа;

Z(14) - массив с цифрами двоичного числа;

CH - целое частное.

Перевод десятичного числа в двоичную систему счисления можно описать следующей последовательностью шагов:

1. Последовательно делим данное число и получаемые целые частные на 2 до тех пор, пока последнее частное не станет равным нулю.
2. Запоминаем все остатки от деления данного числа и целых частных на 2.

3. Составляем число в двоичной системе счисления, начиная с последнего остатка и кончая первым.

Для запоминания остатков от деления надо организовать в памяти одномерный массив. Но какой размерности?

По условию, целое десятичное число не должно превышать 10000, т.е. $10000_{10} = 2710_{16} = 10\ 0111\ 0001\ 0000_2$.

Отсюда видно, что максимальное количество значащих разрядов двоичного числа не может превышать 14. Эта величина и будет определять размерность одномерного массива.

При выводе изображения двоичного числа анализируется его знак, т.к. в условии задачи ничего не сказано о том, какие числа переводятся - положительные или отрицательные.

Вариант программы

```
Program Prim123;
Var
  Z : Array[1..14] of Integer;
  N, I, K, CH : Integer; S,q: Real;
Begin
 Write('Введите целое число ');
 Readln(N);
 K:=0;
 CH:=ABS(N);
 WHILE CH<>0 Do
  Begin
   K:=K+1;
   Z[K]:=CH MOD 2;
   CH:=CH DIV 2;
  End;
 Writeln('Число ',N,' в двоичной системе счисления');
 If N<0 THEN Write('-');
 q:=1; S:=0;
 For I:=1 TO K DO
  Begin
   S:=S+Z [I]*q;
   q:=q*10;
  End;
 Writeln(S:14:0);
End.
```

Контрольные вопросы

1. Что такое цикл?
2. Какие действия необходимо выполнить для организации цикла?
3. Что такое итерационный циклический процесс?

4. Какие операторы цикла имеются в Турбо Паскале?
5. Какие средства языка целесообразно использовать для организации циклов с заданным числом повторений?
6. Как работает оператор цикла While?
7. В чем заключается различие между операторами Repeat и While?
8. Каково может быть условие выхода из цикла при вычислении значения суммы бесконечного ряда?
9. Зачем используются рекуррентные соотношения для вычисления значений члена ряда?

Задачи I уровня

Определить, какой цикл (по счетчику или по условию) требуется в задаче. Для цикла по условию определить, где должна производиться проверка условия выхода из цикла (в начале или в конце тела цикла) и, исходя из этого, использовать соответствующие операторы цикла.

1. Вычислить 1 + 1/2 + 1/3 + ... + 1/10.
2. Вычислить 5 + 8 + 11 + ... + 35.
3. Вычислить $1 + 3 + 3^2 + ... + 3^7$.
4. Вычислить $1/2 + 1/2^2 + ... + 1/2^{10}$.
5. Вычислить 2/3 + 3/4 +... + 10/11.
6. Вычислить $2 + 2^3 + ... + 2^{10}$.
7. Вычислить $y = 1/2 + 2x^3$ при x = -5, -4.5, - 4,... , 3.
8. Составить таблицу стоимости порций сыра весом 50, 100,... , 1000 г (цена 1 кг сыра 80 руб.).

В задачах № 9-15 вычислить сумму заданного числа членов знакопеременного ряда.

9. $S = \frac{2}{3}\sum_{i=1}^{25}(-1)^i \frac{2i}{i^2+2}$

10. $S = \sum_{i=1}^{45}(-1)^{i+1} \frac{i^2}{i^3+3}$

11. $S = 0.3\sum_{i=1}^{15}\frac{(-1)^{i+1}}{(i+1)(i+2)}$

12. $S = 2 + \frac{1}{2}\sum_{l=1}^{15}(-1)^{l+3} \frac{3l}{l^3+1}$

13. $S = 1 + \sum_{k=1}^{30}\frac{(-1)^k}{(k+1)k^3}$

14. $S = \frac{1}{2}\sum_{m=1}^{25}\frac{(-1)^m}{m^2+5m+1}$

15. $S = 0.4 \sum_{n=1}^{25} (-1)^{n+2} \frac{n+3}{n^2+3n+5}$

Задачи II уровня

1. Имеется последовательность ненулевых целых чисел, за которой следует 0. Определять, сколько раз в последовательности меняется знак чисел.

2. Составить программу для вычисления значения:

$y=x + x^3/3! + x^5/5! + x^7/7! +...$,учитывая, что $|x| < 1$.

Расчет продолжать до тех пор, пока модуль разности между очередным и предыдущим значениями y будет больше заданной величины (точности вычислений).

3. Имеется последовательность положительных и отрицательных чисел. Определить, является ли последовательность чисел, находящихся до первого отрицательного числа, возрастающей.

4. Имеется шестизначный номер лотерейного билета. Определить является ли номер «счастливым», т.е. равны ли суммы первых и последних трех цифр.

5. Имеется последовательность вещественных чисел, содержащая нулевые значения. Определить, сколько чисел, расположенных до первого нулевого значения, больше своих «соседей».

6. Перед паромной переправой остановились легковые и грузовые автомобили. Определить, сколько легковых автомобилей находится в начале очереди.

7. На шоссе образовалась «пробка» из легковых и грузовых автомобилей. Определить, сколько грузовых автомобилей находится между легковыми.

8. Вычислить сумму ряда $\frac{1}{2 \cdot 1} + \frac{1}{2 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{2 \cdot 4} + \frac{1}{2 \cdot 5} + \cdots$. Суммирование осуществлять, пока разность между предыдущим и текущим членами остается больше 0.04. Кроме суммы вывести на экран значение последнего слагаемого и его номер.

9. Вывести на экран N чисел Фибоначчи (N задается из диапазона от 2 до 21), значения частных от деления каждого из этих чисел на предыдущее число и сумму этих частных.

Числа Фибоначчи - это такая последовательность целых чисел, в которой два первых числа равны 1, а каждое следующее число равно сумме двух предыдущих. (Числа Фибоначчи: 1, 1, 2, 3, 5, 8,...).

10. В числовые переменные вводятся два натуральных числа X и N. Необходимо найти ближайшее к X и превосходящее его натуральное число, кратное N, а также сумму его цифр. Выдать на экран соответствующие сообщения.

11. Найти минимальное количество слагаемых в сумме членов ряда $1 \cdot 1 + 1 \cdot 2 \cdot 2 + 1 \cdot 2 \cdot 3 \cdot 3 + 1 \cdot 2 \cdot 3 \cdot 4 \cdot 4 + \cdots$, при котором эта сумма станет больше 120. Кроме этого вывести на экран значение последнего слагаемого и его номер.

12**.** Не используя массивы, ввести экзаменационные оценки N учеников по K предметам (N и K задаются). Определить и вывести на экран количество учеников,

не получивших:

а) ни одной "пятерки";

б) ни одной оценки выше "тройки".

13. В числовую переменную последовательно вводятся отличные от нуля целые числа. Количество чисел заранее неизвестно. Определить и вывести на экран, какие (четные или нечетные) числа были введены последними и какова их сумма.

14. Ввести с клавиатуры границы диапазона трехзначных натуральных чисел, из которых необходимо напечатать только простые. Определить количество таких чисел и их сумму.

15. Найти максимальное количество слагаемых в сумме членов ряда $7+\frac{7}{2\cdot1}+\frac{7}{3\cdot2\cdot1}+\frac{7}{4\cdot3\cdot2\cdot1}+\cdots$, при котором эта сумма остается меньше 12. Вывести на экран значение последнего слагаемого и его номер.

16. Вычислить последнюю сумму членов ряда $1+\frac{1}{4}+\frac{1}{12}+\frac{1}{32}+\frac{1}{80}+\cdots$, при которой разность между предыдущим и текущим членами ряда остается больше 0.007. Кроме суммы вывести на экран значение последнего слагаемого и его номер.

17. В числовую переменную последовательно вводятся отличные от нуля целые числа. Количество чисел заранее неизвестно. Определить и вывести на экран, какие (положительные или отрицательные) числа были введены последними и какова их сумма.

18. Вычислить сумму ряда 2 + 4 + 8 + 16 + 32 + ⋯. Суммирование осуществлять, пока разность между текущим и предыдущим членами остается меньше 200. Кроме суммы вывести на экран значение последнего слагаемого и его номер.

19. В числовую переменную последовательно вводятся произвольные числа, не равные нулю. Количество вводимых чисел заранее неизвестно. Определить порядковые номера и значения первого из максимальных и последнего из минимальных чисел этой последовательности, а также произведение двух этих чисел.

20. Вычислить сумму членов ряда $\frac{1^2\cdot5}{2\cdot4}+\frac{2^2\cdot6}{2\cdot4\cdot6}+\frac{3^2\cdot7}{2\cdot4\cdot6\cdot8}+\cdots$. Суммирование осуществлять, пока разность между текущим и предыдущим значениями суммы остается больше 0,001. Кроме этого найти значение последнего слагаемого и его номер.

21. В числовую переменную последовательно вводятся ненулевые целые числа. Количество вводимых чисел заранее неизвестно. Необходимо определить максимальную длину цепочки из подряд идущих положительных чисел и максимальную длину цепочки из подряд идущих отрицательных чисел.

22. В числовую переменную в произвольном порядке последовательно вводятся нулевые и положительные числа. Количество вводимых чисел заранее неизвестно. Определить сумму и количество введенных положительных чисел, а также максимальное количество подряд введенных нулевых чисел.

23. В числовую переменную последовательно вводятся произвольные числа, не

равные нулю. Количество вводимых чисел заранее неизвестно. Требуется вычислить и напечатать сумму тех из них, порядковые номера которых являются простыми числами.

24. В числовую переменную последовательно вводятся целые числа, не равные нулю. Количество вводимых чисел заранее неизвестно. Требуется найти сумму тех введенных чисел, в которых встречаются две цифры 5.

25. В числовую переменную последовательно вводятся ежедневные замеры дневных температур осенних и зимних дней. Количество вводимых замеров заранее неизвестно. Определить количество дней с температурой ниже чем минус 5 градусов, а также максимальное количество подряд идущих дней с нулевой температурой.

26. В одну и ту же переменную x вводятся вещественные числа по модулю больше 1 и меньше 2. Количество вводимых чисел заранее неизвестно. Для каждого значения x вывести на экран сумму ряда $1 + x + x/2 + x/3 + + x/5 + x/8 + x/13 + x/21 +$....., значение последнего слагаемого и его порядковый номер. Суммирование выполнять до тех пор, пока модуль разности между текущим и предыдущим членами остается больше 0.0001.

Глава 2. СТРУКТУРИЗАЦИЯ ПРОГРАММНЫХ ДАННЫХ

2.1. Массивы

Задачи, решаемые на ЭВМ, являются математическими моделями процессов или явлений реальной жизни. В математической модели находят отражение наиболее существенные связи между реальными объектами. Модели реальных объектов вместе с присущими им связями образуют структуры данных, процесс обработки которых и описывается с помощью алгоритмов.

Разные классы решаемых на ЭВМ задач характеризуются и разными структурами данных, что находит отражение и в соответствующих языках программирования.

Рассмотрим наиболее часто используемые структуры данных языка Турбо Паскаль.

Необходимость в массивах возникает всякий раз, когда при решении задачи приходится иметь дело с большим, но конечным количеством однотипных упорядоченных данных. Допустим, что требуется вычислить значение многочлена *P* (объект задачи):

$$P= a + bx + cx^2 + dx^3 + ex^4 + fx^5,$$

где *a, b, c, d ,e, f* — коэффициенты многочлена (объекты задачи арифметического типа). В алгоритме это переменные и их количество (шесть) определено в постановке задачи, т.е. к моменту разработки алгоритма.

Удобно такие разрозненные переменные объединить в совокупность —*массив,* именуя их затем общим именем (именем массива) и индексами (номерами в массиве). В рассматриваемом примере введем массив А с шестью элементами (a_1—a_6):

$$P = a_1 + a_2x + a_3\, a^2 + a_4x^3 + a_5x^4 + a_6x^5.$$

Массив *A* представляет собой одномерный массив, так как для индексации его элементов используется один индекс.

В задачах используются не только одномерные, но и многомерные массивы, в частности двумерные. Для индексации элементов двумерного массива указываются два индекса — номер *строки* и номер *столбца.* Для наглядности такой массив представляется в виде матрицы, состоящей из *m* строк и *n* столбцов.

Например, оценки трех учеников по четырем предметам можно представить массивом *X* из 3 строк и 4 столбцов:

$$\begin{matrix} X_{11} & X_{12} & X_{13} & X_{14} \\ X_{21} & X_{22} & X_{23} & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \end{matrix}$$

При этом одномерные массивы часто называют векторами, а двумерные - матрицами.

Массиву дается имя и указывается его размер.

Итак, *массив* - это поименованный упорядоченный набор фиксированного количества некоторых однотипных значений (компонент массива).

Имя всего массива называют *полной переменной*, поскольку ее значение есть весь массив.

К каждой компоненте массива имеется прямой доступ. Для этого после имени массива следует указать в квадратных скобках индекс компоненты, например X[10], MATR[4,5]. Такие переменные называются *переменными с индексом, или индексными переменными*. Значением такой переменной является не весь массив, а отдельная компонента, номер которой задается индексом. Поэтому переменную с индексом называют еще *частичной переменной*. Отметим, что в двумерном массиве первый индекс указывает номер строки, а второй - номер столбца.

В общем случае в качестве индекса может быть использована не только

константа, но и выражение, значение которого и определяет номер компоненты массива. При этом важно, что в индексное выражение могут входить переменные, так что при изменении их значений меняется и значение индекса, которое определяет номер компоненты массива, например A[K]; A[I + J]; X[1,J]; X[I + 2J]. Таким образом, одна и та же переменная с индексом в процессе выполнения программы может обозначать различные компоненты массива. Очень важно в таких случаях помнить, что переменные, используемые для индексации, к моменту обработки элементов массивов должны быть определены.

Возможны не только арифметические массивы, но массивы других типов, в частности строковые. Например, список фамилий (объект задачи) можно рассматривать как одномерный строковый массив.

Типовые действия с массивами

Массивы должны быть описаны в программе в разделе описания типов или переменных. Например:

```
CONST
   N=10;
TYPE
  Vector = ARRAY[1..20] OF REAL;
   Matr = ARRAY[1..N,1..5] OF BYTE;
VAR
  A, B : Vector;
 C    : ARRAY [1..N] OF INTEGER;
 D    : Matr;
```

Здесь описаны три одномерных и один двумерный массивы. Одномерные массивы *A* и *B* типа Vector состоят из двадцати элементов, а массив *C* – из десяти элементов. Двумерный массив *D* представляет собой матрицу размерами 10 ⨯ 5, имеющую десять строк и пять столбцов.

Далее приводятся фрагменты программ для реализация типовых действий с массивами. Предполагается, что описания массивов в программах имеются, а индексы начинаются с 1.

1. Ввод массивов с клавиатуры:

а) одномерного массива *A*

```
For I:=1 To N Do
                Read( A[i]);
```

б) двумерного массива *C*

```
For I:=1 To N Do {строки}
                For J:=1 To M  Do
                                {столбцы}
                               Read(C[i,j]);
```

При каждом выполнении оператора Read вводится один элемент массива (после набора каждого элемента нужно нажимать клавишу Enter (Ввод)). Ввод двумерного массива осуществляется в приведенном примере по строкам.

2. *Вывод на экран массивов:*

а) одномерного в строку

```
For I :=l To N Do
    Write(A[i]);
```

б) одномерного в столбец

```
For I :=l To N Do
    Writeln(A[i]);
```

в) двумерного по строкам

```
For I :=l To N Do
  Begin
 For J :=l To M  Do
     Write (C[i,j]);
Writeln;
     End.
```

3. *Суммирование элементов массивов:*

а) одномерного

```
S:=0;
For I :=l To N  Do
   S :=S+A[i];
```

б) двумерного

```
S:=0;
For I :=l To N Do
  For J :=l To M Do
      S:=S+C[i,j];
```

4. *Суммирование диагональных элементов матрицы* (вычисление следа матрицы):

```
S:=0;
For I :=l To N Do
  S:=S+C[i,i];
```

5. *Суммирование двух массивов по элементам* (оба массива -слагаемых должны полностью совпадать как по размерности, так и по типам их элементов):

а) одномерного

```
For i:=1 To N Do
  C[i]:=A[i]+B[i];
```

б) двумерного

```
For i:=l To N Do
   For j:=l To M Do
      C[i,j]:=A[i,j]+B[i,j];
```

6. *Суммирование элементов заданной k-й строки матрицы:*

```
S:=0;
For j:=l To M Do
  S:=S+C[k,j];
```

7. *Суммирование элементов строк матриц:.*

Необходимо вычислить сумму элементов каждой строки матрицы *C* размерами *N* ⨯ *M*. Результат получим в виде вектора *D*:

```
For i:=l To K Do
Begin
S:=0;
For j:=l To M Do
     S:=S+C[i,j];
  D[i]:=S;
  End;
```

8. *Транспонирование матрицы:*

Необходимо заменить строки матрицы ее столбцами, а столбцы — строками, т.е. вычислить $d_{ij} = c_{ji}$, i =1,... N, j = 1,... M:

```
For i:=l To N Do
   For j:=l To M Do
      D[i,j]:=C[j,i];
```

Транспонированную матрицу можно получить в исходном массиве C. Для квадратной матрицы размерами $N \times N$ для этого необходимо поменять местами каждый элемент верхнего треугольника с соответствующим элементом нижнего (диагональные элементы переставлять не нужно). При этом для каждой строки нужно выполнять перестановку элементов, расположенных правее главной диагонали, с элементами соответствующего столбца, расположенного ниже главной диагонали, т.е. взаимно перемещать элементы $C_{i,j}$ и $C_{j,i}$,i=1, ...,N-1; $j = i$+1, ... , N.

```
For i:=l To N-1 Do
   For j:=i+l To N Do
     Begin
       X:= C[i,j];
       C[i,j] := C[j,i];
       C[j,i] := X;
     End;
```

Для прямоугольной матрицы алгоритм усложняется.

9**.** *Умножение матрицы на вектор:*

Для вычисления произведения C матрицы A размерами $N \times M$ на вектор B размером M необходимо вычислить $C_i = \sum_{j=1}^{M} a_{ij} b_j$, i=1, ... N:

```
For i:=l To N Do
   Begin
  S:=0 ;
  For j:=l To M Do
     S:=S+A[i,j]*B[j];
  C[i]:=S;
End;
```

10. *Умножение матрицы на матрицу:*

Для умножения матрицы A размерами $N \times L$ на матрицу B размерами $L \times M$ необходимо вычислить $C_{ij} = \sum_{k=1}^{L} a_{ik} b_{kj}$, i = 1 ... N, j = 1,...M;

```
For i:=1 To M Do
  For j:=l To N Do
    Begin
    S:=0 ;
    For k:=l To L
        S:=S+A[i,k]*B[k,j];
    C[i,j]:=S;
```

```
    End;
```

11. *Удаление элемента из массива:*

Требуется удалить *k*—й элемент из массива *A* размером *N*. Удалить элемент, расположенный на *k*—м месте в массиве, можно, сдвинув весь "хвост" массива, начиная с *(k* + 1)—го элемента, на одну позицию влево, т.е. выполняя операцию $a_i := a_{i+1}$, $i = k, k+1, \ldots, N-1$:

```
N:=N-l;
For i:=k To N Do
   A[i]:=A[i+l];
```

12. *Включение элемента в заданную позицию массива:*

Перед включением элемента в *k*—ю позицию необходимо раздвинуть массив, т.е. передвинуть "хвост" массива вправо на одну позицию, начиная с *(k* + 1)-го элемента, выполняя операцию $a_{i+1} := a_i$, $i = N, N-1, \ldots, k$. Перемещения элементов массива нужно начинать с конца. В противном случае весь "хвост" будет заполнен элементом a_k. Далее k-му элементу присваивается заданное значение *B*. Размер массива увеличивается на 1:

```
For i :=N Downto k Do
   A[i+l]:=A[i];
A[k]:=B;
N:=N+1;
```

13. *Включение элемента в массив, упорядоченный по возрастанию, с сохранением упорядоченности:*

Сначала следует найти элемент, перед которым необходимо включать заданное число *B*. Для этого нужно проверять условие $B < a_i$, для всех $i = 1, 2, \ldots$. При выполнении условия текущее значение индекса *i* определяет позицию добавляемого элемента. Далее включение элемента осуществляется по алгоритму, описанному в п. 12.

```
k:=1;
While B<a[k] do
Begin
    k:=k+1;
  End;
For i :=N Dowto k-1 Do
  A[i+l]:=A[i];
A[k-1]:=B ;
```

14. *Удаление строки из матрицы:*

Требуется удалить строку с заданным номером *K*. Решение этой задачи аналогично решению задачи удаления элемента из одномерного массива. Все строки, начиная с *(K* + 1)-й, нужно переместить вверх. Число строк уменьшается на 1.

```
N:=N-1;
For i:=K To N Do
  For j:=l To M Do
```

```
B[i,j]:=B[i+l,j];
```

Удаление столбца осуществляется аналогично.

