№ 26 (219) июль 1999

ИНФОРМАТИКА

еженедельное приложение
к газете «ПЕРВОЕ СЕНТЯБРЯ»

Практикум по Турбо Паскалю

И.А. Бабушкина,
Н.А. Бушмелева,
С.М. Окулов,
С.Ю. Черных

ВЫПУСК 2

Содержание*

* Книга выходит в нескольких номерах нашей газеты. Номера страниц указаны у разделов, вошедших в данный номер.

2. ОСНОВЫ ПАСКАЛЯ

2.1. ПРОЦЕДУРЫ И ФУНКЦИИ

Нередко в программах встречаются повторяющиеся или похожие фрагменты. Имеется возможность оформлять такие фрагменты специальным образом — выделять их в *подпрограммы*. Подпрограмме дается имя, по которому к ней можно обращаться (вызывать подпрограмму). Использование подпрограмм не только улучшает структуру и внешний вид программы, но и уменьшает вероятность ошибок и облегчает отладку.

В Паскале имеется два вида подпрограмм — процедуры и функции. Их структура очень похожа на структуру основной программы.

2.1.1. Описание процедуры. Оператор процедуры

Повторение

1. Из каких разделов состоит программа?
2. Является ли заголовок программы обязательным?
3. Какие разделы описаний вы уже знаете?
4. Каким служебным словом начинается раздел описания констант? Приведите пример описания константы.
5. Каким служебным словом начинается раздел описания типов? Приведите пример описания типа.
6. Каким служебным словом начинается раздел описания переменных? Приведите пример описания переменных.
7. Как оформляется тело основной программы?
8. Какие операторы вы знаете?

Описание процедуры

Описание процедуры начинается с заголовка, который является обязательным (в отличие от заголовка программы). Заголовок состоит из служебного слова **Procedure**, за которым следуют имя процедуры и, в круглых скобках, список формальных параметров. В конце заголовка ставится точка с запятой. После заголовка могут идти те же разделы, что и в программе. Общий вид описания процедуры (в квадратные скобки заключена часть, которая может отсутствовать):

```
Procedure Имя [(Список формальных
                параметров)];
  Описательная часть
Begin
    Тело процедуры
End;
```

При вызове процедуры ее формальные параметры заменяются соответствующими фактическими.

Фактические параметры — это параметры, которые передаются процедуре при ее вызове.

Количество и типы формальных и фактических параметров должны в точности совпадать.

Формальные параметры описываются в заголовке процедуры и определяют тип и место подстановки фактических параметров. Формальные параметры делятся на два вида: параметры-переменные и параметры-значения.

Параметры-переменные отличаются тем, что перед ними стоит служебное слово **Var**. Они используются тогда, когда необходимо, чтобы изменения значений формальных параметров в теле процедуры приводили к изменению соответствующих фактических параметров.

Параметры-значения отличаются тем, что перед ними слово **Var** не ставится. Внутри процедуры можно производить любые действия с параметрами-значениями, но все изменения никак не отражаются на значениях соответствующих фактических параметров, то есть какими они были до вызова процедуры, такими же и останутся после завершения ее работы.

Все переменные программы делятся на глобальные и локальные. *Глобальные переменные* объявляются в разделе описаний основной программы. *Локальные переменные* объявляются в процедурах и функциях. Таким образом, локальные переменные "живут" только во время работы подпрограммы.

Пример

Составить программу для вычисления a^n: целые числа a и n ($n \geq 0$) вводятся с клавиатуры.

Решение

Составим процедуру для вычисления степени целого числа.

```
Procedure Degree(x,y: Integer;
          Var st: Longint);
Var i:Integer;      {описательная часть}
Begin          {тело процедуры}
  st:=1;
  For i:=1 To y Do st:=st*x;
End;
```

Первая строчка описания — это заголовок процедуры, который начинается со слова **Procedure**. Процедура названа именем Degree. В скобках записан список формальных параметров, то есть перечислены переменные с указанием их типа. Мы используем три параметра: первый — основание степени, то есть число, которое надо возвести в степень; второй — показатель степени, третий — результат. Первые два формальных параметра — параметры-значения, третий — параметр-переменная, и перед ним

указано слово **Var**. Все они описаны как целые (x и y — переменные типа Integer, а st — типа Longint, так как степенная функция быстро возрастает).

После заголовка процедуры идут разделы описаний. В нашем примере имеется только раздел описания переменных, в котором описывается одна переменная i (счетчик цикла).

Далее идет тело процедуры. Оно начинается служебным словом **Begin** и заканчивается служебным словом **End**, после которого стоит точка с запятой (в конце программы после последнего **End** ставится точка). В теле процедуры вычисляется степень числа *x* с помощью цикла For.

В программе процедуры и функции описываются после раздела описания переменных программы, но до начала ее основной части, то есть до **Begin**, начинающего эту часть.

Вся программа для решения нашей задачи может иметь следующий вид:

```
Program Example_28;
Var  a, n: Integer;
     s: Longint;
Procedure Degree(x,y: Integer; Var st:
Longint);
Var i:Integer;
Begin
  st:=1;
  For i:=1 To y Do st:=st*x;
End;
Begin
  Writeln('Введите два числа - основание
          и показатель степени');
  Readln(a,n);
  Degree(a,n,s);{обращение к процедуре}
  Writeln('Результат ',s);
  Readln;
End.
```

Процедура вызывается как оператор, состоящий из имени процедуры. В круглых скобках записываются фактические параметры. В нашем примере формальные параметры x, y и st принимают значения фактических параметров a, n и s соответственно. После завершения работы процедуры переменные a и n сохранят те же значения, что и при вызове, а s получит новое значение.

Пример

Даны две целые переменные. Поменять местами их значения.

Решение

Поменять местами значения двух переменных можно двумя способами — через промежуточную переменную или без нее (см. раздел 1.1.2). Напишем процедуру, соответствующую первому способу.

```
Procedure Swap (Var x, y:Integer);
Var z: Integer;
Begin
  z:=x; x:=y; y:=z;
End;
```

Процедура называется Swap. У нее имеется два формальных параметра, которые являются параметрами-переменными, так как необходимо поменять значения переменных и запомнить изменения. Эти параметры являются результатами выполнения процедуры.

В процедуре описана переменная z, которая используется как промежуточная.

Вся программа имеет вид:

```
Program Example_29;
Var a, b: Integer;
Procedure Swap (Var x, y:Integer);
Var z: Integer;
Begin
  z:=x; x:=y; y:=z;
End;
Begin
  Writeln('Введите значения переменных a и b');
  Readln(a, b);
  Swap(a, b);   {обращение к процедуре}
  Writeln('a=',a,' b=',b);
          {вывод новых значений}
  Readln;
End.
```

Решение задач

1. Используя процедуру для вычисления степени числа, найти значение выражения:

$$y = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0.$$

Коэффициенты a_4, a_3, a_2, a_1, a_0 и x вводятся с клавиатуры.

2. Используя процедуру, рассмотренную в примере 2, упорядочить по возрастанию значения трех переменных — a, b и c.

3. Даны координаты трех вершин треугольника. Найти длины всех его сторон.