15. *Включение строки в матрицу:*

Включаемая строка задана как вектор *C*. Включение строки в матрицу аналогично включению элемента в одномерный массив (см. п.12).

```
For i :=N Downto K Do
  For j:=1 To M Do
    B[i+l,j]:=B[i,j];
For j:=1 To M Do
   B[K,j]:=C[j];
N:=N+1;
```

16. *Поиск минимального (максимального) элемента в массиве:*

Требуется найти минимальный элемент в массиве и его значение поместить в переменную *P,* а индекс — в переменную *K*.

```
P:=A[1];
K:=1 ;
For i:=2 To N  Do
   If P>A[i] Then
        Begin
          P:=A[i];
          K:=i
        End;
```

Если в массиве несколько элементов имеют минимальное значение, то в *K* будет запоминаться индекс первого из них. Если проверять условие *P*>=*A*(*i*), то будет запоминаться индекс последнего элемента. Для поиска максимального элемента нужно проверять условие *P*<*A*(*i*). Для двумерного массива алгоритм аналогичный, но нужно просматривать все элементы каждой строки, что требует организации двойного цикла, и запоминать два индекса: номер строки *K* и номер столбца *L*.

```
P:=C[l,l];
K:=l;
L:=l ;
For i:=l To N Do
   For j:=l To M Do
      If P>C[i,j] Then
         Begin
            P:=C[i,j];
           K:=i;
           L:=j
         End;
```

17. *Преобразование матрицы:*

Преобразование матрицы, как правило, производится относительно какой-либо оси, которой могут быть главная диагональ, побочная диагональ, любая строка или столбец.

Рассмотрим эти понятия на примере матрицы *A* размерами *N* ⨯ *N*, обозначив номер строки переменной *I*, а номер столбца - переменной *J*.

1. Элементы главной диагонали имеют одинаковые индексы, т.е. *A*(1,1) (например *A*(1,1), *A*(2, 2),... *A*(*N*, *N*)).

2. У всех элементов, расположенных выше (правее) главной диагонали, *J*>*I*.

3. У всех элементов, расположенных ниже главной диагонали, *J*<*I*.

4. Элементами, симметричными относительно главной диагонали, являются пары *A*(*I*, *J*) и *A*(*J*, *I*).

5. У элементов, образующих побочную диагональ, есть зависимость индексов: *I*+*J*=*N*+1, или *J*=*N*+1-*I*, т.е. *A*(*I*, *N*+1-*I*).

6. У всех элементов, расположенных выше побочной диагонали, *I*<*N*+1-*J*.

7. У всех элементов, расположенных ниже побочной диагонали, *I*>*N*+1-*J*.

8. Элементами, симметричными относительно побочной диагонали, являются пары *A*(*I*, *J*) и *A*(*N*+1-*J*, *N*+1-*I*).

9. Элементами, симметричными относительно средней горизонтальной оси, являются пары *A*(*I*, *J*) и *A*(*I*, *N*+1-*J*).

Примеры решения задач по обработке массивов

Пример 2.1. Сформировать массив *Y* из положительных элементов заданного массива *X* размером 10.

Решение.

Возможный вариант программы:

```
PROGRAM PRIM21;
  VAR
     X, Y : ARRAY [1..10] OF REAL;
      I, K : INTEGER;
     {X – исходный массив}
     {Y – сформированный массив}
     {K – количество элементов массива Y}
  BEGIN
    {Ввод массива X}
   For I:=1 To 10 Do
        Read (X[I]);
    {Формирование массива Y}
 K:=0;
    For I:=1 To 10 Do
     If X[I] > 0 Then
          BEGIN
            K:=K+1;
            Y[K]:=X[i];
          END;
```

```
    {Вывод сформированного массива Y на экран}
    For I:=1 To K Do
       Write(Y[i]:5:2);
  End.
```

Пример 2.2. В одномерный массив размером *N* ввести произвольные числа. Поменять местами элементы массива, стоящие равноудаленно от элемента с заданным индексом *K*. Вывести на экран в строку исходный и новый массивы.

Решение.

Введем произвольные числа в массив *A*(*N*).

Индекс элемента, относительно которого производится обмен, обозначим переменной *K*. При вводе *K* следует выполнить проверку, так как К не должно быть меньше единицы или больше *N*.

Алгоритм. Алгоритм заключается в том, что менять местами элементы массива можно до тех пор, пока не закончится массив слева или справа от *K*, т.е. пока индекс элемента слева больше или равен единице, а индекс элемента справа от *K* меньше или равен *N*.

Для формирования индекса элемента слева от *K* назначаем переменную *L*, а для формирования индекса элемента справа от *K* - переменную *PR*.

Вариант программы

```
PROGRAM PRIM22;
  VAR
   A : ARRAY [1.. 100] OF REAL;
   N , I, L, K,PR : INTEGER;
   X : REAL;
 BEGIN
   Write('Введите размер массива (не более 100)’);
   Readln(N);
   WHILE (N<=2) OR (N>=100) DO
      Begin
       Writeln(‘Ошибка ввода. Повторить’);
       Writeln(‘Введите размер массива’);
       Readln(N);
     End;
  FOR I:=1 TO N Do
    Begin
     Write(‘Введите ‘, I:3, ‘ число ’);
     Readln(A[I]);
     End;
     Writeln(‘Исходный массив’);
   FOR I:=1 TO N Do
     Begin
      Write(A[I]);
      If I Mod 6 = 0 Then Writeln;
```

```
  End;
 Writeln;
 Write('Введите индекс');
 Readln(K) ;
 L:=K-1;
 PR:=K+1;
 WHILE (L>=1) AND (PR<=N) Do
Begin
  X:=A[L];
  A[L]:=A[PR];
  A[PR]:=X;
  L:=L-1;
  PR:=PR+1;
 END;
Writeln;
Writeln('Новый массив');
FOR I:=1 TO N Do
  Begin
   Write(A[I]);
   If I Mod 6 = 0 Then Writeln;
  End;
END.
```

Пример 2.3. В массивы вводятся элементы двух последовательностей A_i и B_j целых чисел, которые содержат 6 и 8 элементов соответственно.
A_i — неубывающая и B_j — невозрастающая последовательности. Необходимо вывести на экран общий список значений элементов этих последовательностей по их возрастанию без создания третьего массива.

Решение.

Для решения этой задачи необходимо каждый раз перед выводом на экран очередного числа сравнивать два числа из разных массивов, пока они не закончились, и выводить на экран меньшее из них.

В соответствии с условиями задачи, массив *A* начинаем просматривать с первой ячейки, а массив *B* — с последней, т.е.

если условие A[1]<B[8] выполняется - на экран выводится A[1];
если условие A[2]<B[8] не выполняется - на экран выводится B[8];
если условие A[2]<B[7] не выполняется - на экран выводится B[7];
если условие A[2]<B[6] не выполняется - на экран выводится B[6];
если условие A[3]<B[1] не выполняется - на экран выводится B[1].

Дальнейшее сравнение чисел из двух массивов невозможно, так как один из них (массив *B*) закончился. Оставшиеся в массиве *A* числа следует вывести на экран.

Отметим, что какой из двух массивов закончится раньше, определяется числами, находящимися в нем, а не длиной массива, поэтому при составлении программы необходимо предусмотреть вывод на экран оставшихся чисел и из массива *A*, и из массива *B*.

Анализируя приведенные выше строки сравнения чисел из разных массивов, приходим к выводу:

- индексы элементов массива меняются, поэтому введем переменные для формирования индексов (*K*1 и *K*2);

- начальное значение индекса элемента в массиве *A* — *K*1 равно 1;начальное значение индекса элемента в массиве *B* — *K*2 равно 8;

- сравнивать числа из разных массивов можно до тех пор, пока один из массивов не закончится, т.е. пока *K*1<=6, а *K*2>1.

Программа

```
PROGRAM PRIM23;
  VAR
    A : ARRAY [1..6] OF INTEGER;
    B : ARRAY [1..8] OF INTEGER;
    I, K1, K2 : INTEGER;
  BEGIN
   Writeln('Введите массив A');
   FOR I:=1 TO 6 Do
     Begin
      Write('Введите ', I:1, ' число массива A');
      Readln(A[I]);
     End;
   Writeln('Введите массив B');
   FOR I:=1 TO 8 Do
     Begin
      Write('Введите ', I:1, ' число массива B');
      Readln(B[I]);
     End;
   Writeln('Общий список чисел по возрастанию');
   K1:=1;
   K2:=8;
   WHILE (K1<=6) AND (K2>=1) Do
     IF A[K1]<B[K2] THEN
       Begin
        Write(A[K1]:4);
        K1:=K1+1
       End
                ELSE
       Begin
        Write(B[K2]:4);
        K2:=K2-1;
       END;
{Из двух циклов For в зависимости от чисел, введенных в массивы, будет работать какой-либо один из них}
     FOR I:=K1 TO 6 Do
       Write(A[I]:4) ;
```

```
  FOR I:=K2 DownTo 1 Do
    Write(B[I]:4) ;
  END.
```

Пример 2.4. В одномерный массив ввести *N* произвольных чисел. Задан индекс *K* одного из элементов массива. Требуется записать в обратном порядке все элементы, стоящие слева и справа от заданного *K*. Вывести на экран в строку новый массив.

Решение.

Предположим, что исходный массив (*A*(*N*)) заполнен произвольно, а *K* = 5.

Алгоритм.

Анализируя расположение чисел в новом массиве, приходим к выводу, что для решения данной задачи следует выполнить обмен элементов отдельно слева и справа от элемента с индексом *K*.

Слева должны меняться местами элементы: *A*[1] с *A*[4]; *A*[2] с *A*[3]. Справа должны меняться элементы: *A*[6] с *A*[14]; *A*[7] с *A*[13]; *A*[8] с *A*[12]; *A*[9] с *A*[11].

При обмене индексы элементов меняются, поэтому для их формирования выбираем переменные *L* и *PR*.

Для реализации предложенного алгоритма необходимо организовать два цикла (обмен слева от *K* и обмен справа от *K*), пока индекс левой переменной меньше индекса правой переменной (*L*<*PR*).

Программа

```
PROGRAM PRIM24;
  LABEL 10;
  VAR
   A : ARRAY [1..100] OF REAL;
   I, N, K, L, PR : INTEGER;
   X : REAL;
  BEGIN
  Write('Введите размер массива (не более 100)');
   10:Readln(N);
   IF (N<=2) OR (N>=100) THEN
     Begin
      Writeln('Ошибка ввода. Повторить');
      GOTO 10;
    End;
  FOR I:=1 TO N Do
    Begin
     Write('Введите ', I:3, ' число ');
     Readln(A[I]);
     End;
     Writeln('Исходный массив');
   FOR I:=1 TO N Do
     Begin
      Write(A[I]);
      If I Mod 6 = 0 Then Writeln;
```

```
  End;
 Write(‘Введите индекс К ’);
 Readln(К);
 {Цикл обмена слева от  К }
 L:=1;
 PR:=K-1;
 WHILE L<PR Do
  Begin
    X:=A[L];
    A[L]:=A[PR];
    A[PR]:=X;
    L:=L+1;
    PR:=PR-1;
  End;
 {Цикл обмена справа от К }
 L:=K+1;
 PR:=N;
 WHILE L<PR Do
  Begin
    X:=A[L];
    A[L]:=A[PR];
    A[PR]:=X;
    L:=L+1;
    PR:=PR-1;
  End;
 Writeln(‘Новый массив');
 FOR I:=1 TO N Do
Begin
  Write(A[I]);
  If I Mod 6 = 0 Then Writeln;
  End;
END.
```

Пример 2.5. Ввести два одномерных массива (первый из *N* целых чисел, второй — из 5 различных целых чисел). Удалить из первого массива числа, содержащиеся во втором. При удалении элементов первого массива он должен быть сжат перемещением оставшихся элементов в массиве.

Если какое-либо число из второго массива не встретилось в первом ни разу, вывести соответствующее сообщение. Вывести измененный массив.

Решение.

Для решения задачи следует по очереди каждое число из массива *B* сравнивать на совпадение с каждым числом массива *A*.

При совпадении осуществляется перемещение справа налево всех чисел, стоящих правее найденного. Например, если найдено число в ячейке *A*[2], то осуществляется последовательное перемещение:

$A[2] \leftarrow A[3]$, $A[3] \leftarrow A[4]$, $A[4] \leftarrow A[5]$ ... $A[X\text{-}1] \leftarrow A[X]$,

где X — переменная, определяющая меняющуюся длину сжимаемого» массива.

Каждый раз после удаления очередного числа массив A укорачивается и для просмотра, и для последующих перемещений элементов, хотя заданная длина массива A не меняется.

Программа

```
PROGRAM PRIM25;
  LABEL 10, 20;
  VAR
    A : ARRAY [1.. 100] OF INTEGER;
    B : ARRAY [1.. 5] OF INTEGER;
    N, I, J, X, F, K : INTEGER;
  BEGIN
   { Ввод размера и элементов массива A }
     Write('Введите количество чисел массива A');
   Readln(N);
   FOR I:=1 TO N Do
     Begin
       Write('Введите ', I:3, ' число массива A');
       Readln(A[I]);
     End;
   { Ввод элементов массива B }
FOR I:=1 TO 5 Do
     Begin
       Write('Введите ', I:3, ' число массива B');
       Readln(B[I]);
  End;
{первоначальная длина массива A заносится в переменную X}
X:=N;
{внешний цикл (по i) для выбора элементов массива B}
FOR I:=1 TO 5 Do
     Begin
{каждое новое число массива B начинаем искать с первой ячейки массива A (K=1), переменная F показывает, встретилось ли очередное число массива B хотя бы раз в массиве A}
        K:=1;
        F:=0;
{цикл поиска очередного числа массива B в массиве A, перемещение чисел массива A справа налево, изменение длины массива A  (X=X-1) и изменение значения переменной F, которое фиксирует, что очередное число массива B встретилось в массиве A}
10:  WHILE K<=X Do
       Begin
         IF B[I]<>A[K] THEN
```

```
            Begin
             K:=K+1;
             GOTO 10
            End;
          FOR J:=K TO X-1 Do
        A[J]:=A[J+1];
      X:=X-1;
      F:=1
     End;
   IF F=0 THEN
     Writeln('Число', B [J],'не встретилось в массиве A');
  End;
IF X=0 THEN
  Begin
    Writeln('Массив A состоял только из чисел массива B, поэтому он полностью сжался');
    GOTO 20;
  End;
{вывод на экран оставшихся чисел массива A в случае, если какие-то числа остались}
Writeln('Измененный массив A');
FOR I:=1 TO X Do
  Write(A[I]:4);
20: END.
```

Пример 2.6. В одномерный массив ввести *N* произвольных чисел (*N* - заданное число). Переместить его элементы таким образом, чтобы в конце массива были все отрицательные числа, сохранив при этом начальный порядок следования отдельно для отрицательных и отдельно для нулевых и положительных элементов. Дополнительный массив не использовать.

Решение.

Возможный вариант программы:

```
PROGRAM PRIM26;
  VAR
    A : ARRAY [1..50] OF REAL;
    N, I, P : INTEGER;
    X : REAL;
 BEGIN
   Write('N= ');
   Readln( N);
   FOR I:=1 TO N Do
     Begin
       Write('Номер ', I);
       Readln( A[I]);
     End;
```

```
 Writeln('Исходный массив');
 FOR I:=1 TO N  Do
   Write( A[I]:6:2);
P:=1;
WHILE P<>0 Do
  Begin
    P:=0;
   FOR I:=1 TO N-1 Do
      IF (A[I]<0) AND (A[I+1]>=0) THEN
         Begin
           X:=A[I];
           A[I]:=A[I+1];
           A[I+1]:=X;
           P:=1
         End;
  End;
Writeln('Модифицированный массив');
FOR I:=1 TO N Do
  Write( A[I]:6:2);
END.
```

Пример 2.7. Сформировать массив из индексов минимальных элементов строк матрицы A размерами 5 ⨯ 4.

Решение.

Возможный вариант программы*:*

```
PROGRAM PRIM27;
      VAR
         A : ARRAY [1..5,1..4] OF REAL;
         X : ARRAY [1..5] OF INTEGER;
         I, J, K : INTEGER;
         MIN : REAL;
       BEGIN
        For I:=1 To 5 Do
       For J:=1 To 4 Do
           Read( A[i,j]);
       For I:=l To 5 Do
       Begin
        MIN:=A[i,1];
        K:=l;
         For J:=2 To 4 Do
       If A[i,j]<MIN Then
          Begin
            MIN:=A[i,j];
            K:=J;
```

```
            End;
          X[i]:=K;
        End;
      For I:=l To 5 Do
         Write(X[i]:4);
   End.
```

Пример 2.8. Задана матрица A размерами 5 ⨯ 5. Найти максимальный по модулю элемент матрицы. Строку, содержащую этот элемент, поменять местами с первой.

Решение.

Возможный вариант программы:

```
Program PRIM28;
VAR
   A : ARRAY [1..5,1..5] of REAL;
    I, J, K, L : INTEGER;
    MAX, Q : REAL;
BEGIN
{Ввод данных }
 For I:=1 To 5 Do
  For J:=l To 5 Do
 Read (A[I,J]);
{Отыскание максимального по модулю элемента в матрице }
MAX:=Abs(A[l,l]);
K:=l;
L:=l ;
For I:=l To 5 Do
  For J:=1 To 5 Do
   Begin
     Q:=Abs(A[I,J]) ;
     If P < Q Then
           Begin
            MAX:=Q;
                K:=I;
                L:=J;
    End;
   End;
WRITELN('Максимальный по модулю элемент =',A[K,L]:7:2);
WRITELN('находится в строке',K:2,' в столбце',L:2);
{Перестановка строк}
For I:=1 To 5 Do
  Begin
    Q:=A[K,I];
    A[K,I]:=A[l,I];
    A[l,I]:=Q ;
```

```
  End;
{Печать преобразованной матрицы}
WRITELN('Матрица-результат');
For I:=l To5 Do
  Begin
   For J:=1 To 5 Do
     WRITE(A[I,J]:7:2);
   WRITELN;
 End;
End.
```

Пример 2.9. Подсчитать количество отрицательных элементов в каждой строке матрицы *X* размерами 5×4. Результат получить в виде вектора.

Решение.

Программа

```
PROGRAM PRIM29;
 VAR
   X : ARRAY [1..5,1..4] OF REAL;
   P : ARRAY [1..5] OF INTEGER;
   I, J, N : INTEGER;
   {X – заданная матрица}
   {P – одномерный массив, значением элементов которого является количество отрицательных элементов в соответствующей строке матрицы X}
   {N – количество отрицательных элементов в строке матрицы X}
 BEGIN
   {Ввод матрицы X}
   For I:=1 To 5 Do
  For J:=1 To 4 Do
   Read (X[I,J]);
   For I:=1 To 5 Do
     Begin
          {Блок определения количества отрицательных элементов в I-й строке матрицы}
    N:=0;
    For J:=l To 4 Do
      If X[I,J] < 0 Then N:=N+1;
    P[I]:=N ;
End;
   {Вывод}
  For I:=l To 5 Do
    WRITELN('B ' ,I:2,' строке ', P[I]:2,' отрицательных элементов');
End.
```

Пример 2.10. В каждой строке матрицы удалить минимальный элемент строки.

Решение.
Программа

```
PROGRAM PRIM210;
 VAR
   A : ARRAY [1..5,1..4] OF REAL;
    I, J, K : INTEGER;
   MIN : REAL;
     BEGIN
        {Ввод матрицы A}
   For I:=l To 5 Do
   For J:=l To 4 Do
     Read (A[I,J]);
For I:=l To 5  Do
   Begin
  {Определение минимального элемента в I-й строке и его индекса К}
    MIN:=A[I,1];
    K:=l ;
    For j=2 To 4 Do
      If A[I,J] < MIN Then
           Begin
            MIN:=A[I,J];
            K := J
           End;
   {Удаление минимального элемента из I-й строки}
   If K < 4 Then
     For J:=k+l To 4 Do
       A[I,J-l]:=A[I,J];
  End;
{Вывод на экран преобразованной матрицы}
For I:=1 To 5 Do
  Begin
   For J:=1 To 3 Do
     WRITE(A[I,J]:5:2);    WRITELN;
    End;
  End.
```

Контрольные вопросы

1. Что такое массив?
2. Как осуществляется доступ к элементам массива?
3. Как осуществляется описание массивов на языке Турбо Паскаль?
4. Как выполняется ввод вектора, матрицы?
5. Как организовать вывод массива?
6. Как осуществить суммирование векторов и матриц?

7. Как выглядит алгоритм транспонирования матрицы?
8. Как выполнить умножение: а) матрицы на матрицу, б) матрицы на вектор?
9. Как выполнить удаление элементов массива?
10. Как добавить элементы в массив?
11. Как осуществить замены (перемещения) элементов внутри массива?

Задачи I уровня

Эти задачи предназначены для приобретения навыков работы с одномерными и двумерными массивами с использованием типовых алгоритмов.

1. Найти среднее значение элементов заданного массива размером 8. Преобразовать исходный массив, вычитая из каждого элемента среднее значение.

2. Для заданного массива размером 10 определить индекс элемента, значение которого наиболее близко к среднему значению элементов массива.

3. Решить уравнение $ax = b$ для семи пар значений a и b, заданных в двух массивах. Результат поместить в массив X.

4. Найти минимальный среди положительных элементов заданного массива размером 10.

5. Вычислить сумму элементов одномерного массива, расположенных до первого отрицательного элемента.

6. Все положительные элементы массива увеличить в 2 раза, оставив остальные без изменения.

7. Поменять местами максимальный и первый отрицательный элементы массива размером 10.

8. Поменять местами минимальный и максимальный элементы массива размером 10.

9. Вычислить среднее значение элементов массива, предварительно отбросив минимальный и максимальный элементы.

10. Задана матрица размерами $n \times m$. Просуммировать положительные элементы каждой строки. Результат получить в виде вектора размером n.

11. Для матрицы размерами $n \times m$ найти максимальный элемент каждой строки. Результат получить в виде вектора размером n.

12. В квадратной матрице размерами 5×5 переставить местами строку с максимальным элементом на главной диагонали и строку с заданным номером.

13. В квадратной матрице размерами 5×5 исключить строку и столбец, на пересечении которых расположен максимальный элемент главной диагонали.

14. Задана матрица размерами $n \times n$ и число K ($1 \le K \le n$). Найти максимальный по модулю элемент в K-м столбце; строку, содержащую этот элемент, переставить с K-й строкой.

15. Задана матрица размерами $n \times m$. Определить номера строк, содержащих минимальный и максимальный элементы и поменять их местами.

16. Задана матрица размерами $n \times n$. Сменить знаки у элементов, лежащих ниже главной диагонали.

17. Ввести массив чисел размерами $M \times N$ (M и N - заданные числа). "Особым"

элементом массива назовем элемент, который является наибольшим в столбце и одновременно наибольшим в строке. Определить количество "особых" элементов в введенном массиве, считая, что в каждой строке (и в каждом столбце) присутствует только один наибольший элемент.

18. *N* ребят образуют круг. Начиная отсчет по кругу от первого, в ходе считалки удаляется каждый *K*-й. После очередного удаления круг смыкается, а считалка начинается вновь с участника, следующего в круге за удаленным. Напечатать номера ребят в порядке их удаления из круга и номер единственного оставшегося.

19. Ввести массив чисел размерами *M* ⨯ *N* (*M* и *N* - заданные числа).Сменить знаки у элементов, лежащих выше главной диагонали и имеющих четную сумму индексов.

20. Ввести массив чисел размерами *M* ⨯ *N* (*M* и *N* - заданные числа). В элементах столбцов, не содержащих отрицательных элементов, сменять знаки.

Задачи II уровня

1. Дана целочисленная квадратная матрица *A* размерами *N*⨯*N*, где *N* - заданное натуральное число. Столбец с индексом *J* (*J*=1,2,..,N) назовем отмеченным, если все элементы в этом столбце, расположенные на главной диагонали и ниже нее являются простыми числами и оканчиваются на цифру 7. Найти количество отмеченных столбцов в матрице *A*.

2. Ввести по 18 чисел в два одномерных массива. Переписать элементы массивов построчно в квадратную матрицу, расположив их по убыванию значений. Дополнительных массивов не использовать. Напечатать матрицу.

3. Ввести числовой двумерный массив размерами *N* ⨯ *N* (*N* - заданное число от 6 до 20). Переставить элементы массива симметрично относительно побочной диагонали. Полученный массив вывести на экран. Побочная диагональ проходит через правый верхний и левый нижний углы таблицы.

4. Ввести числовую прямоугольную матрицу размерами *M* ⨯ *N* (*M* и *N* заданы). Определить элементы, которые, являясь максимальными в столбцах, больше всех своих соседей слева в строке и меньше всех своих соседей справа в строке, указав значения найденных элементов и их индексы, или же сообщить, что таких элементов нет. Крайние столбцы не рассматривать.

5. В одномерный массив ввести *N* произвольных чисел (*N* - заданное число). Переместить его элементы таким образом, чтобы в конце массива были все отрицательные числа, сохранив при этом начальный порядок следования отдельно для отрицательных и отдельно для нулевых и положительных элементов. Дополнительный массив не использовать.

6. Ввести в массив 10 произвольных чисел. Выполнить перемещение чисел в массиве таким образом, чтобы в начале массива оказались все положительные числа, затем - все отрицательные, а в конце массива - нули, сохранив при этом взаимное расположение в массиве ненулевых чисел каждой из двух групп. Дополнительных массивов не использовать.

7. Двигаясь по спирали от центра по периметру введенной числовой квадратной матрицы, состоящей из *N* строк и столбцов (*N* - заданное число), обойти все элементы матрицы и распечатать их в порядке обхода.

8. На основе введенной числовой квадратной таблицы, состоящей из *N* строк и столбцов (*N* - заданное число), сформировать и напечатать массив, *i*-й элемент

которого равен 1 , если *i*-я строка матрицы образует строго возрастающую последовательность. В противном случае *i*-й элемент должен быть равен 0.

9. В введенной числовой таблице, имеющей *N* строк и *M* столбцов (*M* и *N* - заданные числа), переставить строки по возрастанию суммы положительных элементов в них.

10. В введенной числовой квадратной таблице, состоящей из *N* строк и столбцов (*N*-заданное число), определить такие *K* и *L*, для которых *K*-я строка совпадает с *L*-м столбцом (попарно совпадают все элементы строки и столбца), а также общее количество таких совпадений.

11. Переставить в конец одномерного массива, состоящего из *N* элементов, элементы, кратные заданному числу, и расположить их в порядке возрастания.

12. Нулевые элементы переставить в начало одномерного массива, а остальные расположить в порядке возрастания.

13. В введенной числовой матрице, состоящей из *M* строк и *N* столбцов, элементы строки, содержащей максимальное количество положительных элементов, расположить в порядке возрастания.

14. В введенной числовой матрице, состоящей из *M* строк и *N* столбцов, элементы всех строк расположить в порядке убывания.

15. Ввести целое число *N*. Сформировать и вывести на экран целочисленную квадратную матрицу (двухмерный массив) размерами *N* ⨯ *N*.

1	2	3	...	*N*-2	*N*-1	*N*
2	3	4	...	*N*-1	*N*	0
3	4	5	...	*N*	0	0
.	.	.	.	.	.	.
N-1	*N*	0	...	0	0	0
N	0	0	...	0	0	0

16. Ввести целое число *N*. Сформировать и вывести на экран целочисленную квадратную матрицу (двухмерный массив) размерами *N* ⨯ *N*.

N	0	0	...	0	0
N	*N*-1	0	...	0	0
N	*N*-1	*N*-2	...	0	0
.	.	.	.	.	.
N	*N*-1	*N*-2	...	2	0
N	*N*-1	*N*-2	...	2	1

17. В введенной числовой матрице, состоящей из *M* строк и *N* столбцов, элементы столбцов, содержащих хотя бы один отрицательный элемент, расположить в порядке убывания.

18. В введенной числовой матрице, состоящей из *M* строк и *N* столбцов, переставить строки в порядке убывания количества содержащихся в них положительных элементов.

19. В введенной числовой матрице, состоящей из *M* строк и *N* столбцов, переставить строки в порядке возрастания элементов последнего столбца.

20. В введенной числовой матрице, состоящей из *M* строк и *N* столбцов, строки матрицы, не содержащие нулевых элементов, расположить в порядке убывания произведений элементов строк**.**

21. В введенной числовой матрице, состоящей из *M* строк и *N* столбцов,

элементы столбца, содержащего максимальное количество нулевых элементов, расположить в порядке убывания.

22. В введенной целочисленной матрице, состоящей из *M* строк и *N* столбцов, удалить строку и столбец, содержащие минимальный из наиболее часто встречающихся элементов.

23. Замените каждый элемент матрицы целых чисел размерами $M \times N$ на сумму элементов его "креста", т.е. тех элементов, которые находятся в одном с ним столбце и в одной строке.

24. Вычислить сумму элементов одномерного массива размером *N*, являющихся простыми числами. Простыми числами называются числа, которые не делятся нацело ни на какое число кроме 1 и самого себя, при этом 1 простым числом не является.

2.2. Строки

С помощью компьютера часто приходится обрабатывать не только числа, но и строки символов, например имена, фамилии, названия и т.д. Поэтому в Турбо Паскале имеется специальный тип String, облегчающий программирование операций над строками.

Значение стандартного типа String – последовательность символов длиной от 1 до 255.

Данные типа String, как и числовые данные, подразделяются на константы и переменные.

Строковые константы – это последовательность символов, заключенная в апострофы. Мы уже использовали строковые константы при вводе и выводе информации, например 'Введите размер массива', 'Сумма элементов матрицы A ='.

Строковые константы могут быть описаны в разделе описания констант:

```
Const C = 'Результаты вычислений';
      St = 'Строка';
      January: String[10] = 'Январь'; {Типизированная константа}
```

Описание строковых переменных имеет вид

```
Var  имя переменной : String[N];
```

Здесь *N* – целая константа или имя целой константы ($1 \le N \le 255$), указывающая максимальную длину строки (количество символов).

Разрешается не указывать *N*, в этом случае длина строки принимается максимально возможной, а именно равной 255. Например:

```
Var FIO : String[25];
    Nazv : String;
```

Как и в одномерных массивах, к отдельным символам строки можно обратиться с помощью индексов в квадратных скобках: FIO[5], Nazv[8].

Над строковыми данными определены операции: присваивание, сцепление и отношение.

Присваивание последовательности символов строковым переменным осуществляется с помощью оператора присваивания.

Ввод и вывод значений строковых переменных осуществляется без апострофов с помощью стандартных процедур Read, Readln, Write, Writeln.

Если при присваивании или вводе длина строки превышает максимальную длину строковой переменной, то все лишние символы справа отбрасываются.

Операция сцепление применяется для объединения нескольких строк в одну. Для обозначения операции сцепления в Турбо Паскале используется символ ‘+’.

Например:

```
  Var
     St, St1, St2 : String;
Begin
     St1:=’ Турбо ‘;
    St2:=’Паскаль’;
    St:=St1+St2;
    Writeln(‘St=’,St);
End.
```

После выполнения этой программы получим St= Турбо Паскаль.

Если длина объединенной строки превысит максимально допустимую длину *N*, то лишние символы справа отбрасываются.

Произвольные пары строк могут сравниваться с помощью операций отношения: =, <>, <, >, <=, >= .

Приоритет каждого из операторов отношения ниже приоритета оператора сцепления.

Сравнение строк производится посимвольно слева направо до первого несовпадающего символа, и та строка считается большей, в которой первый несовпадающий символ имеет больший ASCII-код. Если такая пара не найдена, а строки совпадают до последнего символа более короткой строки, то короткая строка рассматривается как меньшая.

Две строки равны только тогда, когда они одинаковой длины и состоят из идентичных символов.

Все остальные действия над строками реализуются с помощью стандартных процедур и функций:

LENGTH (St) - функция, возвращающая текущую длину строки St.

COPY(St, I, COUNT) – функция, которая выделяет из строки St подстроку длиной COUNT, начиная с позиции I. Если I>Length(St), то результатом будет пробел.

DELETE(St, I, COUNT) – процедура, которая удаляет из строки St подстроку длиной COUNT, начинающуюся с позиции I. Результатом является новая строка St без удаленной подстроки. Параметр St может быть только переменной.

INSERT(St1, St2, I) – процедура, которая вставляет строку ST1 в строку St2, начиная с позиции I. Параметр St2 может быть только переменной.

POS(St1, St2) – функция, которая производит поиск подстроки St1 в строке St2 и возвращает номер символа в St2, с

которого начинается подстрока St1. Если подстрока St1 не входит в строку St2, то результат равен нулю.

STR(X, St) – процедура, которая преобразует выражение X целого или вещественного типа в строку St.

VAL(St, X, CODE) – процедура, преобразующая строку символов St, представляющую число, в значение целой или вещественной переменной X, которое определяется типом этой переменной. Параметр CODE равен нулю, если преобразование прошло успешно, в противном случае он содержит номер позиции в строке St, где обнаружен ошибочный символ.

Примеры решения задач по обработке символьных данных

Пример 2.11. Ввести строку и подсчитать количество слов в ней. Слова разделяются пробелами.

Решение.

Обозначим: St – исходная строка; L – длина исходной строки; K – количество слов в строке.

Для подсчета количества слов необходимо организовать цикл. Так как после слова может стоять не один пробел, а несколько, то для определения слова используется условие (St[I]<>' ') AND (St[I+1]=' ').

Для того чтобы при подсчете количества слов учесть последнее слово, в конец введенной строки добавляется пробел: St:=St + ' ' .

Программа

```
Program Prim211;
 Var
   St       : String;
   I, K, L  : Integer;
 Begin
  Writeln('Введите строку');
  Readln(St);
  L := Length(St);
  St:=St + ' ';
  K := 0;
  For I:=1 To L Do
   If (St[I]<>' ') AND (St[I+1]=' ') Then K:=K+1;
  Writeln('Количество слов = ', K);
 End.
```

Пример 2.12. Ввести текст длиной до 255 символов в виде строки. Если в строке четное число символов, удалить из нее все последующие вхождения первого символа, в противном случае - удалить символ, расположенный в середине строки. Напечатать новый текст.

Решение.

Обозначим: St - исходная строка; St1- первый символ строки St; L – длина исходной строки St.

Алгоритм.

Ход выполнения алгоритма зависит от длины строки.

В первом случае необходимо организовать цикл с условием, так как после удаления символов длина строки будет меняться. В этом цикле все символы строки сравниваются с первым символом строки (St[I] = St1) и при равенстве удаляются из строки St, причем в случае удаления символа параметр цикла I не меняется, так как символы, соответствующей первому, могут следовать подряд друг за другом.

Во втором случае необходимо из строки St удалить символ с номером (L div 2) +1.

Программа

```
Program Prim212;
Var
  St : String;
  St1: Char;
  I,L  : Integer;
Begin
  Writeln('Введите строку');
  Readln(St);
  L:=Length(St);
  St1:=St[1];
  If (L Mod 2)=0
    Then
      Begin
        I:=1;
        While I<= Length(St) Do
          If St[I]=St1 Then Delete(St,I,1)
                       Else     I:=I+1;
      End
    Else
      Delete(St,(L Div 2)+1,1);
  Writeln('Новая строка');
  Writeln(St);
End.
```

Пример 2.13. Ввести текст длиной до 255 символов. В тексте все знаки плюс, непосредственно за которыми следует цифра, заменить пробелом. Напечатать новый текст.

Решение.

Необходимо организовать цикл просмотра всех пар соседних символов. Если в очередной паре символ St[I] = ‘+’, а второй символ - цифра (т.е.’0’<=St[I+1]<=’9’), то из строки St удаляется I-й символ и вставляется символ «пробел».

Программа

```
Program Prim213;
```

```
Var
  St : String;
  I  : Integer;
Begin
  Writeln('Введите строку');
  Readln(St);
  For I:=1 To Length(St) Do
   If (St[I]='+')And(St[I+1]>='0')And(St[I+1]<='9') Then
     Begin
       Delete(St,I,1);
       Insert(' ',St,I);
     End;
  Writeln('Новая строка');
  Writeln(St);
End.
```

Рассмотрим другой вариант программы этой задачи, в котором используется тип данных множество.

```
Program Prim213a;
Const M=[‘0’..’9’]; {константа типа множество}
Var
  St : String;
  I  : Integer;
 Begin
  Writeln('Введите строку');
  Readln(St);
  For I:=1 To Length(St) Do
   If (St[I]='+')And(St[I+1] In M) Then
     Begin
       Delete(St,I,1);
       Insert(' ',St,I);
     End;
  Writeln('Новая строка');
  Writeln(St);
End.
```

Пример 2.14. Ввести предложение длиной до 255 символов в виде строки, а затем вывести на экран сообщение о том, сколько слов в этом предложении содержат ровно три буквы "и". В качестве разделителя слов в предложении используются символы пробела.

Решение.

Возможный вариант программы.

```
Program Prim214;
 Var
    St : String;
    I, K3i, Ki : Integer;
```

```
  Begin
  Writeln('Введите строку');
    Readln(St);
    K3i := 0; {количество слов с тремя буквами “и”}
    Ki := 0; { количество букв и в слове}
    St := St + ‘ ‘; {добавляется пробел в конец строки}
    For I:=1 To Length(St) Do
   If St[I]=’ ‘
      Then
        Begin
          If  Ki=3 Then
            K3i:=K3i+1;
          Ki:=0
        End
      Else
        If St[I] = ‘и’ Then Ki:=Ki+1;
  If K3i = 0 Then Writeln(‘Слов с тремя “и” нет ‘)
              Else Writeln(‘В строке содержится ‘,K3i,’ слов с тремя буквами “и” ’);
End.
```

Пример 2.15. Ввести текст длиной до 254 символов в виде строки, а затем вывести его на экран, удалив из него лишние пробелы, т.е. из нескольких подряд идущих пробелов оставить только один. Выдать сообщение о количестве удаленных пробелов.

Решение.

Возможный вариант программы.

```
Program Prim215;
Var
  St : String;
  I, L : Byte;
Begin
  Writeln('Введите строку');
  Readln(St);
  L:= Length(St);
  I := 1;
  While I < Length(St) Do
    If (St[I] = ' ') And (St[I+1] = ' ') Then Delete(St,I,1)
                                         Else I:=I+1;
  Writeln('Новая строка');
  Writeln(St);
  Writeln('Количество удаленных пробелов =',L-Length(St));
End.
```

Пример 2.16. Во введенном тексте заменить заданную подстроку символов на другую заданную подстроку (длины двух заданных подстрок могут не совпадать).

Подсчитать количество произведенных замен.

Решение.

Обозначения: St – заданная строка; NSt – новая строка; St1 – удаляемая подстрока, St2 – вставляемая подстрока; L1 – длина удаляемой подстроки; N – номер позиции, с которой начинается удаляемая подстрока St1 в строке St; K – количество замен.

Алгоритм.

Необходимо организовать цикл, в котором формировалась бы новая строка и подсчитывалось количество произведенных вставок. Так как заранее неизвестно количество вставок, этот цикл должен быть итерационным (например цикл с предусловием While). В качестве условия окончания цикла предлагается использовать значение функции Pos, которое равно нулю, если подстроки St1 нет в строке St.

Цикл WHILE требует подготовки, т. е. номер позиции первого вхождения (N:=Pos(St1,St)) подстроки St1 в строку St должен быть введен до цикла. Номера позиций следующих вхождений определяются в цикле.

На каждом шаге цикла к строке NSt добавляется подстрока, состоящая из первых (N-1) символов строки St и символов вставляемой подстроки St2, и из строки St удаляются все символы, предшествующие подстроке St1, вместе с символами этой подстроки. Здесь N – номер позиции, с которой начинается подстрока St1 в строке St.

Возможный вариант программы.

```
Program Prim216;
Var
  St, St1, St2, NSt : String;
  L1, N, K : Byte;
Begin
  Writeln('Введите строку');
  Readln(St);
  Writeln('Введите удаляемую строку');
  Readln(St1);
  Writeln('Введите вставляемую строку');
  Readln(St2);
  L1:=Length(St1);
  N:=Pos(St1,St);
  Nst:='';
  While N<>0 Do
    Begin
      K:=K+1;
      NSt:=NSt + Copy(St,1,N-1)+St2;
      Delete(St,1,N-1+L1);
      N:=Pos(St1,St)
    End;
  Writeln('Новая строка');
  Writeln(NSt);
```

```
  Writeln('Количество произведенных замен',K);
End.
```

Контрольные вопросы

1. Что представляет собой значение типа String?
2. Относится ли тип String к стандартным типам?
3. Как описываются переменные строкового типа?
4. Какова максимальная длина результирующей строки при выполнении операций над строками?
5. Какие операции можно выполнять над строковыми данными?
6. Как выполняется операция сравнения строк?
7. Какие процедуры и функции используются для обработки строковых данных?

Задачи I уровня

Эти задания предназначены для приобретения навыков работы со строками.

1. Удалить из текста все цифры, подсчитав количество удаленных символов.
2. Проверить, имеется ли в заданном тексте баланс открывающихся и закрывающихся скобок.
3. Подсчитать, сколько имеется в тексте пар одинаковых букв.
4. Поменять в тексте квадратные скобки на круглые. Подсчитать количество замен.
5. Удалить в тексте символ " и подсчитать количество оставшихся символов.
6. Из текста выбрать числа и записать их в массив X.
7. Определить, является ли текст “палиндроном”, т.е. читается ли он одинаково без учета пробелов в обоих направлениях.
8. Отредактировать текст, заменяя многоточия точкой. Найти длину измененного текста
9. В заданном тексте подсчитать количество русских букв, обозначающих гласные звуки.
10. Из заданного текста выбрать и напечатать только те символы, которые встречаются в нем один раз.
11. В заданном выражении поменять местами знаки умножения и деления.
12. Подсчитать в тексте количество знаков препинания.
13. Подсчитать в выражении количество арифметических операций.
14. Поменять в тексте круглые скобки на квадратные.
15. Проверить баланс открывающихся и закрывающихся круглых скобок.
16. Распечатать строку с наибольшим количеством букв, обозначающих гласные звуки.
17. Заменить в выражениях знаки строгого отношения ($<$, $>$) на знаки нестрогого отношения ($\leq$, $\geq$).
18. Записать в массив Z, состоящий из четырех элементов, количество знаков сложения, вычитания, умножения и деления в заданном выражении.

19. Заключить в скобки цифры в заданном тексте.

20. Подсчитать в тексте количество запятых. Поставить в конце текста многоточие.

21. Подсчитать в тексте количество слов, начинающихся с букв, обозначающих гласные звуки.

22. Создать массив русских букв, состоящий из 10 элементов. Подсчитать в заданном тексте количество букв, входящих в созданный массив.

23. Создать массив цифр и ввести арифметическое выражение в виде строки. Подсчитать в нем количество цифр, входящих в созданный массив.

24. Найти сумму всех чисел, используемых в тексте.

25. Ввести текст в виде символьной строки. Определить, сколько раз в этом тексте встретился каждый из знаков препинания: запятая, точка, тире, двоеточие.

Задачи II уровня

1. Ввести текст в виде строки. Составить из него новый текст, в котором присутствуют символы старого текста, но каждый символ встречается только один раз.

2. Определить, правильно ли в тексте расставлены круглые скобки, т.е. находится ли справа от каждой открывающейся скобки соответствующая ей закрывающаяся скобка, а слева от каждой закрывающейся – соответствующая ей открывающаяся.

3. Составить новый текст, исключив из старого символы, находящиеся внутри круглых скобок (скобки так же удалить). Подсчитать количество удаленных символов. Предполагается, что внутри каждой пары скобок других скобок нет.

4. Во введенном в символьную переменную тексте переместить первые десять символов, расположив их после первой встреченной запятой. Если данный перенос возможен, вывести на экран измененный текст, а если он невозможен, вывести соответствующее сообщение.