4. Дано натуральное число. Найти все его делители и подсчитать их количество.

5. Даны два натуральных числа. Определить, является ли одно число перевертышем другого.

6. Даны координаты трех вершин треугольника *ABC* и точки *D*. Определить, является ли точка *D* внутренней точкой треугольника.

2.1.2. Функции

Повторение

1. Что такое подпрограмма? Для чего используются подпрограммы? Перечислите преимущества использования подпрограмм.

2. Запишите структуру процедуры.

3. Что такое формальные и фактические параметры?

4. Как записывается вызов процедуры?

Описание

Заголовок функции состоит из слова **Function**, за которым указывается имя функции, затем в круглых скобках записывается список формальных параметров, далее ставится двоеточие и указывается тип результата функции.

В теле функции обязательно должен быть хотя бы один оператор присваивания, в левой части которого стоит имя функции, а в правой — ее значение. Иначе значение функции не будет определено.

Таким образом, общий вид описания функции следующий:

```
Function Имя[(Список формальных параметров)]:
        Тип результата;
  Описательная часть
Begin
    Тело функции, в котором обязательно
    должно быть присваивание
    Имя_функции:=значение
End;
```

Пример

Составить программу, подсчитывающую число сочетаний без повторения из n элементов по k.

Число сочетаний без повторения вычисляется по формуле: $C_n^k = \frac{n!}{k!(n-k)!}$.

Обозначим через n и k переменные для хранения введенных чисел; C — переменную для хранения результата.

Чтобы подсчитать количество сочетаний без повторения, необходимо вычислить $n!$, $(n-k)!$, $k!$

Опишем функцию для вычисления факториала числа n ($n! = 1 \cdot 2 \cdot \ldots \cdot n$).

```
Function factorial(n:Integer):Longint;
{заголовок функции}
Var i: Integer; {описательная часть}
    rez: Longint;
Begin          {тело функции}
  rez:=1;
  For i:=1 To n Do rez:=rez*i;
  factorial:=rez;    {присваивание
                  значения имени функции}
End;
```

Первая строчка в описании функции — это ее заголовок. Служебное слово **Function** (функция) указывает на то, что именем factorial названа функция. В скобках записан список формальных параметров функции, состоящий из одной переменной целого типа. Далее в заголовке указан тип значения функции. В данном примере результат функции factorial — длинное целое число.

За заголовком функции следует описательная часть функции, которая, как и у программы, может состоять из разделов описания переменных, констант, типов. В данном примере имеется только раздел описания переменных. В нем описаны переменные i (счетчик цикла) и rez (для накопления значения факториала).

Далее идет раздел операторов (тело функции). Результат присваивается имени функции, таким образом функция получает свое значение.

В тексте программы описания функций всегда следуют за разделом описания переменных и до начала основной части, как и описания процедур. После того как функция описана, ее можно использовать в программе.

```
Program Example_30;
Var n,k: Integer;
a1, a2, a3, c: Longint;
Function factorial(n:Integer):Longint;
Var i: Integer;
    rez: Longint;
Begin
  rez:=1;
  For i:=1 To n Do rez:=rez*i;
  factorial:=rez;
End;
Begin
  Writeln('Ввод n и k:');
  Readln(n,k);
  a1:=factorial(n); { вычисление n!}
  a2:=factorial(k); {вычисление k!}
  a3:=factorial(n-k); {вычисление (n-k)!}
  c:=a1 div (a2*a3); {результат }
  Writeln(c);
  Readln;
End.
```

Пусть n=5, k=3. Когда в программе встречается оператор a1:=factorial(n), выполняются следующие действия:

- выделяется память для переменных, описанных в функции factorial;
- формальному параметру присваивается значение фактического: n:=n (n=5);
- выполняется функция, вычисляется факториал числа 5;
- значение функции передается в место обращения к этой функции, то есть присваивается переменной a1.

В операторах
a2:=factorial(k) и a3:=factorial(n-k)
еще дважды вызывается функция factorial с параметрами k=3 и n-k=2. Всего в программе имеется 3 обращения к функции factorial, столько же раз выполняются и описанные выше действия.

Еще раз подчеркнем, что функция — это самостоятельная часть программы, имеющая собственные переменные, которым отводится *отдельное* место в памяти ЭВМ. Этим объясняется тот факт, что переменные с одинаковыми именами, используемые в функции и в основной программе, являются разными (в рассмотренном примере — переменная n основной программы и параметр n функции). При выполнении программы машина "не путает" имена этих переменных, так как области их действия не совпадают.

Это особенно важно при написании больших программ.

Пример

Написать функцию, подсчитывающую количество цифр натурального числа. Используя ее, определить, в каком из двух данных чисел больше цифр.

Решение

Мы уже решали похожую задачу, только в ней не требовалось написать функцию.

Надо выделять последнюю цифру числа до тех пор, пока число не станет равным нулю. При этом каждый раз счетчик увеличивается на 1 (начальное значение счетчика — 0).

```
Function Quantity(x: Longint): Byte;
Var k: Byte;
Begin
  k:=0;
  While x <> 0 Do
  Begin
    Inc(k);
    x:=x div 10;
  End;
  Quantity:=k;
End;
```

В заголовке функции указано ее имя — Quantity. Функции передается только один параметр — целое число, количество цифр которого надо найти. Результат — тоже целое число. В разделе переменных описана переменная k — счетчик цифр. В теле функции с помощью цикла While и выполняются указанные выше действия (увеличивается значение счетчика и удаляется последняя цифра).

Еще раз заметим, что память для переменной k, которая является локальной, выделяется только тогда, когда начинает свою работу функция. После завершения работы функции эта часть памяти освобождается и значение k будет не определено.

```
Program Example_31;
Var n1, n2: Longint;
    k1, k2: Byte;
Function Quantity(x: Longint): Byte;
Var k: Byte;
Begin
  k:=0;
  While x<>0 Do
  Begin
    Inc(k);
    x:=x div 10;
  End;
  Quantity:=k;
End;
Begin
  Writeln('Введите  два числа ');
  {ввод чисел}
  Readln(n1, n2);
  k1:=Quantity(n1);{количество цифр
                    первого числа}
  k2:=Quantity(n2);{количество цифр
                    второго числа}
  If k1=k2 Then Writeln('Одинаковое
                         количество цифр')
  Else If k1>k2 Then Writeln('В первом
                      числе цифр больше')
  Else Writeln('Во втором числе цифр
                больше');
  Readln;
End.
```

Решение задач

1. Найти сумму цифр числа.

2. Найти первую цифру числа.

3. Найти количество делителей числа.

4. Найти числа из промежутка от *A* до *B*, у которых больше всего делителей.

5. Найти сумму всех делителей числа.

6. Определить, является ли число совершенным (использовать функцию из предыдущего задания, изменив ее, если это необходимо).

7. Определить, является ли число простым.

8. Среди чисел из интервала от *A* до *B* найти все простые.

9. Составьте программу, проверяющую, является ли данное число палиндромом.

10. Определить, является ли данное число автоморфным. *Автоморфным* называется число, квадрат которого заканчивается им самим. Например, автоморфным является число 6, так как его квадрат 36 заканчивается на 6, или число 25 — его квадрат 625.