5. Во введенном в символьную переменную тексте зашифровать каждое слово путем переписывания символов в порядке, обратном их следованию в слове. Слова текста разделены одним пробелом. Вывести на экран преобразованный текст.

6. Ввести текст в виде символьной строки, в которой слова разделены пробелами. Напечатать слова текста, начинающиеся с буквы "А".

7. Во введенном в символьную переменную тексте поменять местами первое и последнее слово. Длина текста не превышает 254 символа, слова текста разделены одним пробелом.

8. Во введенном в символьную переменную тексте, слова которого разделены одним и более пробелами, заменить окончание ING каждого слова, встречающегося в заданном предложении, на окончание ED. Вывести на экран исходный и отредактированный тексты.

9. Ввести текст в виде символьной строки, в котором слова разделены пробелами. Напечатать слова текста, заканчивающиеся буквой "А".

10. В массив размерности N (N – заданное число) вводятся слова. Вывести на экран эти слова в порядке их длины, а также определить количество самых длинных

слов в массиве.

11. Определите число слов, заканчивающихся и начинающихся одной и той же буквой, в строке, введенной пользователем. Слова разделяются символом пробел.

12. Сформировать новую строку на основе строки, введенной пользователем, переставляя слова в обратном порядке (т.е. первое слово будет последним, второе – предпоследним и т.д.).

13. Дана строка цифр целого двоичного числа длиной не более 17 символов. Необходимо сформировать и напечатать строку цифр этого числа, представленного в восьмеричной системе счисления. Учесть, что двоичная строка может иметь знаковый символ.

14. В массив размерности *N* (*N* - заданное число) ввести слова длиной не более 20 символов каждое. Вывести эти слова в порядке увеличения их длины с указанием количества букв “а” в каждом из них.

15. Задана строка символов, состоящая из слов, где слово — это любая последовательность символов, не содержащая внутри себя пробелов и разделенная с одной или с обеих сторон одним или несколькими пробелами. Распечатать самое короткое слово, в котором нет буквы "A". Если таких слов нет, то выдать соответствующее текстовое сообщение.

2.3. Записи

Запись – это структура данных, состоящая из фиксированного числа элементов, которые называются полями записи.

В отличие от массивов данные в разных полях могут иметь разные типы.

При описании записи задаются имя и тип каждого поля. Описание записи начинается со служебного слова RECORD и заканчивается служебным словом END. В качестве примера опишем тип Person для хранения информации о человеке и переменные Student и Complex типа запись

```
Type
   Person = Record
              Fio, Adres, Gruppa : String; {Фамилия, адрес, группа}
               Sex : (m, w); {Пол}
               Year : Integer; {Год рождения}
            End;
Var
  Student : Person;
  Complex : Record
          Re, Im : Real; {Действительная и мнимая части комплексного числа}
            End;
```

Доступ к полям записи осуществляется с помощью составного имени, которое состоит из имени переменной типа запись и имени поля, записываемого через точку, например Student.Fio, Student.Year, Complex.Re и т.д.

Поле записи используется в программе так же, как и обычная переменная, т.е. поле можно указывать как в левой части оператора присваивания, так и в правой его части (в выражениях), например:

```
Student . Fio := ‘Иванов И.И.’;
```

```
Student . Adres := ‘Владимир’;
Student . Gruppa :=  ‘ИСЕ-199’;
Student . Sex := m;
Student . Year := 1982;
Complex . Re := -17.4;
Complex . Im := Complex . Re;
```

Для того чтобы не выписывать каждый раз имя записи при обращении к ее полям, можно использовать *оператор присоединения* WITH. Его структура выглядит следующим образом:

WITH <список переменных-записей> DO <оператор>;

Например, фрагмент

```
With Student, Complex Do
    Begin
      Fio := ‘Иванов И.И.’;
     Adres := ‘Владимир’;
     Gruppa :=  ‘ИСЕ-199’;
    Sex := m;
    Year := 1982;
    Re := -17.4;
    Im := Re;
  End;
```

эквивалентен фрагменту из предыдущего примера.

Над полями записи можно выполнять любые действия, допустимые для данных его типа. Например, введем с клавиатуры и выведем на дисплей несколько полей записей Student и Complex:

а) ввод данных

```
With Complex Do
  Begin
    Write(‘Введите действительную часть числа:’);
    Readln(Re);
    Write(‘Введите мнимую часть числа:’);
    Readln(Im);
  End;
```

б) вывод данных

```
With Student Do
  Begin
     Writeln(‘Студент  ‘, Fio);
     Writeln(Year, ‘  года рождения’);
     Writeln(‘ Адрес - ’,Adres);
  End;
```

Примеры решения задач на использование записей

Пример 2.17. Дан список, состоящий из 50 записей. Каждая запись содержит фамилию и адрес жителя. Написать программу, которая выводит на экран монитора

фамилии двух жителей из списка, живущих в разных городах по одинаковому адресу.

Решение.

Необходимо организовать сравнение адреса I-го жителя с адресами последующих в списке жителей. Для этого потребуется конструкция вложенный цикл. В случае совпадения адресов печатаются фамилии жителей и осуществляется переход к следующему в списке жителю.

Один из вариантов программы.

```
Program Prim217;
 Type
  Person = Record
            Fam, Town : String;
            Adres : Record
                     Strit, House : String;
                     Flat : Word;
                    End;
           End;
 Var
  List : Array [1..50] Of Person;
  I, J, K : Byte;
 Begin
  Writeln('Введите список жителей');
  For I:=1 To 50 Do
   With  List[I], Adres Do
    Begin
     Writeln('Сведения о ', I:2,' жителе');
     Write('Фамилия');
     Readln(Fam);
     Write('Город');
     Readln(Town);
     Write('Улица');
     Readln(Strit);
     Write('Дом');
     Readln(House);
     Write('Квартира');
     Readln(Flat);
    End;
  K:=0;
  For I:=1 To 49 Do
   For J:=I+1 To 50 Do
     If (List[I].Town<>List[J].Town) And
        (List[I].Adres.Strit=List[J].Adres.Strit) And
        (List[I].Adres.House=List[J].Adres.House) And
        (List[I].Adres.Flat=List[J].Adres.Flat) Then
```

```
          Begin
           Writeln(List[I].Fam, '  ',List[J].Fam);
           K:=1;
           Break
          End;
    If K=0 Then
      Writeln('Нет жителей, живущих в разных городах по одному адресу')
    End.
```

Пример 2.18. Написать программу, представляющую на экране монитора все фамилии студентов из группы, начинающиеся с указанной буквы, и даты их рождения.

Решение.

Возможный вариант программы:

```
Program Prim218;
Type
  Data = Record
         Day : 1..31;
         Month : 1..12;
         Year : 1900..2100;
        End;
  Anketa = Record
           FIO : String;
           Sex : Char;
           DayR : Data;
          End;
  Gruppa = Array [1..25] Of Anketa;
Var
  Gr : Gruppa;
  Buk : Char;
  N, I : Integer;
Begin
  Write('Введите количество человек в группе (<=25):  ');
  Readln(N);
  Writeln('Введите анкетные данные на каждого человека в группе ');
  For I:=1 To N Do
    With Gr[I], DayR Do
     Begin
      Write('ФИО  ');
      Readln(FIO);
      Write('Пол   ');
      Readln(Sex);
      Write('Дата рождения (ДД ММ ГГГГ)  ');
      Readln(Day, Month, Year);
     End;
```

```
  Write('Символ, по которому производится отбор фамилий  ');
  Readln(Buk);
  Writeln('Список фамилий, начинающихся на букву  ',Buk);
  For I:=1 To N Do
    With Gr[I], DayR Do
      If FIO[1] = Buk Then
        Writeln(FIO, Day:4,'.',Month:2,'.',Year:4);
End.
```

Пример 2.19. Задана таблица запуска партий деталей в обработку, которая содержит тип деталей и размер их партии. Следует определить общее количество деталей, тип которых вводится с клавиатуры.

Решение.

Программа

```
Program Prim3;
Const N=7; {Число типов деталей}
Type
  Zap = Record
        TypeDet : String;   {Тип детали}
        SizePart : Integer; {Размер партии}
       End;
Var
  Tab : Array [1..N] Of Zap;
  Kol, I : Integer;
  Det : String;
Begin
  Writeln('Введите таблицу Tab');
  For I:=1 To N Do
   With Tab[I] Do
    Begin
     Write('Введите тип детали  ');
     Readln(TypeDet);
     Write('Введите размер партии  ');
     Readln(SizePart);
    End;

Write('Введите начальные символы, с которых начинается тип детали   ');
  Readln(Det);
  {Определение количества Kol деталей типа Det }
  Kol := 0;
  For I:=1 To N Do
   With Tab[I] Do
    If Copy(TypeDet,1,Length(Det)) = Det
      Then Kol := Kol + SizePart;
 Writeln('Количество деталей, тип котрых начинается на   ', Det,' =  ',Kol);
```

```
End.
```

Пример 2.20. Дана ведомость абитуриентов, сдававших три вступительных экзамена в университет. Определить количество абитуриентов, набравших больше двенадцати баллов.

Решение.

Возможный вариант программы:

```
Program Prim220;
Var
  Ab : Record
        FIO : String;   {ФИО абитуриента}
        Ocen : Array [1..3] Of Byte; {Массив оценок}
       End;
  N, I, J, Kol : Integer;
  Bal : Real;
Begin
 Write('Введите количество абитуриентов  ');
 Readln(N);
 Kol := 0;
{Ввод ведомости}
 For I:=1 To N Do
  With Ab Do
   Begin
    Write('ФИО  ');
    Readln(FIO);
    Write('Введите три оценки   ');
    Readln(Ocen[1],Ocen[2],Ocen[3]);
    Bal := Ocen[1]+Ocen[2]+Ocen[3];
    If Bal>12 Then
      Kol := Kol + 1;

 End;
 Writeln('Количество абитуриентов =  ',Kol);
End.
```

Контрольные вопросы

1. Какой тип характеризует объекты, называемые записями?
2. Как описывается комбинированный тип?
3. Могут ли компоненты записи быть различных типов?
4. Какие операции определены над записями?
5. Как осуществляется доступ к полям записи?
6. С какой целью в Турбо Паскале используется оператор With?

Задачи

1. Опишите тип записи информации о книгах в библиотеке со следующими полями: автор, название, издательство, год издания, взята книга или нет, номер полки. Напишите программу, которая выводит из массива информацию о тех книгах, которые находятся на руках и которые изданы до 1990 г.

2. На кинофестивале 35 стран представили свои фильмы. Общее число фильмов не превышает 100. Известны названия стран-участниц и фильмов, а также баллы, полученные каждым из фильмов. Определить фильм, завоевавший первый приз (максимальный балл), и страну, получившую наибольший средний балл за представленные фильмы. Считать, что фильмы в общем списке по странам неупорядочены, а фильм и страна, его представляющая, являются единственными победителями.

3. Имеются сведения об *N* студентах (*N* - заданное число): фамилия, символьный шифр группы и 4 экзаменационные оценки. Требуется определить наименьшую из средних экзаменационных оценок студентов, а затем для каждой группы получить пронумерованные списки студентов, имеющих такое же значение средней экзаменационной оценки, или выдать сообщение, что таких студентов нет.

4. Известны сведения о каждом из *N* рабочих (*N* - заданное число): фамилия, год рождения и номер бригады. Необходимо для каждой бригады получить два списка: самых молодых и самых старых рабочих. Предусмотреть, что нумерация бригад может быть несплошной.

5. Даны названия *N* различных спортивных обществ (*N* - заданное число), фигуристы которых участвовали в соревновании. У каждого фигуриста известны фамилия, название общества и 10 оценок за его выступление. Требуется для каждого спортивного общества определить фигуриста, показавшего наивысший результат, считая его единственным. Баллы, полученные спортсменом, подсчитываются следующим образом: максимальная и минимальная оценки отбрасываются, а из остальных формируется средняя. При вводе данных обеспечить уникальность наименований обществ и обязательную принадлежность фигуриста к одному из них.

6. Известна стоимость суточного проживания в каждом номере гостиницы из *F* этажей по *K* номеров на этаже. О проживающих известны следующие данные: гостиничный номер, фамилия, дни въезда и выезда. Необходимо напечатать для клиентов счета с указанием дней их выезда, фамилии, номера и этажа, дней въезда и выезда и суммы денег за время проживания. (*F* и *K* заданы, нумерация гостиничных номеров сплошная, днями въезда и выезда считать числа месяца от 1 до 30).

7. Имеются сведения об *N* студентах (*N* - заданное число): фамилия, символьный шифр группы и 4 экзаменационные оценки. Требуется определить максимальную из средних экзаменационных оценок студентов, а затем для каждой группы получить пронумерованные списки студентов, имеющих значение средней экзаменационной оценки, меньшее максимального, или выдать сообщение, что таких студентов нет.

8. В районном обществе автолюбителей имеются сведения об *N* автомобилях (*N* - заданное число). Для каждой машины известно: фамилия владельца, год выпуска и номер автостоянки. Необходимо для каждой из стоянок получить два

списка: список самых новых и список самых старых машин с указанием их владельцев и года выпуска. Предусмотреть то, что нумерация стоянок может быть несплошной.

9. В гостинице проживает *N* постояльцев (*N* - задано). О каждом известны три характеристики: номер проживания, фамилия, заказанное на завтрак блюдо (или отсутствие заказа). Необходимо составить сводные (по наименованиям заказанных блюд) заявки на кухню с указанием гостиничных номеров и фамилий постояльцев.

10. В автохозяйстве имеется *N* автомашин (*N* - заданное число). Для каждого автомобиля заданы три характеристики: номер, марка машины, тип неисправности (или ее отсутствие). Необходимо составить сводные (по типам неисправностей) заявки на ремонт машин с указанием их номеров и марок.

11. Имеется *N* типов товаров (названия известны). Для каждого типа товара задано количество единиц этого товара, цена и вес единицы товара. Требуется загрузить контейнер (не превышая его известной грузоподъемности) товарами одного типа так, чтобы стоимость груза в контейнере была максимальной.

12. 200 учеников шести школ города (номера школ заданы) принимают участие в тестировании по математике. Правильные численные ответы к пяти предложенным задачам даны. У каждого ученика известны: фамилия, номер школы и пять ответов на задачи. Сведения об учениках не имеют определенной упорядоченности. Составить списки учеников по школам, расположив в каждом списке фамилии учащихся в порядке убывания числа решенных ими задач. Предусмотреть возможный ответ "не решил".

13. Имеется список 60 зданий города, подлежащих реконструкции. Сведения о каждом здании содержат названия микрорайона, улицы, номер дома и год постройки. Определить самые старые здания из подлежащих реконструкции и вывести их списки, содержащие полные сведения о них, по микрорайонам. Если в микрорайоне таких домов нет, выдать соответствующее сообщение.

14. Список *N* рабочих цеха (*N* - заданное число) содержит следующие сведения о каждом рабочем: фамилия, числовой номер бригады, зарплата. Список не имеет определенной упорядоченности. Вывести на экран списки рабочих по бригадам, расположив в списках фамилии рабочих в порядке убывания их зарплаты. Определить среднюю зарплату по всем рабочим и подсчитать для каждой бригады количество рабочих, имеющих зарплату ниже средней. Бригады нумеруются подряд, начиная с первой.

15. На заводе имеется *N* станков (*N* - заданное число). Для каждого станка заданы три характеристики: инвентарный числовой номер, марка станка и тип неисправности (или ее отсутствие). Необходимо составить сводные (по типам неисправностей) заявки на ремонт станков с указанием их номеров и марок.

16. Даны названия *N* обществ, спортсмены которых участвовали в соревнованиях по лыжной гонке. О каждом участнике соревнований известны следующие данные: название общества, фамилия спортсмена и время прохождения трассы. Вывести на экран сведения о лучшем результате спортсмена каждого общества, считая, что нет одинаковых результатов.

17. Для каждого участника соревнований вводится фамилия, время старта (часы, минуты, секунды), время финиша. Вывести на экран фамилии участников, выполнивших заданный норматив.

18. Сформировать список из фамилий и размеров обуви, которую носит каждый студент. Используя сформированный список, вывести на экран фамилии студентов, начинающихся с “КА”, у которых размер обуви не менее 40.

19. Сформировать список, содержащий информацию о владельцах автомобилей: фамилия, марка, цвет, гос. номер. Используя сформированный список, вывести на экран всех владельцев автомобилей ВАЗ белого цвета.

20. Сформировать список, содержащий информацию о телевизорах: фирма-изготовитель, размер экрана в дюймах, стоимость. Используя сформированный список, вывести на экран информацию о телевизорах указанной стоимости.

21. Сформировать список, содержащий информацию о торговых точках вблизи вашего дома: название фирмы, ассортимент товаров (продукты питания, одежда, обувь, электроника, хозтовары, культтовары), адрес. Используя сформированный список, вывести на экран информацию о магазинах, торгующих одинаковыми товарами.

22. Сформировать расписание, содержащее информацию о поездах, отправляющихся с Киевского вокзала (номер поезда, станция назначения, время отправления, время в пути). Используя сформированное расписание, вывести на экран дисплея информацию о поездах, следующих в Киев и находящихся в пути менее 12 часов.

23. Сформировать список, содержащий данные о книгах по информатике (фамилия автора, его инициалы, название книги, название издательства, год издания). Используя сформированный список, вывести на экран дисплея фамилии авторов и названия книг, выпущенных издательством "Мир".

Глава 3. ФАЙЛЫ

3.1. Основные процедуры и функции работы с файлами

Часто если программа или набор данных используется многократно, то их следует хранить в виде файлов. *Файл* – это поименованная область на диске, содержащая некоторые данные.

В ряде случаев под файлом подразумевается устройство, которое может выдавать или получать информацию, например принтер или клавиатура. Далее речь будет идти только о файлах данных.

По отношению к программе файлы могут быть *внешними* и *внутренними*. *Внутренними* файлами являются такие, которые создаются, используются и существуют только во время работы данной программы. Файлы, которые существуют вне программы, называются *внешними* файлами. Они могут быть подготовлены (заполнены данными) в одной программе, а использоваться (обрабатываться) - в другой.

С точки зрения работы с компонентами файлов во время исполнения программы различия между внешними и внутренними файлами нет. Это различие проявляется лишь после окончания выполнения программы: внутренние файлы прекращают свое существование, как прекращают свое существование все другие объекты программы, а внешние файлы сохраняются на внешних носителях данных, например магнитных дисках.

Каждый внешний файл должен иметь имя, которое состоит из двух частей: собственно имени и расширения. Имя от расширения отделяется точкой. В системе MS-DOS имя может содержать до восьми символов, расширение — до трех символов. Например, в имени файла File.Dat File — собственно имя, Dat — расширение (принято обычно для файлов данных). Но расширение может и отсутствовать.

Полное имя файла кроме этого содержит еще имя диска и имена директорий, в которых располагается файл. Например, D:\APPDOS\ File.Dat, где D: - имя жесткого диска ("винчестера"); APPDOS — имя директории, где располагается файл; символ "\" — разделитель.

В языке Турбо Паскаль для работы с файлами используется файловый тип. Значение файлового типа представляет собой набор элементов одного и того же типа, причем число элементов, называемое длиной файла, не фиксируется.

Над значениями файлового типа не определены какие-либо операции. Все операции могут производиться лишь с элементами (компонентами) файлов. Множество операций над компонентами файла определяется типом компонент.

Доступ к компонентам файла осуществляется через *указатель файла* с помощью специальных стандартных процедур. При чтении или записи этот указатель перемещается к следующей компоненте и делает ее доступной для обработки. В каждый момент доступен для записи (чтения) только тот элемент файла, на который установлен указатель.

Для доступа к файлу в программе необходимо:

1) определить файловую переменную, которая в последующем используется как логический идентификатор файла;

2) для внешнего файла связать эту переменную с конкретным физическим файлом;

3) инициировать файл.

При определении файловой переменной обычно задается тип файла. Турбо Паскаль имеет три категории фалов: типизированные, текстовые и нетипизированные. Далее будут рассматриваться только типизированные и текстовые файлы.

Описание типизированных файлов имеет следующий вид:

```
Type
  <имя типа> = File of <базовый тип>;
Var
  <имя файловой переменной> : <имя типа>;
```

или

```
Var
    <имя файловой переменной> : File of <базовый тип>;
```

В качестве базового типа элементов файла можно использовать любой тип данных (как простой, так и сложный) за исключением файлового типа, а также сложного типа, компоненты которого являются файлами.

Пример:

```
Type
  Digit = File of Byte;
  Mas = Array [1..20] of Real;
  Zap = Record
        X, Y : Integer
        End;
Var
  F1 : Digit; {Каждый элемент файла – целое число от 0 до 255}
  F2 : File of Real;  {Каждый элемент файла – вещественное число}
  F3 : File of String; {Каждый элемент файла – строка}
  F4 : File of Mas;  {Каждый элемент файла – массив}
  F5 : File of Zap;  {Каждый элемент файла – запись}
```

Особым типом файлов являются текстовые файлы, которые состоят из символов, объединенных в строки. Для описания текстовых файлов в Турбо Паскаль введен стандартный тип TEXT.

Пример:

```
Var
  F6 : Text;
```

После того, как файловая переменная определена, в программе ее нужно связать с конкретным физическим файлом. Для этой цели используется стандартная процедура Assign(F, Name). Она связывает файловую переменную F с внешним файлом, имеющим имя Name. Name – это переменная или константа типа String. Имя файла должно быть написано в соответствии с правилами MS DOS.

Пример:

```
Const
  Name = ‘a:\dir\subdir\out.txt’;
Var
  Finp : Text;
  Fout : File of Real;
Begin
 . . . . . .
 Assign(Finp, ‘D : STR . TXT’);
 Assign(Fout, Name);
. . . . . .
```

Инициировать файл означает указать для этого файла направление передачи данных. В Турбо Паскале можно открыть файл для чтения, для записи данных, а также для чтения и записи одновременно.

Стандартная процедура Reset(F) открывает существующий файл, с которым связана файловая переменная F, и устанавливает указатель файла на первый элемент файла, т.е. на элемент с номером 0.

Стандартная процедура Rewrite(F) открывает новый пустой файл для записи, и ему присваивается имя, заданное процедурой Assign. Если файл с таким именем уже существует, то он уничтожается и никаких сообщений в программу не передается. Файл подготавливается к приему информации, а его указатель устанавливается на первый элемент файла.

Далее кратко описаны некоторые стандартные процедуры и функции, которые используются при работе с файлами. Во всех этих процедурах и функциях F – файловая переменная, связанная с конкретным физическим файлом процедурой Assign.

Процедуры и функции, которые применимы для всех категорий файлов

Процедуры

Read(F, X1,X2,...,XN) — считывает в переменную X1 один элемент файла F (или несколько элементов в переменные X1, ..., XN), начиная чтение с элемента, на котором установлен указатель.

Write(F, Xl, X2, ...,XN) — записывает одно (X1) или более (X1,.... XN) значений переменных в файл F, начиная с той позиции, на которую установлен указатель.

Close(F) — закрывает файл F.

Erase(F) — удаляет внешний файл, связанный с файловой переменной F.

Rename(F, NewName) — присваивает внешнему файлу, связанному с файловой переменной F, новое имя NewName.

Функции.

Eof(F) – конец файла - принимает значение True, если указатель находится за последней компонентой файла, и False - противном случае.

IOResult – возвращает условный признак последней операции ввода-вывода. Если операция завершилась успешно, функция возвращает ноль. В противном случае она возвращает код ошибки ввода-вывода. Следует помнить, что функция

IOResult становится доступной при отключенном автоконтроле ошибок ввода-вывода. Директива компилятора {$I-} отключает, а директива {$I+} включает автоконтроль.

Процедуры и функции для типизированных файлов

Процедуры

Seek(F, N) – смещает указатель файла к компоненту с номером N. При этом первый компонент файла имеет номер N = 0, второй – N = 1 и т.д.

Truncate(F) – удаляет часть файла, начиная с текущего компонента и до конца файла.

Функции.

FilePos(F) – возвращает порядковый номер компонента файла, на котором стоит указатель файла.

FiIeSize(F) – возвращает значение, которое содержит размер (число компонент) файла.

3.2. Работа с файлами

Работа с разными категориями файлов несколько отличается друг от друга, но общие требования к работе со всеми категориями файлов одинаковы:

- файл должен быть открыт для работы с ним;
- файл должен быть создан и записан на внешний носитель (запись файла);
- файл может быть считан с внешнего носителя в оперативную память (чтение файла) и обработан;
- обработанный файл может быть снова записан на внешний носитель;
- файл должен быть закрыт по окончании работы с ним.

Рассмотрим пять основных операций с внешними файлами:

1) запись в файл;
2) чтение файла;
3) добавление данных к файлу;
4) запись и чтение файла;
5) прямая выборка элементов файла.

Запись в файл

Под записью в файл понимается вывод результатов программы из оперативной памяти ЭВМ на диск, т.е. создание нового файла на внешнем устройстве. Внешний файл, в который записываются данные из программы, называют *выходным.*

Для записи файла в программе необходимо выполнить следующие действия:

1) описать файловую переменную;
2) связать ее с выходным файлом (Assign);
3) открыть файл для записи (Rewrite);
4) вывести данные из программы в файл (Write);

5) закрыть файл для записи (Close);

Общая форма записи в файл имеет следующий вид:

```
Rewrite(F);
. . . . . .
Write(F,Al,A2,...,An);
. . . . . .
Close(F);
```

Здесь F — имя файла; Al, A2, ..., An — выражения (в том числе константы и переменные) того же типа, что и элементы файла.

Процедура Rewrite осуществляет подготовку к записи информации в файл F (очищает файл и устанавливает указатель в начало файла). Оператор Write записывает значения выражений Al, A2, ... An по одному в конец файла F (после имеющейся в нем информации).

Пример 3.1. Создать файл данных с именем D:\APPDOS\FP1.DAT, содержащий фамилии студентов и год их рождения.

Решение.

Возможный вариант программы:

```
Program Prim31;
 Type
  Stud = Record
           Fio : String;
          Year : Integer;
         End;
 Var
  F : File of Stud;
  St : Stud;
  C : Char;
 Begin
  Assign(F, 'D:\APPDOS\FP1.DAT');
  Rewrite(F);
  C:='Д';
  While (C='Д') OR (C ='д') Do
   Begin
   {Ввод с клавиатуры двух полей записи}
    Write('Введите фамилию   ');
    Readln(St.Fio);
    Write('Введите год   ');
    Readln(St.Year);
    {Запись в файл. Элементом файла является вся запись}
    Write(F, St);
    Writeln('Продолжим ввод?(д/н)');
    Readln(C);
   End;
  Close(F);
  End.
```

Чтение файла

Под чтением файла понимается ввод данных из внешнего файла, находящегося на диске, в оперативную память ПЭВМ.

Для чтения файла в программе необходимо выполнить следующие действия:

1) описать файловую переменную;
2) связать ее с внешним файлом (Assign);
3) открыть файл для чтения (Reset);
4) ввести данные файла в программу (Read);
5) закрыть файл для чтения (Close).

Общая структура фрагмента программы, составленной для чтения файла, имеет вид:

```
Reset(F);
. . . . . .
Read(F,Xl.X2,...,XN);
. . . . . .
Close(F);
```

Тип переменных X1, X2, ..., XN должен соответствовать базовому типу элементов файла (это условие необязательно для текстовых файлов).

Процедура Reset осуществляет подготовку к чтению из файла F (открывает файл и устанавливает указатель на первый элемент файла).

Оператор Read последовательно присваивает переменным X1, X2, ..., XN значения из файла F. Количество этих переменных не должно превышать количества данных, которые могут быть введены из файла. В противном случае выполнение программы прекращается. Вместо Read (F,X1,X2,... ,XN) можно использовать несколько операторов Read:

```
Read(F, Xl); Read(F, X2); ...; Read(F, XN);
```

После чтения очередного элемента указатель файла устанавливается на следующий за прочитанным элемент файла.

Процедура Close выполняет закрытие файла F.

Пример 3.2. Из созданного программой Prim31 файла с именем D:\APPDOS\ FP1.DAT вывести на экран информацию о студентах, родившихся в 1985 г.. Полученные данные записать (для отчета) в файл с именем D:\APPDOS\FP2.DAT.

Решение.

Возможный вариант программы:

```
Program Prim32;
 Type
  Stud = Record
          Fio : String;
          Year : Integer;
         End;
 Var
  St : Stud;
```

```
 F1, F2 : File of Stud;
Begin
 Assign(F1, 'D:\APPDOS\FP1.DAT');
 Assign(F2, 'D:\APPDOS\FP2.DAT');
 Reset(F1);
 Rewrite(F2);
      Writeln('Список студентов, родившихся в 1985 году');
{ Пока не достигнут конец файла F1, осуществляется считывание из него данных, их обработка и запись в новый файл}
 While Not EOF(F1) Do
  Begin
{Чтение записи из файла}
   Read(F1, St);
   If St.Year = 1985 Then
     Begin
{Вывод записи на экран}
      Writeln(St.Fio, St.Year:6);
{Вывод записи в файл}
      Write(F2, St);
     End;
  End;
Close(F1);
Close(F2);
  End.
```

Заметим, что в этом примере мы используем файл D:\APPDOS\FP1.DAT, пребывая в полной уверенности, что такой файл на диске существует. Действительно, в предыдущем примере мы создали такой файл. Однако если делается попытка инициировать чтение несуществующего файла, возникает ошибка периода исполнения, которая может быть сообщена программе ненулевым значением функции IOResult. Поэтому более правильно было бы убедиться в наличии файла на диске прежде, чем начать работу с файлом в новой программе. Ниже приводится вариант программы, в которой устанавливается существование требуемого файла на диске:

```
Program Prim32a;
 Label 10;
Type
 Stud = Record
        Fio : String;
        Year : Integer;
       End;
Var
 St : Stud;
 F1, F2 : File of Stud;
Begin
 Assign(F1, 'D:\APPDOS\FP1.DAT');
```

```
 {$I-}
 Reset(F1);
{$I+}
        If IOResult <> 0 Then
           Begin
            Writeln('Файл не существует');
            GoTo 10
          End
                        Else
           Begin
          Assign(F2, 'D:\APPDOS\FP2.DAT');
              Rewrite(F2);
              Writeln('Список студентов, родившихся в 1985 году');
              { Пока не достигнут конец файла F1, осуществляется считывание
          из него данных, их обработка и запись в новый файл}
               While Not EOF(F1) Do
                Begin
              {Чтение записи из файла}
                 Read(F1, St);
                 If St.Year = 1970 Then
                   Begin
              {Вывод записи на экран}
                    Writeln(St.Fio, St.Year:6);
              {Вывод записи в файл}
                    Write(F2, St);
                   End;
                End;
          Close(F1);
          Close(F2);
          10: End;
   End.
```

Пример 3.3. Составить программу для вычисления среднего арифметического положительных значений элементов файла с именем File1.Dat.

Решение.

В данной программе для чтения файла используется оператор цикла с постусловием Repeat.

Программа

```
Program Prim33;
 Var
   F : File of Real;
   S, X : Real;
   N : Integer;
 Begin
  Assign(F, 'File1.Dat');
  Reset(F);
```

```
 N:=0;
 S:=0;
 Repeat
  Read(F, X);
  If X > 0 Then
    Begin
      S := S + X;
      N := N + 1;
    End;
 Until Eof(F);
 S := S / N;
 Writeln('Среднее арифметическое =  ', S);
End.
```

Мы рассмотрели два основных действия над последовательными файлами: их чтение и запись. Все остальные действия представляют собой сочетание записи и чтения файла.

Добавление данных к файлу

В Турбо Паскале разрешается обращаться к типизированному файлу, открытом процедурой Reset (т.е. для чтения информации), с помощью процедуры Write (т.е. для записи информации).

Пусть необходимо к уже существующему типизированному файлу добавить ряд элементов. Для этого надо выполнить следующие действия:

1) открыть уже существующий файл F процедурой Reset;

2) установить указатель файла за последним его компонентом процедурой Seek следующим образом:

```
Seek(F,FileSize(F));
```

3) записать дополнительные данные (Write);

4) закрыть файл (Close).

Пример 3.4. Пусть уже создан файл F, состоящий из целых чисел 1, 2, 3, 4, 5. Необходимо добавить к этому файлу числа 10, 20, 30.

Решение.

Возможный вариант программы:

```
Program Prim34;
 Var
       F : File of Integer;
    I,X : Integer;
 Begin
  Assign(F, 'F');
 Reset(F);
 Seek(F, FileSize(F));
 For I :=1 to 3 do
   Begin
     X:=10*I;
```

```
      Write(F, I)
    End;
  Close(F);
End.
```

В результате выполнения программы получим файл F, состоящий из чисел 1, 2, 3, 4, 5, 10, 20, 30.

Поиск заданного элемента файла

Пусть необходимо найти в конкретном файле определенный элемент. Для этого надо выполнить следующие действия:

1) открыть файл для чтения (Reset);
2) читать файл, пока не будет найден нужный элемент (Read);
3) закрыть файл.

Пример 3.5. Во внешнем файле с именем Sessia.Dat хранится информация о результатах экзаменационной сессии студентов. Каждый элемент этого файла представляет собой запись, состоящую из полей: ФИО, номер группы и четыре оценки. Написать программу, которая по введенным с клавиатуры ФИО и номеру группы представляет на экране монитора сведения о результатах сдачи экзаменов данным студентом.

Решение.

Возможный вариант программы:

```
Program Prim35;
 Type
  Zap = Record
        Fio, NumGr : String;
        Oc : Array [1..4] of Byte;
       End;
 Var
  Sessia : File of Zap;
  S : Zap;
  F, N : String;
  Begin
  Assign(Sessia, 'Sessia.Dat');
  Reset(Sessia);
  Write(' Введите ФИО студента  ');
  Readln(F);
  Write('Введите номер группы  ');
  Readln(N);
  Repeat
    Read(Sessia, S);
  Until (S.Fio = F)And(S.NumGr = N) Or Eof(Sessia);
  IF (S.Fio = F) And (S.NumGr = N) Then
          Writeln('ФИО ',S.Fio,' Группа',S.NumGr:5,' Оценки: ',
S.Oc[1]:4,S.Oc[2]:4,S.Oc[3]:4,S.Oc[4])
```

```
    Else Writeln('Запись не найдена');
  Close(Sessia);
   End.
```

Корректировка элементов файла

Для проведения корректировки необходимо выполнить следующие действия:
1) открыть корректируемый файл (Reset);
2) определить корректируемый элемент;
3) подвести указатель файла к корректируемому элементу (с помощью Seek);
4) считать корректируемый элемент (Read);
5) откорректировать элемент файла;
6) повторить процедуру подвода указателя файла (Seek);
7) записать откорректированный элемент (Write);
8) закрыть файл (Close).

Пример 3.6. Пусть уже создан файл F, содержащий числа 1, 2, 3, 4, 5. Необходимо вместо последних трех элементов файла F записать числа 30, 40, 50.

Решение.

Программа

```
program Prim36;
  Var
    F : File of Integer;
    X, I : Integer;
Begin
  Assign(F, 'F');
  Reset(F);
  For I := 3 to 5 do
    begin
     Seek(F, I-1);
     Read(F, X);
     X := 10*I;
     Seek(F, I-1);
     Write(F,X)
    end;
    Reset(F);
    While not Eof(F) do
      Begin
        Read(F, X);
        Writeln(X)
      end;
    Close(F);
  end.
```

Удаление элемента из файла

Для этого используется вспомогательный файл, в который сначала переписываются из внешнего файла все оставляемые записи, затем внешний файл открывается для записи, вспомогательный файл - для чтения и все содержимое вспомогательного файла переписывается во внешний файл.

Пример 3.7. Из внешнего файла с именем D:\APPDOS\ FP1.DAT, созданным в примере 3.1, удалить все записи о студентах, родившихся до 1980 года.

Решение.

Программа

```
Program Prim37;
 Type
  Stud = Record
          Fio : String;
          Year : Integer;
         End;
 Var
  F, FV : File of Stud; {F - внешний файл, FV - вспомогательный файл}
  S : Stud;
 Begin
  Assign(F, 'D:\APPDOS\FP1.DAT');
  Assign(FV, 'D:\APPDOS\FP3.DAT');
 {запись в вспомогательный файл отобранных записей}
  Reset(F);
  Rewrite(FV);
  While Not EOF(F) Do
   Begin
    Read(F, S);
    If S.Year >= 1980 Then
      Write(FV, S);
   End;
     { внешний файл открывается для записи, вспомогательный файл - для чтения, и все содержимое вспомогательного файла переписывается во внешний файл}
    Reset(FV);
    Rewrite(F);
    While Not EOF(FV) Do
     Begin
      Read(FV, S);
      Write(F, S);
     End;
 {вывод для контроля после удаления записей}
    Reset(F);
    While Not Eof(F) do
     Begin
       Read(F,S);
      Writeln(S.Fio, S.Year:6);
```

```
    End;
  Close(F);
  Close(FV);
  {удаление вспомогательного файла с диска}
  Erase(FV);
End.
```

Контрольные вопросы

1. Что такое файл данных?
2. Какие виды файловых типов существуют в Турбо Паскале?
3. Как в разделе типов задается файловый тип?
4. Каким образом описываются переменные файловых типов?
5. Как подразделяются файлы по видам доступа к их компонентам?
6. Какие операции определены над файлами?
7. Что необходимо выполнить для открытия файлов?
8. Какие процедуры предназначены для открытия файлов и как они работают?
9. Как распознать конец файла?
10. Как распознать файл на диске?
11. Для чего предназначена процедура Close?
12. По каким правилам выполняется чтение типизированных файлов?
13. Как осуществляется запись в файл?
14. Какие процедуры и функции предназначены для прямого доступа к элементам типизированных файлов?
15. В чем состоят особенности работы с текстовыми файлами?

Задачи I уровня

Эти задачи предназначены для приобретения навыков работы с типизированными файлами с использованием типовых алгоритмов.

Постановка задачи

Подготовить данные об абитуриентах, поступающих в университет. Информацию о каждом абитуриенте оформить в виде записи, содержащей следующие поля:

1. Фамилия, имя, отчество.
2. Год рождения.
3. Год окончания школы.
4. Средний балл в аттестате.
5. Признак - нуждается ли в общежитии.
6. Оценки вступительных экзаменов.

Разработать программу записи подготовленных данных во внешний файл и программу обработки созданного внешнего файла.

Варианты задания

I. Из внешнего файла, содержащего исходные данные, удалить записи об абитуриентах:

1. Получивших хотя бы одну оценку 2.

2. Получивших все оценки 3.
3. Имеющих средний балл больше 4,5 и нуждающихся в общежитии.
4. Имеющих средний балл в аттестате меньше 4.
5. Старше восемнадцати лет.
6. Неполучивших ни одной оценки 5.
7. Имеющих отличный аттестат и получивших все оценки 5.
9. Нуждающихся в общежитии и получивших хотя бы одну оценку 3.
10. Старше семнадцати лет, имеющих отличный аттестат.

Распечатать полученный файл.

II. Используя внешний файл, содержащий исходные данные, добавить N записей и распечатать список абитуриентов:

1. Имеющих в аттестате оценки только 5; N=2.
2. Имеющих в аттестате одну оценку 4, а остальные оценки 5; N=3.
3. Имеющих средний балл больше 4,5; N=4.
4. Имеющих средний бал меньше 4; N=3.
5. Ненуждающихся в общежитии; N =2.
6. Нуждающихся в общежитии; N=3.
7. Сдавших вступительные экзамены только на оценки 5; N=4.
8. Сдавших вступительные экзамены на оценки 4 и 5; N=2.
9. Сдавших экзамены с двумя оценками 4 и остальными оценками 5; N=3.
10. Получивших на вступительных экзаменах одну оценку 3; N=4.

Задачи II уровня

1. Создать файл, содержащий сведения о месячной заработной плате рабочих завода. Каждая запись содержит поля: фамилия рабочего, наименование цеха, размер заработной платы за месяц. Количество записей произвольное. Вычислить общую сумму выплат за месяц по цеху *X,* а также среднемесячный заработок рабочего этого цеха. Напечатать для бухгалтерии ведомость для начисления заработной платы рабочим этого цеха.

2. Создать файл, содержащий сведения о количестве изделий, собранных сборщиками цеха за неделю. Каждая запись содержит поля: фамилия сборщика, количество изделий, собранных им ежедневно в течение шестидневной недели, т.е. раздельно — в понедельник, вторник и т.д. Количество записей произвольное. Написать программу, выдающую на печать следующую информацию: фамилия сборщика и общее количество деталей, собранное им за неделю; фамилия сборщика, собравшего наибольшее число изделий, и день, когда он достиг наивысшей производительности труда.

3. Создать файл, содержащий сведения о телефонах абонентов. Каждая запись имеет поля: фамилия абонентов, год установки телефона, номер телефона. Количество записей произвольное. Написать программу, выдающую информацию следующего вида: по вводимой фамилии абонента выдается номер телефона; определяется количество установленных телефонов указанного года. Номер года вводится с терминала.

4. Создать файл, содержащий сведения об ассортименте игрушек в магазине. Структура записи: название игрушки, цена, количество, возрастные границы, например 2—5, т.е. от 2 до 5 лет. Количество записей произвольное. Написать программу, в результате выполнения которой выдаются следующие сведения: названия игрушек, которые подходят детям от 1 до 3 лет; стоимость самой дорогой игрушки и ее наименование; название игрушки, которая по стоимости не превышает *x* руб. и подходит ребенку в возрасте от *a* до *b* лет. Значения *x, a, b* ввести с терминала.

5. Создать файл, содержащий сведения об ассортименте обуви в магазине фирмы. Структура записи: артикул, наименование, количество, стоимость одной пары. Количество записей произвольное. Артикул начинается с буквы Д для дамской обуви, М - для мужской, П - для детской. Написать программу, выдающую следующую информацию: наличие и стоимость обуви артикула Х; ассортиментный список дамской обуви с указанием наименования и имеющегося в наличии числа пар каждой модели.

6. Создать два файла, содержащих сведения о десяти нападающих хоккейных команд «Динамо» и «Спартак»: имена нападающих, число заброшенных ими шайб, сделанных голевых передач, заработанное штрафное время. Написать программу, которая по данным, извлеченным из этих файлов, создает новый третий файл, содержащий имя, команду, сумму очков (голы + передачи) для шести лучших игроков обеих команд. Имена и показатели результативности хоккеистов вывести на экран.

7. Создать файл, содержащий сведения о том, какие из пяти предлагаемых дисциплин по выбору желает слушать студент. Структура записи: фамилия студента, индекс группы, 5 дисциплин, средний балл успеваемости. Выбираемая дисциплина отмечается символом 1, иначе — пробел. Количество записей произвольное. Написать программу, которая печатает список студентов, желающих прослушать дисциплину *X*. Если число желающих превысит 8 человек, то отобрать студентов, имеющих более высокий средний балл успеваемости.

7. Создать файл, содержащий сведения об отправлении поездов дальнего следования с вокзала. Структура записи: номер поезда, станция назначения, время отправления, время в пути, наличие билетов. Количество записей произвольное. Написать программу, которая позволяет получить следующую справочную информацию: время отправления поездов в город *X* во временном интервале от *A* до *B* часов; наличие билетов на поезд с номером *XXX*.