11. Используя функцию из предыдущей задачи, найти все автоморфные числа из промежутка от *A* до *B*.

Примечание. Задачи 8 и 11 можно модифицировать, изменив условие следующим образом: найти *N* первых таких чисел или найти первые *N* таких чисел начиная с числа *M*.

12. Составить программу нахождения наибольшего общего делителя нескольких чисел, используя функцию нахождения НОД двух чисел (см. 1.2.3. "Алгоритм Евклида").

13. Составить программу, вычисляющую наименьшее общее кратное четырех чисел (использовать функцию из предыдущего задания).

14. Даны четыре числа. Вывести на экран наибольшую из первых цифр заданных чисел. Например, если $a = 25$, $b = 730$, $c = 127$, $d = 1995$, то надо напечатать цифру 7.

15. Дано натуральное число *n*. Выяснить, имеются ли среди чисел *n*, *n*+1, ..., 2*n* *близнецы*, т.е. простые числа, разность между которыми равна двум.

2.1.3. Примеры рекурсивного программирования

Алгоритм, который в процессе работы обращается сам к себе, называется *рекурсивным*. Для иллюстрации понятия рекурсия часто приводят пример с телевизором, на экране которого изображен этот же телевизор, на экране второго — опять телевизор и так далее.

Можно разбить все рекурсивные задачи на 4 вида.

Задачи с рекурсивной формулировкой

Некоторые объекты являются рекурсивными по определению, поэтому рекурсивные алгоритмы их получения буквально повторяют соответствующие определения.

Пример

Вычисление факториала натурального числа.

Для того чтобы вычислить $N!$, надо значение $(N-1)!$ умножить на N, при этом $1! = 1$. В общем виде это можно записать так:

$$N! = \begin{cases} 1 & \text{при } N = 1 \\ N(N-1)! & \text{при } N > 1 \end{cases}$$

Для вычисления факториала опишем функцию.

```
Function factorial(n: Integer): Longint;
Begin
  If n=1 Then factorial:=1
  Else factorial:=n*factorial(n-1);
End;
```

Рассмотрим последовательность вызовов этой функции для `n=5`. Первый вызов функции происходит в основной программе `a:=factorial(5)`. Отметим, что при каждом обращении к функции будет создаваться свой набор локальных переменных (в данном случае в функции `factorial` имеется всего одна локальная переменная `n`). Для каждой локальной переменной на время работы функции выделяется память. После завершения работы функции эта память освобождается и переменные удаляются.

Так как `n≠1`, то управление передается на ветку `Else` и функции `factorial` присваивается значение `n*factorial(n-1)`, то есть `5*factorial(4)`. Происходит второй вызов функции `factorial`, с параметром `4`. Этот процесс повторяется до тех пор, пока значение параметра не станет равным `1`. Тогда `n=1`, а поэтому значение функции `factorial=1`. Таким образом, `n=1` — это условие окончания рекурсии. Управление передается в точку вызова, то есть в предыдущую функцию для `n=2.factorial:=n*factorial(n-1)`, значит, `factorial:=2*1`, следовательно, `factorial(2)=2`. Возвращаемся назад, поднимаясь "вверх" по цепочке рекурсивных вызовов. Таким образом, получаем значение `factorial(5)=120`, это значение и присваивается переменной `a` (см. *рис.* 5).

```
                                    120
1 вызов(n=5)

Function factorial(n: Integer): Longint;
Begin
  If n=1 Then factorial:=1
  Else factorial:=n*factorial(n-1);
End;


2 вызов(n=4)

Function factorial(n: Integer): Longint;
Begin
  If n=1 Then factorial:=1
  Else factorial:=n*factorial(n-1);
End;


3 вызов(n=3)

Function factorial(n: Integer): Longint;
Begin
  If n=1 Then factorial:=1
  Else factorial:=n*factorial(n-1);
End;

4 вызов(n=2)

Function factorial(n: Integer): Longint;
Begin
  If n=1 Then factorial:=1
  Else factorial:=n*factorial(n-1);
End;

5 вызов(n=1)

Function factorial(n: Integer): Longint;
Begin
  If n=1 Then factorial:=1
  Else factorial:=n*factorial(n-1);
End;
```

Рис. 5

Задания

1. Написать рекурсивную процедуру для ввода с клавиатуры последовательности чисел (окончание ввода — 0) и вывода ее на экран в обратном порядке.

2. Написать рекурсивную функцию вычисления значений функции Аккермана для неотрицательных чисел n и m, вводимых с клавиатуры.

$$A(n,m) = \begin{cases} m+1, & \text{если } n = 0 \\ A(n-1,1), & \text{если } n \neq 0, \quad m = 0 \\ A(n-1, A(n, m-1)), & \text{если } n > 0, \quad m \geq 0 \end{cases}$$

3. Найти сумму первых N членов арифметической (геометрической) прогрессии.

4. Найти первые N чисел Фибоначчи. Каждое число Фибоначчи, кроме первых двух, равно сумме двух предыдущих чисел, а первые два равны 1 (1, 1, 2, 3, 5, 8, 13, 21, ...).

$$\Phi(n) = \begin{cases} 1, & \text{если } n = 1 \text{ или } n = 2 \\ \Phi(n-1) + \Phi(n-2), & \text{если } n > 2 \end{cases}$$

Задачи, из постановки которых можно извлечь рекурсию

В формулировках некоторых задач рекурсия не присутствует в явном виде, но их можно свести к рекурсивным.

Пример

Сложение двух чисел.

Пусть надо сложить два целых числа a и b, а можно только прибавлять или вычитать 1. Тогда:

если $b = 0$, то $a + b = a$;
если $b > 0$, то $a + b = (a + 1) + (b - 1)$;
если $b < 0$, то $a + b = (a - 1) + (b + 1)$.

Решение

Можно дать следующее рекурсивное определение операции сложения двух чисел:

$$a+b = \begin{cases} a, & \text{если } b = 0 \\ (a+1)+(b-1), & \text{если } b > 0 \\ (a-1)+(b+1), & \text{если } b < 0 \end{cases}$$

Опишем соответствующую функцию:

```
Function Sum(a, b: Integer): Integer;
Begin
  If b=0 Then Sum:=a
  Else If b>0 Then Sum:=Sum(a+1, b-1)
       Else Sum:=Sum(a-1, b+1 );
End;
```

Пример

Найти НОД двух натуральных чисел.

Решение

Ранее мы уже рассматривали один из способов реализации алгоритма Евклида (см. 1.2.3. "Алгоритм Евклида"). Рассмотрим еще один способ.

Имеются два натуральных числа a и b. Если $a=b$, то НОД$(a, b)=a$. Если $a>b$, то НОД$(a, b)=$НОД$(a-b, b)$. Если $a<b$, то НОД$(a, b)=$НОД$(a, b-a)$.