8. Создать файл, содержащий сведения о сотрудниках института. Структура записи: фамилия работающего, название отдела, год рождения, стаж работы, должность, оклад. Количество записей произвольное. Написать программу, которая позволяет получить следующую информацию: список сотрудников пенсионного возраста на сегодняшний день с указанием стажа работы; средний стаж работающих в отделе *X*.

9. Создать файл, содержащий сведения о сдаче студентами сессии. Структура записи: номер группы, фамилия студента, оценки по пяти экзаменам и пяти зачетам («з» означает зачет, «н» — незачет). Количество записей — 25. Написать программу, выдающую следующую информацию: фамилии неуспевающих

студентов с указанием индексов групп и количества задолженностей; средний балл, полученный каждым студентом группы *X* и всей группой в целом.

10. Создать файл, содержащий сведения о личной коллекции книголюба. Структура записи: шифр книги, автор, название, год издания, местоположение (номер стеллажа, шкафа и т.п.). Количество записей произвольное. Написать программу, выдающую следующую информацию: местонахождение книги автора *X* названия *Y* (значения *X, Y* ввести с терминала); список книг автора *Z,* находящихся в коллекции; число книг издания указанного года, имеющееся в библиотеке.

11. Создать файл, содержащий сведения о наличии билетов и рейсах Аэрофлота. Структура записи: номер рейса, пункт назначения, время вылета, время прибытия, количество свободных мест в салоне. Количество записей произвольное. Написать программу, выдающую информацию следующего вида: время отправления самолетов в город *X;* наличие свободных мест на рейс в город *X* с временем отправления *Y*. Значения *X, Y* вводятся по запросу с терминала.

12. Протокол лыжных гонок записать в файл D:\UCHEBA\SKI.DAT. Для каждого участника вводятся фамилия, время старта (часы, минуты, секунды), время финиша. Используя сформированный файл, вывести на экран фамилии участников, выполнивших заданный норматив.

13. Сформировать файл D:\UCHEBA\BOOK.DAT из фамилий любимых писателей студентов группы. Студенты вводят по очереди по три фамилии. Используя сформированный файл, напечатать фамилии пяти наиболее популярных писателей.

14. Сформировать файл D:\UCHEBA\CAR.DAT, содержащий информацию о владельцах автомобилей: фамилия, марка, цвет, госномер. Используя сформированный файл, вывести на экран список владельцев автомобилей ВАЗ белого цвета.

15. Сформировать файл D:\UCHEBA\KURS.DAT, содержащий информацию о студентах курса: фамилия, пол, год и месяц рождения. Используя сформированный, файл, вывести на экран фамилии студентов мужского пола, родившихся в период с 1 сентября по 31 декабря.

16.Сформировать файл, содержащий информацию о поездах, отправляющихся с Курского вокзала г. Москвы: номер поезда, станция назначения, время отправления, время в пути. Используя сформированный файл, вывести на экран список поездов до Нижнего - Новгорода, отправляющихся с 12 до 24 часов.

17. Сформировать файл, содержащий информацию о торговых точках вблизи вашего дома: название фирмы, ассортимент товаров (продукты питания, одежда, обувь, электроника, хозтовары, культтовары), адрес. Используя сформированный файл, вывести на экран информацию о магазинах, торгующих одинаковыми товарами.

18. Сформировать файл, содержащий данные о книгах по информатике (фамилия автора, его инициалы, название книги, название издательства, год издания). Используя сформированный файл, вывести на экран дисплея фамилии авторов и названия книг, выпущенных издательством "Мир".

19. Сформировать файл, содержащий информацию о поездах, отправляющихся с Киевского вокзала г. Москвы (номер поезда, станция назначения, время

отправления, время в пути). Используя сформированный файл, вывести на экран дисплея информацию о поездах, следующих в Киев и находящихся в пути менее 12 часов.

Глава 4. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ. ПРОЦЕДУРЫ И ФУНКЦИИ

4.1. Понятие подпрограммы

Часто в разных местах программы приходится выполнять по сути дела один и тот же алгоритм, но с различными данными. Для обеспечения большей компактности и повышения наглядности программы такой частичный алгоритм целесообразно выделить из основного текста программы и записать его только один раз, оформив его в виде самостоятельного программного объекта – *подпрограммы* Таким образом, подпрограммы представляют собой инструмент, с помощью которого любая программа может быть разбита на ряд в известной степени независимых друг от друга частей.

Подпрограмма должна быть описана до того, как она будет использована в программе или другой подпрограмме.

Структура любой подпрограммы аналогична структуре всей программы, т.е. включает заголовок и тело подпрограммы. В отличие от основной программы процедура завершается не точкой, а точкой с запятой. В свою очередь, тело подпрограммы состоит из раздела описаний (меток, констант, типов, переменных, процедур и функций, являющихся локальными по отношению к данной подпрограмме) и раздела операторов. В заголовке подпрограммы за ключевым словом (Function или Procedure) указываются имя подпрограммы, а в круглых скобках – список формальных параметров со своими описаниями. В отличие от процедуры в заголовке функции должен быть определен тип результата, передаваемого функцией.

Обращение к подпрограмме осуществляется с фактическими параметрами, которые должны соответствовать формальным по числу, типу и месту расположения.

4.2. Процедуры и функции

В Турбо Паскале выделяют два вида подпрограмм: процедуры и функции.

Отличие функции от процедуры заключается в том, что результатом исполнения операторов, образующих тело функции, всегда является некоторое единственное значение простого, строкового типа или указателя. Поэтому обращение к функции можно использовать в соответствующих выражениях наряду с переменными и константами. Другое отличие заключается в том, что в теле функции хотя бы один раз имени функции должно быть присвоено значение.

Чтобы нагляднее показать отличие функции от процедуры, запрограммируем выполнение одного и того же действия, а именно вычисление суммы элементов одномерного массива.

Программа с процедурой будет выглядеть следующим образом:

```
Program P1;
  Const M = 100;
 Type
    Vector = Array [1 .. M] of Integer;
 Var
    K, SumV : Integer;
    A : Vector;
 Procedure Inp_Vec(var Mas: Vector; N: Integer);
{Процедура ввода массива со списком формальных параметров.
Входной параметр: N типа Integer – размерность массива, не может превышать
100. Проверка N <= 100 - в основной программе.
Выходной параметр Mas типа Vector – исходный массив. }
   Var I: Integer;          {локальный параметр - параметр цикла}
   Begin
     Writeln('Введите ', N, ' целых чисел');
      For I := 1 to N do
           Read(Mas[I])
   End;
Procedure Sum_Vec (var Mas : vector; N : Integer; var Sum : Integer);
{Процедура вычисления суммы элементов массива
 входные параметры:
1) Mas типа Vector - массив, элементы которого необходимо суммировать.
2) N типа Integer - размерность массива .
Выходной параметр:
1) Sum типа integer - сумма элементов массива}
     Var I : Integer;
  Begin
    Sum:=0;
    For I:= 1 to N do
      Sum := Sum+Mas[I]
  End;
{Основная программа}
Begin
```

```
 Repeat
  Write('Введите размерность массива');
  Readln(K);
 Until (K>=1)And(K<=100);
 Inp_Vec(A , K);
 Sum_Vec (A, K, SumV);
 Writeln('Сумма элементов массива A = ', SumV);
End.
```

Указание. Если в качестве исходных данных в подпрограмму передается массив, то его следует передавать как параметр-переменная в целях экономии памяти, так как в этом случае при вызове подпрограммы не образуется вспомогательного массива.

Поэтому входной параметр Mas типа Vector описан в процедуре Sum_Vec как параметр-переменная.

Программа с функцией будет выглядеть следующим образом:

```
Program P2;
  Const M = 100;
 Type
    Vector = Array [1 .. M] of Integer;
 Var
    K : Integer;
    A : Vector;
 Procedure InpVec(var Mas: Vector; N: Integer);
{Процедура ввода массива со списком формальных параметров.
Входной параметр: N типа Integer – размерность массива, не может превышать
100. Проверка N <= 100 - в основной программе.
Выходной параметр Mas типа Vector – исходный массив. }
   Var I: Integer;          {локальный параметр - параметр цикла}
   Begin
     Writeln('Введите ', N, ' целых чисел');
      For I := 1 to N do
           Read(Mas[I])
   End;
Function Sum_V(var Mas : Vector; N : Integer) : Integer;
{Функция возвращает значение суммы элементов вектора;
возвращаемое значение типа integer;
входные параметры:
1) Mas типа Vector - вектор, элементы которого необходимо суммировать.
2) N типа Integer - размерность вектора, не может превышать 100}
 Var
  I, S : Integer;
 Begin
   S:=0;
   For I:=l to N do S:=S+Mas[I];
```

```
    Sum_V:=S;
  End;
{Основная программа}
Begin
 Repeat
  Write('Введите размерность массива');
  Readln(K);
 Until (K>=1)And(K<=100);
 InpVec(A , K);
 Writeln('Сумма элементов массива A = ', Sum_V(A, K));
   End.
```

Приведем теперь простые примеры использования подпрограмм-процедур и подпрограмм-функций.

Примеры решения задач с использованием процедур и функций

Пример 4.1. Составить программу для определения значений *n*!, *m*!, (*n*-*m*)!, используя функцию при вычислении факториала.

Решение.

Программа имеет вид:

```
Program Prim41;
Var
  N, M, NF, MF, NMF : Integer;
Function FACT(K : Integer) : LongInt;
     Var PF, I : Integer;
     Begin
       PF := 1;
       For I:=1 TO K Do
          PF := PF*I;
       FACT:= PF
     End;
Begin
  Write('Введите N и M');
  Readln(N, M);
  NF := FACT(N);
  MF:= FACT(M);
  NMF:= FACT(N-M);
  Writeln('HF= ',NF,'  HF= ', MF,'  NHF= ',NMF)
End.
```

В операторах присваивания программы записаны три обращения к функции FACT (N), FACT (M), FACT (N—M).

Пример 4.2. Составить программу для вычисления значения функции $y=ax^2+bx+d$, где

$$a=\sum_{i=1}^{n_t} t_i;\ b=\sum_{i=n_t+1}^{100} t_i;\ \ d=\sum_{i=1}^{20} q_i,$$

используя функцию SUM.

Решение.

Программа имеет вид:

```
Program Prim42;
Const N = 100;
Type
   INDEX = 1..N;
   VECT = Array[INDEX] Of Real;
Var
   I,  NT : Integer;
   T,Q : VECT;
   X,Y : Real;
Function SUM (MAS : VECT; K : Integer; MM : INDEX) : Real;
  Var J : Integer; S : Real;
  Begin
    S := 0;
    For J:= 1 To MM DO
      S := S+MAS[J];
   SUM := S
  End;
Begin
  Write('Введите Nt ');
  Readln(NT);
  For I:=1 To N DO Read(T[I]);
  For I:=1 TO 20 DO Read(Q[I]);
  Y:=SUM(T, 1, NT)*X*X + SUM(T, NT+1, N)*X + SUM(Q, 1, 20) ;
  Write('Y: = ', Y)
    End.
```

Указание. Несмотря на то что обрабатываемые массивы имеют разную длину, они описываются в программе как массивы одного и того же типа (VECTOR), так как при обращении к процедуре типы соответствующих формальных и фактических параметров должны совпадать.

Пример 4.3. Составить программу вычисления функции

$$Z=\frac{x^{k_1}x^{k_2}}{s_1+s_2},$$

где s_1 и k_1 — сумма и количество положительных элементов массива A(A_1, A_2, ..., A_{70}); s_2 и k_2 — сумма и количество положительных элементов массива B(B_1, B_2, ..., B_{40}). Для вычисления суммы и количества положительных элементов массива

использовать процедуру SUMKOL, в которой вычисляются s — сумма положительных элементов массива MAS и *k* — количество этих элементов.

Решение.

Программа имеет вид:

```
Program Prim43;
Const Rasm=100;
Type
    INDMAX = 1..Rasm;
    Vector = Array[INDMAX] Of Real;
Var
  NA, NB, I, K1, K2 : Integer;
  Z, S1, S2, X : Real;
  A, B : Vector;
Procedure InpVec(var Mas: Vector; N: INDMAX);
   Var I: Integer;
   Begin
     Writeln('Введите ', N, '  чисел');
      For I := 1 to N do
           Read(Mas[I])
   End;
Procedure SUMKOL(Var MAS : Vector; MM : INDMAX; Var S : Real; Var K : In-
teger);
{Процедура, в которой подсчитывается количество К и сумма S
положительных элементов массива Mas.
Входные параметры:
1)Mas – одномерный массив;
2)MM – размер массива (количество элементов массива).
Выходные параметры:
1)S – сумма положительных элементов массива Mas;
2)K – количество положительных элементов массива Mas.}
  Var J : Integer;
  Begin
    S := 0;
    K := 0;
    For J:= 1 To MM Do
      If MAS[J] > 0 Then
         Begin
           S := S + MAS[J];
           K := K + 1;
        End;
  End;
Function St(Y: Real; N : Integer) : Real;
{Функция, вычисляющая через умножение возведение вещественного числа Y
в целую степень N, т.е. Y^N}
 Var I        : Integer;
```

```
        P, AY : Real;
 Begin
  AY := ABS(Y);
  P := 1;
   For I := 1 To N Do P := P*AY;
  If Z >=0 Then St := P
                Else St := 1/P;
End;
Begin
  Write('Введите размер массива A ');
  Readln(NA);
  InpVec(A, NA);
  Write('Введите размер массива B ');
  Readln(NB);
  InpVec(B, NB);
  SUMKOL(A, NA, S1, K1);
  SUMKOL(B, NB, S2, K2);
  Write('Введите X: ');
  Readln(X);
  Z := St(X, K1)*St(X, K2) / (S1 + S2);
  Writeln('Z= ', Z);
End.
```

Пример 4.4. Написать программу, которая в первой вводимой с клавиатуры строке подсчитывает количество точек, а во второй – количество букв "я".

Решение.

Возможный вариант программы:

```
Program Prim44;
 Var
  St1, St2 : String;
 Function KOL(S : String; Sim : Char): Integer;
 Var I, K : Integer;
  Begin
   K := 0;
   For I := 1 To Length(S) Do
     If S[I] = Sim Then K := K + 1;
   KOL := K
  End;
 Begin
  Writeln('Введите первую строку');
  Readln(St1);
  Writeln('Введите вторую строку');
  Readln(St2);
  Writeln('Количество точек = ',KOL(St1, '.'));
  Writeln('Количество букв я = ',KOL(St2, 'я'))
```

```
End.
```

*Пример 4.5.*Вычислить значения многочлена

$$F(x) = 1.5\,x^7 - 2x^6 - 7x^5 + 3.4x^4 + 8x^3 - 0.5x^2 + 5.4x - 4$$

при x, изменяющемся от 0 до 1 с шагом 0.1, пользуясь схемой Горнера

$$F(x) == (\ldots((a_0x + a_1)\,x + a_2)\,x + a_3)\,x\,) + \ldots + a_n.$$

Значения аргумента x и функции F(x) занести в массивы X в Y соответственно и вывести на экран. Вычисление F(x) оформить в виде функции.

Решение.

Программа

```
Program Prim45;
 Type
  Vector = Array[0..7] Of Real;
 Const
  {Типизированная константа-массив}
  A : Vector = (1.5, -2, -7, 3.4, 8, -0.5, 5.4, -4);
 Var
  X, Y : Array[1..11] Of Real;
  I : Integer;
 Function Gorner(B : Vector; Z : Real; N : Integer):Real;
  {Параметры функции:
   B - массив коэфициентов полинома;
   Z - аргумент;
   N - степень полинома}
  Var
   J : Integer;
   S : Real;
  Begin
   S := B[0];
   For J := 1 To N Do
     S := S*Z + B[J];
   Gorner := S;
  End;
 Begin
  Writeln('X        F(X)');
  Writeln('_______________');
  For I := 1 To 11 Do
   Begin
   {Заполнение и вывод массивов X и Y}
    X[I] := 0.1*(I-1);
    Y[I] := Gorner(A, X[I], 7);
    Writeln(X[I]:4:2, Y[I]:9:4);
   End;
 End.
```

Контрольные вопросы

1. В каких случаях целесообразно применять процедуры и функции?
2. Какова структура описания процедур и функций?
3. Какой вид имеют заголовки процедуры и функции?
4. В чем состоит отличие описания функции от описания процедуры?
5. Можно ли для упорядочения по возрастанию массива вместо процедуры использовать функцию?
6. Какие параметры называются формальными и какие – фактическими?
7. Какие способы передачи параметров реализованы в Турбо Паскале?
8. Как происходит в программе обращение к процедуре?
9. Как происходит в программе обращение к функции?
10. Как и куда осуществляется выход из подпрограммы?

Задачи I уровня

В следующих задачах необходимо составить программу, содержащую функцию.

1. Вычислить сумму значений функций

$Y = F(a, b) + F(a^2, b^2) + F(a^2 - 1, b) + F(a - b, b) + F(a^2 + b^2, b^2 - 1)$,

где

$$F(u, t) = \begin{cases} u^2 + t^2, \text{ если } u > 0;\ t > 0; \\ u + t, \quad \text{если } u \le 0,\ t \le 0; \\ u - t, \quad \text{если } u > 0,\ t \le 0; \\ u + t, \quad \text{если } u \le 0,\ t > 0. \end{cases}$$

2. Вычислить сумму значений функций

$Y = F(\sin(a), a) + F(\cos(a), a) + F(\sin^2(a), a - 1) + F(\sin(a) - \cos(a), a^2 - 1)$,

где

$$F(u, t) = \begin{cases} u + \sin(t), \text{ если } u > 0; \\ u + t, \quad \text{если } u \le 0. \end{cases}$$

3. Вычислить сумму значений функций

$Y = F(\sin(a) + \cos(b), a + b) + F(\sin(a), \cos(b)) + F(a - b, a) + F(\sin^2(a) - 2, a)$,

где

$$F(u, t) = \begin{cases} u + t, \text{ если } u > 1; \\ u - t, \text{ если } 0 \le u \le 1; \\ t - u, \text{ если } u < 0. \end{cases}$$

4. Получить таблицу значений функции $y = e^x \sin x$ при x, изменяющемся от 0 до 2π с шагом $\pi/10$. Вычисление значений функции оформить в виде функции. Результаты представить в виде таблицы.

5. Получить таблицу значений функции $y = (\sin x)/x$ при x, изменяющемся от π до 2π с шагом $\pi/10$. Вычисление значений функции оформить в виде функции.

Результаты представить в виде таблицы.

6. Получить таблицу значений функции $y = \mathrm{tg}\, x$ при x, изменяющемся от $-\pi$ до π с шагом $\pi/5$. Вычисление значений функции оформить в виде функции. Результаты представить в виде таблицы.

7. Получить таблицу значений функции $y = 2x^2 + x + -4$ при x, изменяющемся от -5 до 5 с шагом 1. Вычисление значений функции оформить в виде функции. Результаты представить в виде таблицы.

8. Получить таблицу значений функции $y = \mathrm{sh}\, x$ при x, изменяющемся от -1 до 1 с шагом 0.1. Вычисление значений функции оформить в виде функции. Результаты представить в виде таблицы. Функция гиперболический синус определяется формулой $sh = \frac{e^x - e^{-x}}{2}$.

9. Получить таблицу значений функции $y = \mathrm{ch}\, x$ при x, изменяющемся от -1 до 1 с шагом 0.1. Вычисление значений функции оформить в виде функции. Результаты представить в виде таблицы. Функция гиперболический косинус определяется формулой $ch = \frac{e^x + e^{-x}}{2}$.

10. Получить таблицу значений функции $y = \mathrm{th}\, x$ при x, изменяющемся от -1 до 1 с шагом 0.1. Вычисление значений функции оформить в виде функции. Результаты представить в виде таблицы. Функция гиперболический тангенс определяется формулой $th = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

11. Два треугольника заданы длинами своих сторон a, b и c. Вычислить площади треугольников по формуле Герона и определить наибольший из них.

12. Два треугольника заданы координатами своих вершин $A(x_1, y_1)$, $B(x_2, y_2)$ и $C(x_3, y_3)$. Вычислить площади треугольников и определить, который из них больше. Площадь треугольника вычислять по формуле Герона:

$$S = \frac{\left|(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)\right|}{2}.$$

13. Футболист ударом ноги посылает мяч вертикально вверх с высоты 1 м с начальной скоростью 20 м/с. На какой высоте мяч будет через 1 , 3, 4 с? Движение мяча описывается зависимостью

$$y(t) = y_0 + v_0 t - \frac{gt^2}{2},$$

где y_0 — начальная высота; v_0 — начальная скорость; t- время. Оформить вычисление $y(t)$ в виде функции. Предусмотреть печать необходимой текстовой информации.

14. Вычислить приближенно площадь фигуры, ограниченной осью x, прямыми $x = 1$ и $x = 3$ и кривой $y(x) = 1/x + 5$, разделив интервал изменения x на 10 частей и суммируя площади получившихся прямоугольников, принимая их высоту, равную значению функции в середине каждого интервала. Вычисление площади прямоугольника оформить в виде функции.

15. Счастливым считается число N из шести цифр, в которой сумма левых трех цифр равна сумме правых трех цифр. Например, 457961 : 4+5+7= 9+6+1 = 16. Найти все счастливые числа в интервале от $N1$ до $N2$ и подсчитать их количество.

Выделение цифры в числе оформить в виде функции.

16. Найти все такие числа в интервале от 1 до 1000, которые содержатся в последних разрядах их квадрата, например: $5^2 = 25$, $25^2 = 625$. Выделение последних разрядов в числе оформить в виде функции.

17. Число Армстронга — число, состоящее из k цифр, у которых сумма k-х степеней его цифр равна самому числу. Например: $153 = 1^3+5^3+3^3$. Найти все числа Армстронга, состоящие из двух, трех и четырех цифр. Нахождение числа Армстронга оформить в виде функции.