Рассмотрим конкретный пример: найдем наибольший общий делитель чисел 123 и 36 (см. таблицу).

```
Function NOD(a,b: Integer): Integer;
Begin
  If a=b Then NOD:=a
  Else If a>b Then NOD:=NOD(a - b, b)
       Else NOD:=NOD(a, b - a);
End;
```

a	b	Примечание
123	36	Так как $a>b$, $a:=a-b$
87	36	$a:=a-b$
51	36	$a:=a-b$
15	36	Так как $b>a$, $b:=b-a$
15	21	$b:=b-a$
15	6	$a:=a-b$
9	6	$a:=a-b$
3	6	$b:=b-a$
3	3	Так как $a=b$, НОД$:=a$

Задания

1. Нарисовать схему вызовов функций.

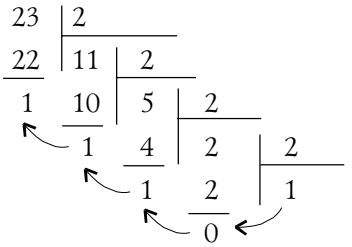
2. Написать рекурсивную функцию нахождения НОД двух натуральных чисел, используя вариант алгоритма Евклида, рассмотренный в разделе 1.2.3.

Пример

Перевести натуральное число из десятичной системы счисления в двоичную.

Решение

Переведем число 23 в двоичную систему счисления. Для этого разделим его на 2, получим целую часть и остаток от деления. Целую часть снова делим на 2 и получаем целую часть и остаток. Так делаем до тех пор, пока целая часть не станет меньше делителя (то есть пока она не станет равна 1).



Теперь начиная с этой единицы выписываем в обратном порядке все остатки от деления. Это и будет запись числа 23 в двоичной системе счисления:

$23_{10}=10111_2$.

Опишем соответствующую процедуру:

```
Procedure Rec(n: Integer);
Begin
  If n>1 Then Rec(n div 2);
  Write(n mod 2);
End;
```

Прорисуем вызовы процедуры для числа 23. Первый вызов процедуры производится в основной программе (*рис.* 6).

Результат: 10111.

Первая цифра (1) выводится на экран из последнего вызова процедуры `Rec`, следующая цифра (0) — из предпоследнего и так далее, последняя (1) — из первого. Таким образом, вывод очередной цифры происходит перед выходом из очередного экземпляра процедуры `Rec`.

Задание

Написать процедуру перевода из десятичной системы в N-ичную при условии, что $2 \le N \le 16$ (значение N вводится с клавиатуры). Каким будет условие прекращения рекурсии?

Задачи, которые можно решить как частный случай обобщенной

Если нельзя извлечь рекурсию из постановки задачи, то можно расширить задачу, обобщить ее (например, введя дополнительный параметр). Удачное обобщение может помочь увидеть рекурсию. После этого возвращаемся к частному случаю и решаем исходную задачу.

```
1 вызов(n=23)

Procedure Rec(n: Integer);
Begin
  If n>1 Then Rec(n div 2);
  Write(n mod 2);
End;

2 вызов(n=11)

Procedure Rec(n: Integer);
Begin
  If n>1 Then Rec(n div 2);
  Write(n mod 2);
End;

3 вызов(n=5)

Procedure Rec(n: Integer);
Begin
  If n>1 Then Rec(n div 2);
  Write(n mod 2);
End;

4 вызов(n=2)

Procedure Rec(n: Integer);
Begin
  If n>1 Then Rec(n div 2);
  Write(n mod 2);
End;

5 вызов(n=1)

Procedure Rec(n: Integer);
Begin
  If n>1 Then Rec(n div 2);
  Write(n mod 2);
End;
```

Рис. 6

Пример

Определить, является ли заданное натуральное число простым.

Решение

Данную задачу можно обобщить, например, так: определить, верно ли, что заданное натуральное число N не делится ни на одно число, большее или равное M ($2 \le M \le N$), но меньшее N.

Соответствующая функция должна принимать значение "истина" в двух случаях:

- Если $M = N$;
- Если N не делится на M и функция принимает значение "истина" для чисел $M+1$ и N.

```
Function Simple(M,N: Integer): Boolean;
Begin
  If M=N Then Simple:=True
  Else Simple:=(N Mod M <> 0) And
       Simple(M+1,N);
End;
```

Вернемся к исходной задаче. Она является частным случаем обобщенной, если M положить равным 2. Первое обращение к функции будет таким: `Simple(2,N)`, где `N` — это данное число.

Задание

Изучить работу функции в пошаговом режиме и нарисовать схему вызовов функций.

Задачи, в которых можно использовать характеристику или свойство функции

Пример

Для заданного натурального числа $N \ge 1$ определить натуральное число a, для которого выполняется неравенство: $2^{a-1} \le N < 2^a$.

Решение

Заметим, что значение a зависит от N следующим образом:

$$a(N) = \begin{cases} 1, & \text{если } N = 1 \\ a(N \ \ div \ \ 2) + 1, & \text{если } N > 1 \end{cases}$$

Рассмотрим пример. Пусть $N = 34$.

$2^{a-1} \le 34 < 2^a$, прибавим 1 и переходим к $34 \ div \ 2$

$2^{a-1} \le 17 < 2^a \ \ +1$

$2^{a-1} \le \ 8 < 2^a \ \ +1$

$2^{a-1} \le \ 4 < 2^a \ \ +1$

$2^{a-1} \le \ 2 < 2^a \ \ +1$

$2^{a-1} \le \ 1 < 2^a$; получим $a = 1$

А теперь возвращаемся назад, к последней единице прибавляем все предыдущие. Таким образом, получается 6.

Опишем соответствующую функцию:

```
Function A(n: Integer): Integer;
Begin
  If N=1 Then A:=1 Else A:=A(N Div 2) +1;
End;
```

Задание

1. Нарисовать схему вызовов функции `A`.

2. Функция $F(n)$ определена для целых положительных чисел следующим образом:

$$F(n) = \begin{cases} 1, & \text{если } n = 1 \\ \sum_{i=2}^{n} F(n \ \ div \ \ i), & \text{если } n \ge 2 \end{cases}$$

Вычислить значения этой функции для $n = 5, 6, 7, \ldots, 20$. Нарисовать схему вызовов (обратите внимание, что из одной функции может быть несколько вызовов этой же функции).

2.2. ФАЙЛОВЫЙ ТИП ДАННЫХ

2.2.1. Общие положения

Множества значений или переменных с одним общим именем называются структурированными (составными) типами. Имеется несколько способов построения составных типов, каждый из которых отличается способом обращения к отдельным компонентам и, следовательно, способом обозначения компонентов, вхо-

дящих в данные структурированных типов.

По способу организации и типу компонентов выделяют четыре основные разновидности структурированных типов:

- регулярный тип (массивы);
- комбинированный тип (записи);
- файловый тип (файлы);
- множественный тип (множества).

Использование структурированных типов данных позволяет решать разнообразные и достаточно сложные задачи.

2.2.2. Файловый тип данных. Операции для работы с файлами последовательного доступа

Описание

В задачах, которые мы рассматривали, данные поступали с клавиатуры, а результаты выводились на экран дисплея. Поэтому ни исходные данные, ни результаты не сохранялись. Всякий раз при выполнении одной и той же программы, особенно во время ее отладки, приходилось заново вводить исходные данные. А если их очень много? Тогда удобно оформить исходные данные и результаты в виде файлов, которые можно хранить на диске точно так же, как и программы.