Задачи II уровня

В следующих задачах необходимо составить программу, содержащую процедуру.

1. Для каждой из двух матриц сформировать одномерные массивы, составленные из минимальных элементов столбцов.

2. В каждую из двух матриц вставить заданные числа перед максимальным элементом каждой строки.

3. В каждую из двух матриц вставить заданные числа после минимального элемента каждого столбца.

4. Для каждой из двух матриц сформировать одномерный массив, составленный из максимальных элементов строки.

5. В каждую из двух матриц добавить по строке, в которой значения элементов равны количеству отрицательных элементов в соответствующих столбцах.

6. В каждую из двух матриц добавить по строке, в которой значения элементов равны суммам элементов соответствующего столбца.

7. В каждую из двух матриц добавить по столбцу, в котором значения элементов равны количествам положительных элементов в соответствующих строках.

8. Для каждой из двух матриц сформировать одномерные массивы, значения элементов которых равны суммам элементов в соответствующих строках.

9. Для каждой из двух матриц сформировать одномерные массивы, значения элементов которых равны суммам элементов в соответствующих столбцах.

10. Для каждой из двух матриц сформировать одномерные массивы, в которых значения элементов равны количествам отрицательных элементов в соответствующих строках.

11. Найти разность и произведение максимальных по модулю элементов двух матриц.

12. В каждую из двух матриц добавить по столбцу, в котором элементы равны суммам элементов соответствующих строк.

13. Найти среднее геометрическое и среднее арифметическое максимального и минимального элементов каждой из двух матриц.

14. Сформировать одномерные массивы из столбцов каждой из двух матриц, содержащих максимальные количества положительных элементов.

15. Найти среднее арифметическое минимальных элементов каждой из двух матриц.

16. Сформировать одномерные массивы из строк каждой из двух матриц, содержащих минимальные элементы этих матриц.

17. Для каждой из двух матриц сформировать одномерные массивы, в которых значения элементов равны количествам положительных элементов в соответствующих столбцах.

18. Максимальные элементы двух матриц поменять местами. Найти среднее арифметическое максимальных элементов строк каждой из двух матриц.

19. Сформировать одномерные массивы из отрицательных элементов каждой из двух матриц.

20. Проверить, является ли произведение матриц A и B перестановочным, т.е. проверить равенство A ⨯ B = B ⨯ A .

21. Вычислить суммы и количества элементов, находящихся в интервале от *p* до *q* для матриц A и B.

22. Для каждой из двух целочисленных матриц вывести на печать элементы, кратные трем.

23. Найти суммы элементов главной диагонали матриц, равных произведению A ⨯ B и B ⨯ A.

24. В каждой из двух матриц вычислить суммы элементов над главной диагональю.

Глава 5. СОРТИРОВКА И ПОИСК

5.1. Сортировка

Очень часто на практике встречаются задачи, в которых требуется переставить элементы неупорядоченного массива, после чего они становятся упорядоченными по возрастанию или убыванию. *Сортировка* элементов одномерного массива – это такая перестановка его элементов, в результате которой они оказываются упорядоченными определенным образом.

Процесс упорядочивания при большом объеме данных очень трудоемок, поэтому часто прибегают к помощи компьютеров.

Проблема сортировки остро встала в связи с широким внедрением ЭВМ в бизнес, где необходимо было обрабатывать огромные массивы числовой и текстовой информации. Технические средства внешней памяти тогдашних компьютеров - накопители на магнитной ленте (НМЛ) - обеспечивали только последовательную запись и чтение информации, поэтому необходимо было упорядочивать данные для того, чтобы обеспечить их быстрый поиск.

Современные устройства внешней памяти компьютеров - накопители на магнитных дисках (НГМД) - обеспечивают прямой доступ к записанной на них информации, что в некоторых случаях организации массивов данных позволяет обходиться без сортировки, но до сих пор она остается важным аспектом вычислительной техники.

Рассмотрим задачу упорядочивания в простейшей постановке: дан числовой массив $x_1, \ldots, x_n$, элементы которого попарно различны; требуется переставить элементы массива так, чтобы после перестановки они были упорядочены в порядке возрастания: $x_1 < \ldots < x_n$.

Способы сортировки

Существует несколько способов решения задачи сортировки. Каждый из них требует выполнения различного количества операций, а следовательно, времени расчета и памяти ЭВМ, необходимой для их реализации.

Способы сортировки делятся на прямые и улучшенные. Выделим следующие виды сортировок прямым способом:

- обменом ("пузырьковая сортировка");
- выбором (выделением);
- вставкой (включением).

Улучшенные способы сортировки основываются на тех же принципах, что и прямые, с использованием оригинальных идей и рационализации для ускорения процесса преобразования массива данных.

Сортировка обменом

Одной из наиболее простых (но и наиболее медленных) является так называемая *сортировка обменом пар.*

Основой алгоритма является сравнение двух элементов и их перестановка для получения порядка по возрастанию или убыванию значений элементов массива.

Для упорядочивания массива попарно различных чисел $x_1, ..., x_n$ может быть использован алгоритм "*пузырьковой" сортировки* (*сортировки обменами).* Опишем этот алгоритм применительно к упорядочиванию по возрастанию. На каждой итерации i=1, 2, 3, … N-1 последовательно сравниваются пары соседних элементов x_j и x_{j+1} , j=1, 2, 3, …, N-i, и если $x_j > x_{j+1}$, то они переставляются. Таким образом, после каждой итерации по i наибольший элемент оказывается на своем месте в конце массива.

Метод "пузырька" предполагает следующую физическую модель преобразования массива данных.

Из N чисел массива всплывает (как в бокале с шампанским) "пузырек"- число, оказывающееся больше в случае сортировки по возрастанию, чем $(N - 1)$ чисел. Затем всплывает "пузырек" - число, оказывающееся больше чем $(N - 2)$ оставшихся, но меньше, чем первый всплывший "пузырек" и т.д.

Рассмотрим массив из 6 элементов, содержащих следующие значения:

1 3 4 2 6 5.

Пройдем по массиву, сравнивая первый элемент со вторым, второй с третьим, и так далее до пятого. Будем проверять упорядоченность и, если она нарушена, т.е. если значение некоторого элемента больше значения следующего элемента, то поменяем эти значения местами. Например, дойдя до третьего элемента, значение которого равно 4, мы обнаружим, что оно больше значения четвертого элемента, равного 2. Выполнив обмен, получим новую последовательность значений:

1 3 2 4 6 5.

После этого перейдем к четвертому элементу. Теперь его значение равно 4. Значение следующего элемента равно 6, оно больше 4, поэтому обмена значениями не происходит. Перейдя к пятому, обнаруживаем, что его значение больше значения шестого, и производим обмен. Итак, выполнив один проход по массиву, получим последовательность значений:

1 3 2 4 5 6.

Эта последовательность до конца не отсортирована. Однако после первого прохода можем с уверенностью сказать, что наибольшее значение находится на своем месте - на правой границе массива. Выполнив еще один проход по первым пяти элементам массива, получим последовательность значений:

1 2 3 4 5 6

После этого прохода предпоследнее по величине значение 5 стало на свое место. Выполняя новые проходы, каждый раз сокращая количество просматриваемых элементов на 1, мы в конечном счете отсортируем все значения. Этот алгоритм можно реализовать с помощью двух циклов for - внешнего и внутреннего. Число повторений внешнего цикла будет на 1 меньше числа элементов массива (так как последовательность из одного значения проверять нет смысла). Число повторений внутреннего цикла будет последовательно сокращаться от *N*-1 до 1:

```
program sort;
{Программа сортировки одномерного массива в порядке возрастания
методом обмена пар ("пузырьковая" сортировка)}
const
   n = 12;
var
  a : array[1..n] of integer;
  x, i, j : integer;
begin
    {ввод массива}
     for i:=1 to n do
         begin
           write( ‘Введите ‘, i,’-е значение’);
           readln(a[i]);
         end;
   {сортировка}
     for i:=1 to n-1 do
        for j:=1 to n-i do
           if a[j]>a[j+1] then
             begin
            {перестановка элементов массива}
               x:=a[j];
              a[j]:=a[j+1];
              a[j+1]:=x;
            end;
{вывод отсортированных значений массива}
       for i:=1 to n do
          write(a[i]:5);
          readln; end.
```

Этот алгоритм нередко называют "пузырьковой" сортировкой, так как значения элементов как бы "всплывают" на свои места.

Алгоритм, сформулированный выше, — это, строго говоря, лишь один из алгоритмов сортировки обменом. Есть и другие алгоритмы, которые естественно отнести к алгоритмам сортировки обменом. Приведем пример такого алгоритма. Последовательными просмотрами чисел $x_1, ..., x_n$ найти наименьшее i такое, что $x_i >$

x_{i+1}. Поменять x_i и x_{i+1} местами и возобновить просмотр с начала массива. Когда не удается найти i, массив будет упорядочен.

Другой вариант алгоритма сортировки обменом. Предположим, что k - 1 элементов массива отсортированы. Последовательным просмотром чисел $x_k, ..., x_n$ найти такое число, что $x_k > x_i$, и поменять местами x_k и x_i. В результате такого просмотра чисел $x_k, ..., x_n$ на k-м месте окажется нужное число.

Сортировка выбором

Сортировку выбором называют еще *сортировкой поиском последовательных минимумов.*

Рассмотрим один из алгоритмов сортировки выбором.

Очевидно, что первое место в массиве должен занять наименьший элемент, второе — наименьший из всех остальных элементов и т. д. Пусть $x_1, ..., x_{i-1}$ уже получили нужные значения. Тогда определение индекса k наименьшего элемента из $x_i, x_{i+1}, ..., x_n$ и перестановка x_i с x_k приведут к тому, что $x_1, ..., x_i$ будут иметь нужные значения. Таким образом, схема программы решения поставленной задачи может быть следующей:

```
program vibor;
  описания;
begin
 ввести массив x;
          for i:=1 to n do
           begin
               найти индекс k наименьшего элемента среди xi, xi+1 ,.., xn;
                переставить xi с xk ;
                 writeln(xi)
           end
        end.
```

Как только элемент занял свое место, он сразу же выводится.

Приступим к детализации схемы. Ввод массива не доставляет затруднений:

```
for i:=1 to n do read(x[i])
```

и можно перейти к поиску индекса k наименьшего элемента среди $x[i], ..., x[n]$:

```
k:= i;
for j:= i+1 to n do
  if x[j] < x[k] then k:= j
```

Для перестановки x[i] с x[k] привлекаем дополнительную переменную v: v:=x[i]; x[i]:= x[k]; x[k]:=v.

Теперь можно написать программу (считаем элементы массива действительными числами):

```
program vibor;
const n=20;
type vect=array[1. .n] of real;
var  x : vect;  v : real;
      i, j, k : integer;
begin
 for i:=1 to n do read(x[i]);
```

```
for i:=1 to n do
begin
  k:= i;
 for j:= i+1 to n do
  if x[j] < x[k] then k:= j;
v:=x[i]; x[i]:= x[k]; x[k]:=v
writeln(x[i])
 end
 end.
```

При решении практических задач упорядочивание $x_1, ..., x_n$, как правило, сопровождается некоторыми дополнительными действиями. Например, если $x_1, y_1, x_2, y_2,..., x_n, y_n$—это значения аргумента x и некоторой функции $y=f(x)$, то перестановка $x_1,..., x_n$ должна сопровождаться перестановкой $y_1, ..., y_n$. Элементы $y_1, ..., y_n$ переставляются так же, как $x_1, ...,x_n$ вне зависимости от значений самих $y_1 ,.., y_n$. Рассмотрим эту задачу, переставленные значения $x_1, y_1, ..., x_n, y_n$ будут выведены в два столбца:

$$\begin{array}{cc} x_1 & y_1 \\ x_2 & y_2 \\ \multicolumn{2}{c}{\dots\dots\dots} \\ x_n & y_n \end{array}$$

Программа:

```
program tabl;
const n =20;
type vect=array[1. .n] of real;
var  x, y : vect;
       v : real;
        i, j, k : integer;
begin
for i:=1 to n do read(x[i], y[i]);
for i:=1 to n do
  begin
    k:= i;
   for j:=i+1 to n do
    if x[j] < x[k] then k:= j;
       v:=x[i]; x[i]:= x[k]; x[k]:=v;
       v:=y[i]; y[i]:=y[k]; y[k]:=v;
      writeln(x [i], y[i])
end
end.
```

Ниже приведен пример программы, в которой определяется максимальный из оставшихся элементов. Программа, реализующая этот алгоритм для сортировки по убыванию, выглядит следующим образом:

```
program sort1;
```

```
{Программа сортирует значения одномерного массива в порядке убывания методом выбора максимума)
const
  n = 20;
var
  a : array[ 1..n] of integer;
  i, j, max, i_max : integer;
begin
{ввод значений элементов массива}
  for i:=1 to n do
     read(a[i]);
{сортировка}
  for i:=1 to n-1 do
     begin
        max:=a[i];          {поиск максимального значения элемента}
        i_max:=i;           {среди значений элементов от}
       for j:=i+1 to n do {i-го до n-го}
             if a[j] > max then
              begin
                max:=a[j];
                i_max:=j;
              end;
       a[i_max]:=a[i];      {обмен i-го и максимального элементов}
       a[i]:=max;
end;
{вывод отсортированного массива}
   for i:=1 to n do
      write(a[i]:7);
end.
```

Сортировка вставкой

Для упорядочивания массива попарно различных чисел $x_1, ..., x_n$ может быть использован алгоритм *сортировки простыми вставками.* Опишем этот алгоритм применительно к упорядочиванию по возрастанию.

Массив данных разделяется на две части: отсортированную и неотсортированную. Элементы из неотсортированной части поочередно выбираются и вставляются в отсортированную часть так, чтобы не нарушить в ней упорядоченность элементов. В начале работы алгоритма в качестве отсортированной части принимается первый элемент массива.

Таким образом, будет иметь место (n - 1) проходов (где n - размерность массива), каждый из которых будет включать четыре действия:

- взятие очередного элемента из неотсортированной части и сохранение в дополнительной переменной;

- поиск позиции в отсортированной части массива для вставки элемента, взятого предыдущим действием, который бы не нарушил порядок;

- сдвиг элементов массива для вставки элемента массива в отсортированную часть;
- вставка элемента в найденную позицию.

Для реализации данного метода существует несколько алгоритмов, которые отличаются способом поиска позиции вставки.

Ниже приведена программа, реализующая один из алгоритмов сортировки простыми вставками.

```
{Сортировка вставкой}
Program SortInsetr ;
Uses Crt;
Const n=5;
Var        a :array [1..n] of byte;
b, i , j , k, p, c : byte ;
begin
ClrScr ;
Randomize;        {Инициализация генератора случайных чисел}
WriteLn('Исходный массив:');
{Генерация и печать исходного массива}
for i:=1 to n do
begin
a[i]:=Random(100) ;
Write(a[i]:3)
end;
{Внешний цикл сортировки}
for i:=2 to n do
begin
 b:=a[i];
{Взятие элемента массива из неотсортированной части и сохранение его в дополнительной переменной}
j:=1;   {Присвоение значения индексной переменной}
{Цикл поиска позиции вставки}
while b>a[j] do
j:=j+l;              {Фиксация позиции вставки}
{Цикл сдвига элементов массива для вставки}
for k:= i-1 downto j do
a[k+1]:=a[k] ;
{Вставка элемента массива в найденную позицию}
a[j]:=b;
end;
WriteLn('Отсортированный массив: ') ;
{Цикл печати отсортированного массива}
for i:=1 to n do Write(a[i]:3) ;

ReadKey
end.
```

Один из возможных результатов работы программы выглядит следующим образом:

Сортировка методом вставки
Исходный массив:
22 45 6 24 4
Отсортированный массив:
4 6 22 24 45

Алгоритм сортировки простыми вставками можно изменить следующим образом. Место, на которое надо вставить x_i в уже упорядоченную совокупность x_1, ..., x_{i-1}, определяется алгоритмом деления пополам (см. алгоритм поиска места элемента в п. 5.2). Получается новый алгоритм сортировки, который называется алгоритмом *сортировки бинарными вставками* (слова «бинарная вставка» следует понимать как «вставка делением пополам»).

По числу сравнений алгоритм сортировки бинарными вставками намного лучше, чем рассмотренные выше алгоритмы. Различными авторами предпринимались попытки доказательства оптимальности алгоритма сортировки бинарными вставками и даже печатно сообщалось о якобы найденном доказательстве. Позднее выяснилось, что и этот алгоритм не оптимален, так как он требует восьми сравнений для упорядочивания пяти чисел, а на самом деле достаточно семи сравнений.

Алгоритм быстрой сортировки

Алгоритм быстрой сортировки заключается в следующем. Отсортируем последовательность значений одномерного массива по возрастанию. Для этого возьмем некоторый элемент последовательности (назовем его контрольным). Пусть это будет, например, последний элемент. Далее разделим всю последовательность на две группы: в первую поместим те, которые предшествуют контрольному элементу, а во вторую - те, которые должны следовать за ним. Для этого будем сравнивать с контрольным элементы последовательности, начиная с первого, и до тех пор, пока не дойдем до элемента, значение которого больше контрольного. Запомним координату этого элемента в переменной After. Далее, двигаясь от правого конца последовательности, будем сравнивать элементы с контрольным до тех пор, пока не дойдем до элемента, меньшего контрольного или до элемента с индексом After. Запомним индекс элемента, на котором закончились сравнения, в переменной Before. Совпадение значений Before и After означает, что все элементы, стоящие до After, меньше контрольного, а элементы с номером After и после него - больше. Поменяв местами элементы After и контрольный, получим правильно разбитую на две части последовательность: в первой, слева от контрольного, стоят элементы, меньше его, а во второй, справа, - элементы, больше контрольного. Если переменные After и Before не совпадают, то это означает, что элемент с индексом After должен стоять после контрольного, а элемент с индексом Before - перед контрольным. Поменяем местами элементы Before и After и будем повторять процедуру нахождения элементов Before и After до тех пор, пока они не совпадут, т.е. пока не представится возможность получить последовательность, правильно разбитую на две группы.

Итак, мы разбили последовательность таким образом, что элементы, не превышающие контрольный, стоят слева от него, а остальные - справа. Следовательно, контрольный элемент находится на том месте, на котором он должен стоять в отсортированной последовательности. Далее, если в двух последовательностях, на которые мы разбили исходную, содержится более одного элемента, можно и их разбить на две группы, поставив контрольные элементы этих последовательностей на свои места. И так будем продолжать до тех пор, пока последовательность не будет отсортирована. Возможный вариант программы с процедурой быстрой сортировки может выглядеть, например, следующим образом:

```
Program Sort_Q;
Type
  Vector=Array [1..20] of Byte;
Var
  I, N : Byte;
  Vec : Vector;
Procedure Quick_Sort(var A:Vector; First, Last : Byte);
{Процедура быстрой сортировки массива.
Входные параметры:
1) A типа Vector - сортируемый одномерный массив;
2) First, Last типа Byte - индексы, указывающие
начальный и конечный элементы сортируемой последовательности}
  Var
    After, Before : Byte;
    Control, Work : Integer;
  Begin
 {обрабатываем массив, если он содержит не менее 2 элементов}
    If First < Last Then
      Begin
        Control:=A[Last];
        After:=First;
        Before:=Last;
        {разбиение массива на 2 группы}
        Repeat
        {поиск индекса первого элемента, большего или равного контрольному,  After}
          While A[After]<Control Do
            Inc(After);
        {поиск индекса элемента, меньшего контрольного во второй половине массива,
          Before}
          While (A[Before] >= Control) And (Before > After) Do
            Dec(Before);
        {если во второй половине массива такой элемент найден, поменять его местами
          с элементом After)
          If After < Before Then
            Begin
              Work:=A[Before];
```

```
          A[Before]:=A[After];
          A[After]:=Work;
        End;
      Until After=Before;
      A[Last]:=A[After];
      A[After]:=Control;
      {массив разбит правильным образом:
      слева от контрольного элемента находятся элементы, предшествующие ему, а
      справа - следующие за ним}
      Quick_Sort(A, First, After-1); {обработать левую подгруппу}
      Quick_Sort(A, After+1, Last);   {обработать правую подгруппу}
    End;
  End;
  Begin {Main}
    Write('Введите размерность массива (не более20)');
    Readln(N);
    For I:=1 To N Do
      Begin
       Write('Введите A[',I:2,']= ');
       Readln(Vec[I]);
      End;
    Quick_Sort(Vec, 1, N);
    Writeln('Отсотированный массив');
    For I:=1 To N Do
     Write (Vec[I]:4);
    Writeln;    End.
```

Процедура Quick_Sort является рекурсивной, так как в ней имеется обращение самой к себе. Рекурсия упрощает реализацию алгоритма быстрой сортировки. Этот способ сортировки является более эффективным, чем рассмотренные нами прямые способы сортировки, хотя и не так прост для понимания.

Рассмотрим, как будет осуществляться сортировка массива из 6 элементов при выполнении программы Sort_Q. В основной программе после операторов, предназначенных для ввода с клавиатуры значений элементов исходного массива, происходит вызов процедуры быстрой сортировки Quick_Sort. При этом обрабатываемой последовательностью элементов будет весь массив *A*, так как индекс начального элемента равен 1, индекс конечного элемента - *N* (*N*=6). При каждом обращении к процедуре Quick_Sort программа переходит на следующий уровень или возвращается на предыдущий, в зависимости от выполнения условия: индекс начального элемента (First) меньше индекса конечного элемента (Before), т.е. First < Before. Так как условие First < Before выполняется, то программа переходит на уровень 1.

Ниже приведены изменения исходного массива (3 1 5 6 2 4) на различных уровнях обработки. Элементы After и Before помечены соответственно символами *a* и *b*, а контрольный элемент - символом *c*.