Файл — это область памяти на внешнем носителе, в которой хранится некоторая информация. В языке Паскаль файл представляет собой последовательность элементов одного типа. Мы будем работать только с т.н. *файлами последовательного доступа.* В таких файлах, чтобы получить доступ к элементу, необходимо *последовательно* просмотреть все предыдущие.

Файл последовательного доступа можно сравнить с магнитной лентой, на которой записаны песни. Для того чтобы найти конкретную песню, надо перемотать кассету на начало и прослушивать песню за песней до тех пор, пока не будет найдена нужная.

Зачем нужны файлы? Объем информации, которую можно сохранить в файле, очень велик. Он значительно больше, чем объем, который можно хранить в оперативной памяти, например, при использовании массивов.

Объявление файловой переменной в разделе описания переменных имеет вид:

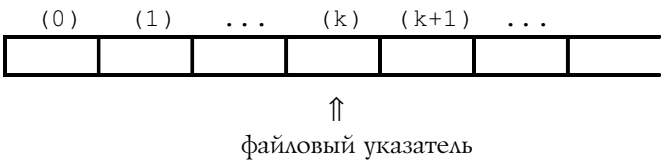
```
Var <имя файла>: File Of <тип элементов>;
```

Например,

```
Var Ft: File Of Integer;
{файл, элементами которого являются
 целые числа}
    M: File Of Char;
{файл, элементами которого являются
 символы}
Type File_Integer=File Of Integer;
     File_Char=File Of Char;
Var F1: File_Integer;
    F2: File_Char;
```

Так как в описании указывается тип элементов, такие файлы называются *типизированными.* Все элементы файла пронумерованы начиная с нуля.

С каждым файлом связан так называемый *файловый указатель.* Это неявно описанная переменная, которая указывает на некоторый элемент файла.



⇑

файловый указатель

Все операции производятся с элементом, который определен файловым указателем.

Обработка файлов

Связь переменной файлового типа с файлом на диске

Для установления связи между файловой переменной и файлом на диске имеется стандартная процедура Assign.

```
Assign(<имя файловой переменной>,
     '<имя файла на диске>');
```

Например,

```
Assign(F1,'A:INT.DAT');
```

После установления такого соответствия все операции, выполняемые над переменной F1, будут выполняться над файлом, хранящимся на диске A и имеющим имя INT.DAT.

Файл в каждый момент времени может находиться в одном из двух состояний: либо он открыт только для записи, либо только для чтения.

Чтение из файла

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

Для чтения из файла необходимо открыть файл для чтения посредством процедуры

```
Reset(<имя файловой переменной>);
```

Собственно чтение данных из файла выполняется процедурой

```
Read(<имя файловой переменной>,
     <имя переменной>);
```

Переменная должна иметь тот же тип, что и компоненты файла. Отметим, что если оператор ввода имеет вид Read(<имя переменной>), то данные вводятся с клавиатуры, а если Read(<имя файловой переменной>, <имя переменной>), то данные вводятся из файла, хранящегося на диске.

Закрытие файла

После того как данные из файла прочитаны, его необходимо закрыть посредством процедуры

```
Close(<имя файловой переменной>).
```

Общая схема чтения данных из файла, таким образом, следующая:

```
Reset(<имя файловой переменной>);
...
Read(<имя файловой переменной>,
     <имя переменной>);
...
Close(<имя файловой переменной>);
```

Признак конца файла

Так как число элементов файла не известно заранее, необходимо уметь определять, что файл кончился. Для этого используется логическая функция `EOF(<имя файловой переменной>)` *(EOF — End Of File)*. Она принимает истинное значение (`True`), если достигнут конец файла, и ложное (`False`) — в противном случае.

Пример

Прочитаем из файла целые числа и выведем их на экран:

```
Assign(F1,'A:INT.DAT');
{связываем файловую переменную
 с файлом на диске}
Reset(F1);{открываем файл для чтения}
While Not EOF(F1) Do
 {пока не достигнут конец файла F1}
Begin
  Read(F1,n);{считываем очередное число}
  Write(n,' ');{выводим его на экран}
End;
Close(F1);{закрываем файл}
```

Запись в файл

Под записью в файл понимается вывод результатов программы из оперативной памяти ЭВМ в файл на диске.

Для записи в файл необходимо открыть файл для записи посредством процедуры

```
Rewrite(<имя файловой переменной>);
```

Собственно запись данных в файл выполняется процедурой

```
Write(<имя файловой переменной>,
      <значение>);
```

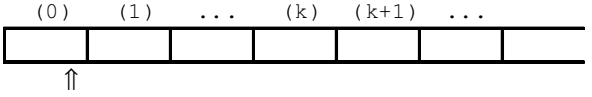
Если оператор вывода имеет вид `Write(<значение>)`, то данные выводятся на экран дисплея, а если `Write(<имя файловой переменной>,<значение>)`, то данные записываются в файл. После работы с файлом его необходимо закрыть.

Общая схема записи данных в файл, таким образом, следующая:

```
Rewrite(<имя файловой переменной>);
...
Write(<имя файловой переменной>,
      <значение>);
...
Close(<имя файловой переменной>);
```

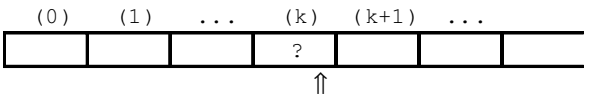
Примечания

1. После выполнения процедур открытия файла для чтения или для записи (`Reset` или `Rewrite`) текущий указатель "смотрит" на первый элемент (элемент с номером `0`). Например, после выполнения оператора `Reset(F1)` картина следующая:

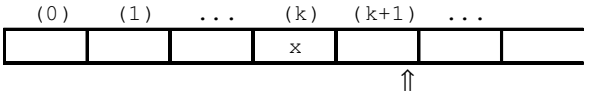


2. Если перед выполнением процедуры `Write` (например, оператора `Write(F1,x)`) указатель "смотрел" на элемент с номером `k`, то после выполнения он уже будет "смотреть" на `(k+1)`-й элемент, а `k`-й станет равен `x`.

До выполнения процедуры записи в файл

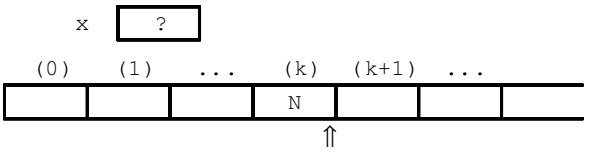


После выполнения процедуры записи в файл

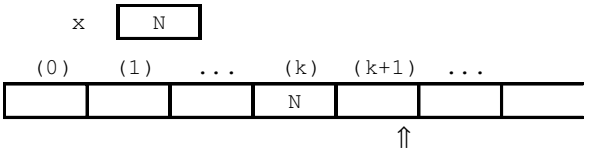


3. Если перед выполнением процедуры `Read` (например, оператора `Read(F1,x)`) указатель "смотрел" на элемент с номером `k`, то после выполнения он будет "смотреть" на следующий — `(k+1)`-й элемент, а значение переменной `x` станет равно `k`-му элементу:

До выполнения процедуры чтения из файла



После выполнения процедуры чтения из файла



Прямой доступ к элементам файла

Несмотря на то, что в стандартном Паскале имеются лишь файлы последовательного доступа, Турбо Паскаль содержит процедуры и функции для более эффективной работы с файлами. В частности, имеется возможность осуществлять прямой доступ к элементам файла.

Установка указателя

Процедура Seek(<имя файловой переменной>,N) устанавливает файловый указатель на N-й элемент. Например, Seek(F1,3). Напомним, что элементы файла нумеруются с нуля.

Определение номера элемента

Функция FilePos(<имя файловой переменной>) возвращает номер элемента, на который "смотрит" файловый указатель.

Пример

Найти номер элемента, с которым будет выполняться следующая операция.

Решение

Воспользуемся функцией FilePos: N:=FilePos(F1).

Определение количества элементов в файле

Функция FileSize(<имя файловой переменной>) возвращает количество элементов в файле.

Удаление и переименование файлов

Удаление файлов. Процедура Erase(<имя файловой переменной>) удаляет файл на внешнем носителе, с которым связана файловая переменная.

Переименование файлов. Процедура Rename (<имя файловой переменной>,'<новое имя на диске>') переименовывает файл, связанный с данной файловой переменной.

Пример

Ввести с клавиатуры и записать в файл DAN1.DAT последовательность целых чисел.

Решение

Сначала свяжем файловую переменную с конкретным внешним файлом при помощи процедуры Assign. Откроем файл для записи посредством процедуры Rewrite. Признак конца последовательности чисел — ввод числа 0.

```
Program Example_32;
Var F: File Of  Integer;
    n: Integer;
Begin
  Assign(F,'DAN1.DAT');
        {Связываем файловую переменную
         с файлом на диске}
  Rewrite(F);{Открываем файл для записи}
  Writeln('Конец ввода чисел  - 0');
  Repeat  {Пока не будет введен 0}
    Writeln('Введите число ');
    Readln(n);
     {Если введено число, отличное от 0,
     то записываем его в файл}
    If n<>0 Then Write(F, n);
  Until n=0; {Если введен 0, то
                выходим из цикла}
    Close(F);    {Закрываем файл}
End.
```

Пример

В файле DAN1.DAT записаны целые числа (см. предыдущую задачу). Вычислить сумму элементов файла и результат вместе с исходными данными записать в файл DAN2.DAT.

Вопросы для обсуждения

1. Сколько переменных нужно для решения данной задачи? Каких?
2. Как определить конец файла?
3. Как считывать элементы из файла DAN1.DAT?
4. Как записать данные в файл DAN2.DAT?

```
Program Example_33;
Var F1, F2: File Of Integer;
                    {Файловые переменные}
    S, N: Integer;
Begin
  {С файловой переменной F1 связываем
   файл на диске}
  Assign(F1,'DAN1.DAT');
  Reset(F1);    {Открываем файл F1
              для чтения}
  {С файловой переменной F2
   связываем файл на диске}
  Assign(F2,'DAN2.DAT');
  Rewrite(F2);
  {Открываем файл F2 для записи}
  S:=0;
  While Not EOf(F1) Do  {Проверка на
                         конец файла F1}
  Begin
    Read(F1,N);{Чтение элемента из
                файла F1 }
    Write(F2,N);
          {Запись элемента в файл F2 }
    S:=S+N; {Накопление суммы}
  End;
  {Запись суммы элементов в конец файла F2}
  Write(F2,S);
  Write('Результат находится в файле
         DAN2.DAT')
  Close(F1);    {Закрываем файл F1}
  Close(F2);    {Закрываем файл F2}
  Readln;
End.
```

Решение задач

1. Дан файл F, элементами которого являются целые числа. Найти:

a) количество элементов в файле;

b) наибольший из элементов. Если имеется несколько элементов с одинаковым наибольшим значением, то подсчитать количество таких элементов;

c) среднее арифметическое элементов.

2. Даны файлы F и G, элементами которых являются символы. Записать в файл H:

a) все элементы файлов F и G;

b) все латинские буквы (прописные и строчные) из файла F.

3. Дан файл A, элементами которого являются целые числа. Записать в файл B все четные числа из файла A, а в файл C — все нечетные.

4. Даны два файла — A и B, состоящие из элементов одинакового типа. Поменять местами содержимое этих файлов.

5. Дан файл F, элементами которого являются символы. Переписать содержимое файла F в файл G в обратном порядке.

6. Даны два файла — A и B. Элементами файла A являются целые числа, а файла B — символы. Вывести на экран все числа из первого файла, а рядом с ними — элементы из второго файла с соответствующими номерами. Если во втором файле нет элемента с данным номером, то вывести соответствующее сообщение.

2.2.3. Текстовые файлы

Повторение

1. Что такое файл?
2. Как описываются файлы?
3. Какие операции можно производить с файлами?
4. Как производится запись в файл?
5. Как производится чтение из файла?
6. Как закрыть файл?
7. Как получить доступ к элементу файла с данным номером?
8. Как подсчитать количество элементов в файле?
9. Какие из следующих описаний неправильные и почему?

```
Var f:  Fail Of Real;
    f1, f2: File  Of Char;

Type ff: file Of Integer;
     gg = file Of bolean;
Var  f: ff;  g: gg;

Type ff=file Of file;
Var  f1,f2: ff;
```

10. Какие из следующих операторов правильные?

```
Assign(f1,'A:STR1.DAT');
Reset(f1,f2);
Rewrite;
Assign(f2,'C:\TT\TAB1.DAT')
Rewrite(f1);
```

Текстовые файлы состоят из символьных строк. Строки могут иметь различную длину, и в конце каждой строки стоит признак конца строки. Для описания текстовых файлов используется служебное слово Text:

```
Var A: Text;
```

Обработка текстовых файлов

Для обработки текстовых файлов используются те же процедуры и функции, что и для обработки обычных типизированных файлов. Для связывания файловой переменной с файлом на диске употребляется процедура Assign. Текстовые файлы могут быть открыты для чтения процедурой Reset или для записи процедурой Rewrite.

Для чтения данных применяется процедура Read. Если необходимо после чтения данных перейти на следующую строку, то используется процедура Readln. Если необходимо просто перейти к следующей строке, то можно использовать процедуру Readln(<имя файловой переменной текстового файла>), которая устанавливает файловый указатель на первый элемент следующей строки.

Процедура Write записывает данные в текущую строку. Если надо записать данные и перейти к следующей строке, то можно использовать процедуру Writeln. Если требуется только перейти для записи на новую строку, то применяется процедура Writeln(<имя файловой переменной текстового файла>), которая записывает в файл признак конца строки и устанавливает файловый указатель на начало следующей строки.

Так как в строках может быть разное количество символов, имеется логическая функция Eoln(<имя файловой переменной текстового файла>), которая принимает значение True, если достигнут конец строки.

Кроме перечисленных процедур и функций, к текстовым файлам применяется процедура Append(<имя файловой переменной текстового файла>). Она открывает файл для "дозаписи", помещая файловый указатель в конец файла.