Уровень 1. Последовательность 3 1 5 6 2 4

a *bc*
3 1 5 6 2 4
a *b c*
3 1 2 6 5 4
a *b c*
3 1 2 6 5 4
ab *c*
3 1 2 4 5 6
abc

Значения переменных After и Before совпадают. Это означает завершение выполнения действий уровня 1, в результате чего исходный массив будет разбит на две подпоследовательности элементов: левую и правую. В левой подпоследовательности находятся элементы, значения которых меньше значения контрольного элемента, а в правой – элементы, значения которых больше или равны значению контрольного элемента.

Вызов процедуры Quick_Sort(A, First, After-1) приведет к переходу на уровень 2. Так как фактические параметры процедуры First = 1, а After-1 = =4 - 1 = 3, то процедурой Quick_Sort будет обрабатываться последовательность, состоящая из 3 элементов и находящаяся слева от контрольного элемента, т.е. 3 1 2. Поэтому этот уровень обработки массива обозначим как 2.Л, , где символ Л означает обработку левой подпоследовательности. Ниже приводятся результаты ее обработки процедурой Quick_Sort.

Уровень 2.Л. Последовательность 3 1 2
a *bc*
3 1 2
a b c
1 3 2
a b c
1 3 2
ab c
1 2 3
abc

Совпадение значений After и Before означает завершение уровня 2.Л и переход на уровень 3.Л, так как опять происходит вызов процедуры Quick_Sort(A, First, After-1) обработки левой подпоследовательности элементов. Обрабатываемая подпоследовательность состоит только из 1 элемента, так как First = 1, а Last = After – 1 = 2 – 1 = 1.

Уровень 3.Л. Так как условие First < Last не выполняется, то никаких действий в процедуре Quick_Sort(A, First, After-1) не происходит и управление передается оператору вызова процедуры Quick_Sort(A, After+1, Last). Программа переходит на уровень 3.П. На этом уровне обрабатывается подпоследовательность, стоящая справа от контрольного элемента, полученного на уровне 2. В рассматриваемом примере эта подпоследовательность состоит только из одного элемента равного 3.

Уровень 3.П. При вызове процедуры Quick_Sort(A, After+1, Last) формальный параметр First получит значение First = After + 1 = 1 + 1 =2, а формальный параметр

- Last = 1. Условие First < Last не выполняется, и программа переходит на уровень 2.П, где обрабатывается подпоследовательность, стоящая справа от контрольного элемента, равного 4.

Уровень 2.П. Последовательность 5 6

a bc

5 6

abc

Уровень 3.Л. Процедурой Quick_Sort(A, First, After-1) обрабатывается последовательность, состоящая из одного элемента, равного 5, параметр First = Last, поэтому происходит переход на уровень 3.П.

Уровень 3.П. Так как обрабатываемая подпоследовательность на этом уровне состоит из одного элемента, то происходит возврат на уровень 2.

Обработка подпоследоватедьностей, стоящих слева и справа от контрольного элемента, полученного на уровне 1 и равного 4, закончена. Программа возвращается на уровень 1, где происходит выход из процедуры Quick_Sort в основную программу. Действие программы завершается выводом результата:

Отсортированный массив

1 2 3 4 5 6.

Контрольные вопросы

1. Зачем нужно сортировать массив данных?
2. Что такое сортировка данных?
3. Какие группы сортировок существуют? Чем они отличаются друг от друга?
4. Какие способы прямых сортировок Вы знаете?
5. Как выглядит алгоритм “пузырьковой” сортировки?
6. Как выглядит алгоритм сортировки выбором?
7. Как выглядит алгоритм сортировки вставкой?
8. В чем суть алгоритма сортировки бинарными вставками?

Задачи

1. Дан одномерный массив X, состоящий из N элементов, N – заданное натуральное число. Упорядочить элементы массива по неубыванию методом “пузырька”.

2. Дан одномерный массив X, состоящий из N элементов, N – заданное натуральное число. Упорядочить элементы массива по невозрастанию методом вставки.

3. Дан одномерный массив A, состоящий из N элементов, N – заданное натуральное число. Упорядочить элементы массива по неубыванию методом выбора.

4. Даны целые $x_1, \ldots, x_n$. Получить в порядке возрастания все различные числа, входящие в $x_1, \ldots, x_n$. Воспользоваться алгоритмом сортировки бинарными вставками. В процессе сортировки следует отбрасывать элементы, уже встречавшиеся раньше. Если в результате поиска места x_i в упорядоченной совокупности

$x_1, ..., x_k$ ($k < i$) обнаружится, что среди $x_1, ..., x_k$ есть элемент, равный x_i, то следует перейти к рассмотрению x_{i+1} без изменения $x_1, ..., x_k$.

5. Даны действительные попарно различные $a_1, ..., a_n$. Получить попарно различные целые $j_1, ..., j_n$, такие, что $1 \le j_k \le n$ (k=1, ..., n) и $a_{j_1} < a_{j_2} < ... < a_{j_n}$.

6. Даны целые $a_1, ..., a_n$. Найти наибольшее значение, содержащееся в последовательности чисел $a_1, ..., a_n$ после выбрасывания из нее:

а) одного из членов со значением max ($a_1, ..., a_n$);

б) всех членов со значением max ($a_1, ..., a_n$).

7. Даны натуральные $n, a_1, ..., a_n$ ($n > 4$). Числа $a_1, ..., a_n$ — это измеренные в сотых долях секунды результаты n спортсменов в беге на 100 м. Составить команду из четырех лучших бегунов для участия в эстафете 4×100, т. е. указать одну из четверок натуральных чисел i, j, k, I такую, что $1 \le i < j < k < l \le n$ и сумма $a_i + a_j + a_k + a_l$ имеет наименьшее значение.

8. Задан одномерный массив строк из N элементов. Отсортировать его элементы по алфавиту.

9. Радиостанция провела опрос 120 слушателей. Каждому был задан вопрос: “За кого из 13 политиков Вы намерены голосовать на предстоящих выборах?“ Составить программу получения 5 наиболее часто встречающихся ответов и их долей (в %), результат вывести в виде таблицы.

10. Составить программу для обработки результатов кросса на 500 м для женщин. В кроссе участвует не более 100 студенток. В протоколе указывается фамилия, шифр группы, результат. Получить итоговую таблицу результатов спортсменов в порядке занятых ими мест. Определить количество участниц, выполнивших заданный норматив.

11. Лыжные гонки проводятся отдельно для двух групп участников (в каждой группе не более 15 человек). Протокол соревнований каждой группы включает в себя фамилии участников и результат. Объединить результаты в обеих группах, упорядочить фамилии спортсменов в порядке занятых ими мест и вывести в виде таблицы с заголовком.

12. На конкурсе песни каждый зритель должен назвать лучшую по его мнению песню и ее исполнителя. Обработать результаты опроса, упорядочив их по количеству голосов и вывести в виде таблицы информацию о первых десяти песнях, возглавляющих список (песня, исполнитель, количество голосов).

13. Результаты соревнований по прыжкам в длину определяются по сумме двух попыток. В протоколе для каждого участника указываются фамилия, общество, результаты первой и второй попыток. Вывести протокол в виде таблицы с заголовком с фамилиями участников соревнований в порядке занятых мест.

5.2 Поиск

При подготовке таблиц статистических сводок, справочных изданий, словарей, как правило, производят упорядочивание данных. Это делает работу с таблицей, справочником и тому подобными более удобной. Например, выигравшие номера упорядочены в лотерейных таблицах по возрастанию. Упорядоченность облегчает проверку билета: если бы номера не были упорядочены, то проверка могла бы по-

требовать сравнения номера билета со всеми номерами, содержащимися в таблице. Точно так же словарный порядок облегчает поиск слова в словаре или справочнике.

Поиск сведений в разнообразных таблицах и сводках может быть возложен на компьютер. Мы будем заниматься алгоритмом быстрого поиска элемента в упорядоченной совокупности $a_1, ..., a_n$. При этом будем считать, что $a_1, ..., a_n$ - целочисленный массив, b - некоторое целое число. Работая со словарем, мы имеем дело не с целыми числами, а со словами, но это не играет принципиальной роли: меняется только техника каждого сравнения, а не число сравнений.

Пусть $a_1<...<a_n$. Рассмотрим задачу: выяснить, входит ли данное число b в массив $a_1, ..., a_n$, и если входит, то каково значение p, для которого $a_p=b$? Эту задачу мы назовем задачей поиска элемента. Тривиальный алгоритм решения этой задачи основывается на последовательных сравнениях b с элементами $a_1, ..., a_n$: сравнить b с a_1, если они равны, то p=1, иначе сравнить b с a_2 и т. д. Среднее число требуемых здесь сравнений можно считать равным $n/2$. Известен алгоритм, который требует гораздо меньших затрат.

Предположим, что в массиве $a_1, ..., a_n$ имеется элемент, равный b, т. е. существует такое p, что $a_p = b$. По результату любого сравнения $a_s < b$ ($1 \le s \le n$) мы сразу определяем, лежит ли p в диапазоне от 1 до s или же в диапазоне от $s + 1$ до n: второе будет иметь место, если неравенство $a_s < b$ справедливо, а первое—если несправедливо. Если само s находится примерно посередине между 1 и n, то сравнение $a_s < b$ сужает диапазон поиска примерно вдвое. Этот прием можно использовать многократно. Получается алгоритм, называемый алгоритмом деления пополам. В соответствии с этим алгоритмом надо взять первоначально 1 и n в качестве границ поиска индекса элемента; далее, до тех пор, пока границы не совпадут, шаг за шагом сдвигать их следующим образом: сравнить b с a_s, где s—целая часть среднего арифметического границ; если $a_s < b$, то заменить прежнюю нижнюю границу на $s + 1$, оставив верхнюю границу без изменения, иначе оставить без изменения нижнюю границу, а верхнюю заменить на s. Поиск закончится, когда границы совпадут.

Сказанное можно записать в виде последовательности операторов Паскаля (p и q обозначают упомянутые верхнюю и нижнюю границы; когда p и q совпадут, выполнение алгоритма закончится, и p дает результат выполнения);

```
p:=1; q:= n;
while p < q do
  begin
    s: =(p+q) div 2;
    if a[s] < b then p:= s+1
                else q:= s
 end;
```

Вообще идея деления пополам (сведение задачи к примерно вдвое более простой в количественном отношении) оказывается весьма плодотворной для построения многих алгоритмов. В научной литературе деление пополам иногда именуется греческим термином «дихотомия» или латинским «бисекция», а для обозначения сущности самой идеи иногда используются слова «разделяй и властвуй».

Заметим сразу следующее. Мы исходим из предположения, что среди элементов $a_1, ..., a_n$ имеется такой, который равен b. Если заранее неизвестно, имеется или нет такой элемент, то, получив p, надо дополнительно проверить, действительно ли $a_p=b$. И если обнаружится, что равенство несправедливо, то из этого будет следовать, что среди $a_1, ..., a_n$ нет элемента, равного b.

Продемонстрируем применение этого алгоритма на конкретном примере. Пусть $a_1, ..., a_{10}$ соответственно равны 2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Тогда для b = 19 поиск будет разворачиваться следующим образом. В начале границы поиска равны 1 и 10. Имеем [(1 + 10) / 2] = 5, неравенство $a_5 < b$, т.е. 11 < 19, справедливо, поэтому переходим к границам 6, 10. Имеем [(6 +10) / 2] = 8, неравенство $a_8 < b$, т.е. 19 < 19, несправедливо, поэтому переходим к границам 6, 8. Имеем [(6+8)/2] = 7, неравенство $a_7 < b$, т. е. 17 < 19, справедливо, поэтому переходим к границам 8, 8. Эти границы совпадают; следовательно, если в массиве вообще есть элемент со значением b, то это a_8. В данном случае равенство $b = a_8$ справедливо, так как 19 = 19.

Нетрудно убедиться, что если бы мы взяли в качестве b не 19, а 18, то до самого последнего момента применение алгоритма разворачивалось бы точно так же, но в самом конце дополнительная проверка равенства $a_8 = b$ дала бы отрицательный результат, что явилось бы свидетельством того, что в массиве $a_1, ..., a_n$ нет элемента со значением 18.

Алгоритм поиска элемента делением пополам обладает высоким быстродействием. Если $n=2^m$, то число сравнений b с элементами $a_1, ..., a_n$ в процессе применения этого алгоритма не превзойдет m, а если n не равно 2^m ни для какого целого m, то в качестве границы для числа сравнений можно взять наименьшее m такое, что $n < 2^m$ (без учета дополнительного сравнения $a_p=b$, необходимого в том случае, когда заранее неизвестно, имеется или нет среди $a_1, ..., a_n$ элемент, равный b).

Например, если n = 10, то число сравнений не превосходит 4, так как $10 < 2^4$, для n = 32 это число не превосходит 5, так как $32 = 2^5$, для n = 1000 это число не превосходит 10, так как $1000 < 2^{10}$ и т. д.

Теперь мы приведем пример программы, использующей написанную выше последовательность операторов, т.е. алгоритм деления пополам.

Пусть таблица выигрышей лотереи представлена в виде двух массивов $a_1, ..., a_n$ и $c_1, ..., c_n$ (n - некоторая константа) так, что натуральные $a_1, ..., a_n$ — это выигравшие номера ($a_1 < ... < a_n$), а $c_1, ..., c_n$ - действительные положительные числа, означающие выигрыши в рублях, выпавшие соответственно на номера $a_1, ..., a_n$. Требуется найти выигрыши, выпавшие на ряд номеров (если номера нет в таблице, то выигрыш считается равным нулю). Схема программы:

```
program Loto;
описания;
begin
ввод массивов a и c;
1: write(‘ номер='); read(b);
найти p - предполагаемое место b в массиве a;
if a[p]=b then f:= c[p] else f:=0;
writeln(' выигрыш =',f); goto 1
```

```
end.
```

Мы легко можем детализировать эту схему и получить настоящую программу. Условимся, что значения элементов массивов *a* и *c* заданы в следующей последовательности: a_1, c_1, a_2, c_2,... a_n, c_n, где *n*—константа, значением которой будем считать 20:

```
program Loto;
      const n =20;
      type u=array[1 .. n] of integer;
           v=array[1.. n] of real;
     var a:u;
         c:v;
        i, p, q, s, b:integer;
       f:real;
    begin
     for i:=1 to n do read(a[i], c[i]);
     write('НОМЕР='); read(b);
      p:=1; q:=n;
     while p < q do
      begin
        s:=(p+ q) div 2;
        if a[s] < b then p:= s+1
                    else q:=s
      end;
   if a[p]=b then f:=c[p]
             else f:= 0;
  writeln('выигрыш=' ,f);
end.
```

Идея деления пополам («разделяй и властвуй») может быть положена в основу алгоритма решения еще одной задачи, похожей на задачу поиска элемента. Пусть по-прежнему $a_1 < a_2 < ... < a_n$, и пусть *b*—некоторое число. Для числа *b* имеется *n* + 1 возможность: $b \le a_1$, $a_1 < b \le a_2$, $a_2 < b \le a_3$, ..., $a_{n-1} < b \le a_n$, $a_n < b$.

Требуется определить, какая из возможностей имеет место. Ответом должно быть одно из чисел 1, 2, **...**, *n*, *n*+1 (порядковый номер возможности).

По результату любого сравнения $a_s < b$, $1 \le s \le n$ мы сразу определяем, лежит ли ответ в диапазоне от 1 до *s* или же в диапазоне от *s*+1 до *n*+1. Если само *s* находится примерно по середине между 1 и *n*+1, то сравнение $a_s < b$ сужает диапазон поиска примерно вдвое. Это приводит к следующему алгоритму:

```
p:=1; q:=n+1;
while p < q do
begin
  s: =(p+q) div 2;
  if a[s] < b then p:=s+1
              else q:=s
```

end;

Единственное отличие последнего алгоритма от алгоритма поиска элемента состоит в начальном присваивании q:=n+1 (вместо q:=n*)*. Этот алгоритм удобен, например, когда надо вставить в таблицу новый элемент, не нанося при этом ущерба упорядоченности таблицы. Этот алгоритм называется алгоритмом поиска места элемента (до него мы рассматривали алгоритм поиска элемента: названия похожи, но не совпадают).

Контрольные вопросы

1. Какие методы поиска данных в массиве Вы знаете?
2. В чем состоит алгоритм прямого перебора?
3. Как выглядит алгоритм бинарного поиска?
4. Как должен быть преобразован массив для организации бинарного поиска?
5. Как определяется быстродействие алгоритма бинарного поиска?

Задачи

1**.** Изменить программу *Loto:* как только в качестве номера (т.е. значения переменной *b)* с клавиатуры указывается отрицательное число, выполнение программы должно заканчиваться.

2**.** Изменить программу *Loto:* не нужно выводить все выигрыши по отдельности, потребуется вывести суммарный выигрыш, выпавший на все указанные номера. В качестве сигнала об окончании последовательности номеров используется любое отрицательное число (см. предыдущую задачу).

3. Изменить программу *Loto,* предполагая, что номера в таблице заданы в порядке убывания: $a_1 > a_2 > ... > a_n$.

4**.** Даны натуральные a_1, ..., a_{50}, m, b_1, ... b_m, ($a_1 > a_2 > ... > a_{50}$). Подсчитать количество тех b_i , $1 \le i \le m$, для которых нет равных среди a_1, ..., a_{50}.

5. Составить программу определения задуманного человеком числа от 1 до 1000 с помощью 10 вопросов. Каждый вопрос имеет вид: “Верно ли, что задуманное число больше *k*?” При этом указывается конкретное *k.* Ответы человека—это Д и Н. Применить идею деления пополам (“разделяй и властвуй”).

6. Исследовать предложенную в предыдущей задаче игру в угадывание задуманного числа. Указать количество вопросов, которое при «экономном» ведении игры будет достаточным для угадывания числа от 1 до 2000, 3000, 4000, 5000.

7. Даны целые a_1, ..., a_n, b (*n*—некоторая константа, $a_1 < a_2 < \ < a_n$). Если среди чисел a_1, ..., a_n есть равное b, то оставить a_1, ..., a_n без изменений. В противном случае добавить b к a_1, ..., a_n без нарушения упорядоченности по возрастанию.

8. Написать программу поиска места элемента, предполагая, что числа a_1, **...,** a_n упорядочены по убыванию: $a_1 > a_2 > ... > a_n$.

9. В случае, если среди a_1, **...** a_n допускаются равные ($a_1 \le a_2 \le ... \le a_n$), то алгоритм поиска места элемента, записанный в виде последовательности операторов (см. с. 141), позволяет узнать самое первое из мест, на которое может быть

вставлено число b в $a_1, ..., a_n$ без нарушения упорядоченности по неубыванию. Изменить последовательность операторов так, чтобы указывалось последнее из этих мест.

ЗАКЛЮЧЕНИЕ

В настоящем учебном пособии рассмотрены основные сведения об алгоритмах, наиболее часто используемых типов данных, а также наиболее распространенных операциях над значениями этих типов. В него не вошел материал, связанный с изучением динамических структур данных, методов и средств объектно-ориентированного программирования, что является предметом отдельных изданий.

Используемый в пособии учебный материал, в том числе и задачи для самостоятельного решения, подобраны с учетом того, что дисциплина “Программирование и основы алгоритмизации” изучается на первом - втором курсах и, как правило, на этом этапе обучения уровень подготовки студентов в данной области очень разный.

Полученные знания будут использованы студентами в специальных дисциплинах при разработке алгоритмов и программ, реализующих процессы обработки цифровой информации, особенно при управлении техническими объектами.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Абрамов В.Г, Трифонов Н.П., Трифонова Г.Н.* Введение в язык Паскаль. - М.: Наука, 1988. – 320 с.

2. *Боон К.* Паскаль для всех. — М.: Энергоатомиздат, 1988. – 188 с.

3. *Вирт Н.* Алгоритмы и структуры данных. — М.: Мир, 1989. - 360 с.

4. Вычислительная техника и программирование. Практикум по программированию: Практ. пособие/ В.Е. Алексеев, А.С. Ваулин, Г.Б. Петрова; Под ред. А.В. Петрова. – М.: Высш. шк., 1991. – 400 с.

5. *Грогоно П.* Программирование на языке Паскаль.— М.: Мир, 1982. – 382 с.

6. *Довгаль С.И., Литвинов Б.Ю., Сбитнев А.И.* Персональные ЭВМ: Турбо Паскаль V7.0, объектное программирование, локальные сети. — Киев: Информсистема сервис, 1993. – 480 с.

7. *Джонс Ж., Харроу К.* Решение задач в системе Турбо Паскаль. - М.: Финансы и статистика,1991. – 720 с.

8. *Епанешников А.М., Епанешников В.А.* Программирование в среде Turbo Paskal 7.0. - М.: ДИАЛОГ-МИФИ, 1997. – 288 с.

9. *Марченко А.И.* Программирование в среде Borland Pascal 7.0. — Киев: Бином Универсал: ЮНИОР, 1998. – 506 с.

10. Офицеров Д.В., Старых В.А. Программирование в интегрированной среде Турбо Паскаль. - Минск, Беларусь, 1992. – 240 с.

11. *Перминов О.Н.* Программирование на языке Паскаль. — М.: Радио и связь, 1988. – 224 с.

12. *Пильщиков В.Н.* Сборник упражнений по языку Паскаль: Учеб. пособие для вузов. – М.: Наука. 1989. – 160 с.

13. *Прайс Д.* Программирование на языке Паскаль: практическое руководство. — М.: Мир, 1987. – 230 с.

14. *Фаронов В.В.* Программирование на персональных ЭВМ в среде Турбо Паскаль. — М.: Издательство МГТУ, 1992. – 443 с.

15. *Фаронов В.В.* Турбо Паскаль 7.0: Практика программирования. – М.: Нолидж, 1997. – 429 с.