Пример

Дан текстовый файл, содержащий только целые числа, в каждой строке может быть несколько чисел, которые разделяются пробелами. Вывести на экран все числа с учетом разбиения на строки и подсчитать количество элементов в каждой строке.

Решение

Пусть в файле содержится следующая информация:

```
-32  16  0  8  7
4  5  9  13  11  -5  -8
6  -8  0  -12
5  4  3  2  1  12
1  2
-1  -2  -4
-1  -2  4
```

Этот файл можно создать в среде Турбо Паскаль следующим образом:

- создайте новый файл посредством команды **New** меню **File**;

• запишите все числа, разделяя их пробелами, и разбейте на строки, как указано в задании;

• сохраните файл, например, под именем INT1.DAT.

Этот файл используется в программе, которая приведена ниже.

```
Program Example_34;
Var  F: Text;
     x, k: Integer;
Begin
  Assign(F, 'INT1.DAT');
  {Связываем файловую переменную
   с файлом на диске}
  Reset(F);     {Открываем файл для чтения}
  While Not Eof(F) Do {Пока не
                достигнут конец файла}
  Begin
   k:=0; {Счетчик элементов строки}
   While Not Eoln(F) Do   {Пока не
               достигнут конец строки}
    Begin
     Read(F, x);
    {Считываем очередное число}
     Write(x,'  ');
     {Выводим его на экран}
     Inc(k); {Увеличиваем счетчик }
    End;
    Writeln(' В строке', k,' элементов');
   Readln(F);
  {Переходим к следующей строке файла}
End;
Close(F); {Закрываем файл}
Readln;
End.
```

Пример

Дан текстовый файл, содержащий программу на языке Паскаль. Проверить эту программу на соответствие числа открывающих и закрывающих круглых скобок. Считать, что каждый оператор программы занимает не более одной строки файла.

Решение

Так как по условию задачи каждый оператор занимает не более одной строки, то будем подсчитывать количество открывающих и закрывающих скобок в каждой строке.

```
Program Example_35;
Var F: Text;
    kl, k2, n: Integer;
    Ch: Char;
    Logic,Pp: Boolean;
Begin
  {С файловой переменной F связываем файл
   на диске}
  Assign(F, 'TEST.PAS' );
  Reset(F);{Открываем файл для чтения}
  n:=0; {Счетчик количества строк}
  Logic:=True; {Пока ошибок не было,
   переменная Logic имеет значение True}
  While Not EOf(F) Do {Пока не достигнут
                       конец файла}
  Begin
   Inc(n); {Увеличиваем счетчик
            количества строк}
   k1:=0; {Счетчик количества
           открывающих скобок}
   k2:=0; {Счетчик количества
           закрывающих скобок}
  Pp:=False; {Переменная Pp предназна-
    чена для определения ошибки расста-
    новки скобок в строке. Она принимает
    значение True, когда обнаруживается
    ошибка}
  While Not Eoln(F) Do    {Пока не
  достигнут конец текущей строки файла}
  Begin
   Read(F, Ch);
  {Читаем очередной символ строки}
  {Если встретили открывающую скобку, то
  увеличиваем соответствующий счетчик}
  If Ch = '('  Then Inc(k1);
  {Если закрывающая скобка стоит после
  открывающей (k1>k2), то просто уве-
  личиваем счетчик. Иначе - устанавли-
  ваем Pp в True}
  If (Ch = ')')  Then
    If (k1>k2) Then Inc(k2)
    Else Pp:=True;
  End;
{Если не все закрывающие скобки расставлены
(k1<>k2) или одна из закрывающих скобок стоит
раньше открывающей (Pp=True), то была ошибка
расстановки}
If (kl<>k2) or Pp Then
  Begin
    Writeln('Ошибка в ',N,' строке');
    Logic:=False;
  End;
  Readln(F);{Переходим на следующую
              строку файла}
End;
{Если значение переменной Logic осталось ис-
тинным, то ошибок расстановки не было}
If Logic Then Writeln('Скобки расставлены
                       правильно');
Close(F); {Закрываем файл}
Readln;
End.
```

Решение задач

1. Дан текстовый файл, содержащий целые числа. Найти максимальный элемент в каждой строке.

2. Дан текстовый файл, содержащий символьные строки. Найти:

a) количество строк;

b) количество строк, начинающихся и заканчивающихся одинаковыми символами;

c) самые короткие строки;

d) симметричные строки.

3. Дан текстовый файл. Вставить в начало каждой строки ее номер и записать преобразованные строки в новый файл.

4. Даны два текстовых файла. Записать в третий файл только те строки, которые есть и в первом, и во втором файлах.

5. Дан текстовый файл. Дописать в его конце следующие данные: количество строк и количество символов в каждой строке.

2.3. РЕГУЛЯРНЫЕ ТИПЫ ДАННЫХ

2.3.1. Одномерные массивы. Работа с элементами

Повторение

1. Что такое константа?
2. Как описываются константы?
3. Как описать новый тип данных в разделе описания типов?
4. Что такое процедуры и функции? Чем они различаются? А чем похожи?
5. Как вызываются процедуры и функции? Как им передаются параметры?

Одномерные массивы. Описание одномерных массивов (разбор на примерах)

Пример

Найти сумму пяти целых чисел.

Решение

Для решения этой задачи можно описать пять целых переменных для данных чисел и еще одну — для их суммы. Обозначим исходные числа a1, a2, a3, a4 и a5, а их сумму — s. Тогда можно составить такую программу, используя функцию нахождения суммы пяти чисел:

```
Program Example_36;
Var a1,a2,a3,a4,a5,s: Integer;
Function Sum(x1,x2,x3,x4,x5: Integer): Integer;
Begin
  Sum:=x1+x2+x3+x4+x5;
End;
Begin
  Writeln('Введите пять целых чисел');
  Readln(a1,a2,a3,a4,a5);
  {Вводим пять целых чисел}
  s:=Sum(a1,a2,a3,a4,a5);
  {Находим их сумму}
  Writeln('их сумма равна ',s);
  {Выводим результат на экран}
  Readln;
End.
```

Пример

Найти сумму тридцати целых чисел.

Решение

Если решать эту задачу по аналогии с предыдущей, то необходимо будет описать 30 переменных для всех исходных чисел. Это не очень удобно. Поэтому используем для решения этой задачи одномерный массив.

Одномерный массив — это фиксированное количество элементов одного типа, объединенных одним именем, причем каждый элемент имеет свой уникальный номер и номера элементов идут подряд. Например, введем 30 целых чисел от 25 до 54 и объединим их общим именем A.

№	A
1	25
2	26
3	27
...	...
29	53
30	54

Имя A — общее для всех элементов.
Элементы массива — целые числа, их 30.

Опишем в разделе описания тип — одномерный массив, состоящий из 30 целых чисел.

```
Type myarray = Array[1..30] Of Integer;
```

Напомним, что раздел типов начинается со служебного слова **Type**, после него идут имя нового типа и его описание. Между именем типа и его описанием ставится знак "равно" (в разделе описания переменных между именем переменной и ее описанием ставится двоеточие). Тогда:

myarray — это имя нового типа;

Array — служебное слово (в переводе с английского означает "массив", "набор");

[1..30] — в квадратных скобках указывается номер первого элемента, затем, после двух точек, номер последнего элемента массива; в этом примере первый элемент имеет номер 1, а последний — номер 30;

Of — служебное слово (в переводе с английского означает "из");

Integer — тип всех элементов массива.

Таким образом, одномерный массив описывается следующим образом:

```
Array[n1..n2] Of <тип элементов>;
```

где n1 — номер первого элемента, n2 — номер последнего элемента, в качестве типа элементов может использоваться любой тип данных, кроме файлового.

Для того чтобы обратиться к элементу этого массива, необходимо указать имя массива и в квадратных скобках — номер элемента. Например, первый элемент массива A — A[1], а пятый — A[5].

Пример

Составить программу нахождения суммы элементов массива.

Решение

Опишем две процедуры (формирования и вывода массива) и функцию нахождения суммы элементов. Заметим, что *заполнение и вывод массива можно осуществить только поэлементно*, то есть можно сначала присвоить значение первому элементу, затем второму и так далее. Аналогично обстоит дело и с выводом массива на экран — мы станем выводить первый элемент, второй, третий и так до последнего. Значения элементов массива будем вводить с клавиатуры.

```
Program Example_37;
Const n=30;{Количество элементов массива}
Type myarray=Array[1..n]  Of  Integer;
Var  A: myarray;
     s: Integer;
{Значение этой переменной s будет равно
 сумме всех элементов массива}
```

```
Procedure Init1(Var  m: myarray);
Var i: Integer;
{Переменная  для работы с элементами массива}
Begin
  Writeln('Введите ',n, ' чисел');
  For i:=1 To n Do
  {Ввод массива с клавиатуры }
  Readln(m[i]);
  {Чтение i-го элемента}
End;
Procedure Print(m: myarray);
Var i: Integer;
Begin
  For i:=1 To n Do
     {Вывод массива }
     Write(m[i]: 3);
     {Вывод i-го элемента}
  Writeln;
End;
Function Sum(m: myarray): Integer;
Var i, sum: Integer;
Begin
  sum:=0;
  {Начальное значение суммы}
  For i:=1 To n Do sum:=sum+m[i];
  {К уже найденной сумме прибавляем i-й элемент}
End;
Begin
  Init1(A);
  {Обращение к процедуре формирования массива}
  Print(A);
  {Вывод массива}
  s:=Sum(A);
  {Нахождение суммы элементов}
  Writeln('их сумма равна ',s);
  {Вывод результата на экран}
  Readln;
End.
```

При решении задач часто приходится заполнять массивы (присваивать значения элементам). Рассмотрим несколько способов заполнения массивов.

Первый способ заполнения одномерного массива — это заполнение с клавиатуры (этот способ был рассмотрен выше — в процедуре Init1).

Второй способ — это заполнение с помощью генератора случайных чисел. Этот способ более удобен, когда в массиве много элементов, а их точные значения не слишком важны.

Составим программу заполнения одномерного массива с помощью генератора случайных чисел. Процедура вывода уже составлена ранее.

```
Program Example_38;
Const n=30; dd=51;
{n - количество элементов массива,
dd используется в генераторе случайных чисел}
Type myarray = Array[1..n]  Of  Integer;
Var  A: myarray;
Procedure Init2(Var m:myarray);
{Процедура заполнения (инициализации)
 массива случайными числами}
Var i:Integer;
Begin
   For i:=1 To n Do m[i]:=-25+Random(dd);
      {Функция Random выбирает случайное
      число из отрезка от 0 до dd-1. Очередному элементу массива будет присвоена сумма выбранного случайного числа и -25, таким образом, массив будет заполняться случайными числами от -25 до -25+(dd-1), то есть до -26+dd}
End;
Procedure Print(m:myarray);
{процедура вывода (распечатки) массива}
...
Begin
  Randomize;
  {Инициализация генератора случайных чисел}
  Init2(A);
  {Обращение к процедуре заполнения массива}
  Print(A);
  {Обращение к процедуре вывода массива}
  Readln;
End.
```

Третий способ заполнения массива — чтение значений элементов из файла. Можно заранее создать типизированный файл одномерных массивов (например, по тридцать элементов), а затем считывать из него сразу целый массив. Но мы воспользуемся текстовым файлом, так как его создавать намного удобнее. Пусть в файле записано несколько строк, а в каждой из них по 30 целых чисел. Тогда вся программа может быть такой:

```
Program Example_39;
Const n=30;
{Количество элементов массива}
Type myarray = Array[1..n] Of Integer;
Var  A: myarray;
F: text;
Procedure Init3( Var  m:myarray);
{Процедура заполнения (инициализации) массива}
Var i:Integer;
Begin
  For  i:=1 To n Do Read(f, m[i])
  {Чтение из файла очередного числа}
End;
Procedure Print(m:myarray);
{Процедура вывода (распечатки) массива}
...
Begin
{Связываем файловую переменную
 с файлом на диске}
Assign(F, '...');
Reset(F);
{Открываем файл для чтения}
While Not EOf(F) Do
Begin
  {Считываем очередную строку }
  Init3(A);
  {Обращение к процедуре заполнения массива}
  Print(A);
  {Обращение к процедуре вывода }
  Readln(F);
End;
Readln;
End.
```

Работа с элементами массива (разбор на примерах)

При работе с элементами массива можно выделить несколько видов задач.

Нахождение суммы (или произведения) элементов

Такая задача была рассмотрена выше. Часто встречаются различные модификации этой задачи, например, требуется найти сумму элементов с заданным свойством.

Пример

Найти сумму элементов, кратных заданному числу.

Решение

Изменим функцию Sum из программы Example_37. Будем суммировать не все элементы, а только те, которые удовлетворяют данному условию, то есть только те, которые делятся нацело на заданное число (остаток от деления на данное число равен 0).

```
Function Sum(m: myarray): Integer;
Var i, s, k: Integer;
Begin
  Writeln('Введите число ');
  Readln(k);
  s:=0;
  {Начальное значение суммы}
  For  i:=1 To n Do
    {Нахождение суммы}
  If  m[i] Mod k =0 Then s:=s+m[i];
  {Если элемент кратен k, то прибавляем
   его к сумме}
  sum:=s;
End;
```

Остальную часть программы Example_37 можно оставить без изменений.

Нахождение номеров элементов, обладающих заданным свойством

Пример

Найти номера четных элементов.

Решение

Необходимо просмотреть весь массив, и если просматриваемый элемент является четным, то вывести его номер. Опишем процедуру, которой передается данный массив и выводятся нужные номера.

```
Procedure Solve(m: myarray);
Var i: Integer;
Begin
  For i:=1 To n Do
    If  m[i] Mod 2=0
    Then Write(i:5);
End;
```

Нахождение количества элементов, обладающих заданным свойством

Пример

Найти количество положительных и отрицательных элементов в данном массиве.

Решение

Опишем процедуру, которая имеет три параметра — массив и два счетчика, первый — для положительных элементов, второй — для отрицательных, элементы, равные нулю, учитывать не будем.

```
Procedure Quantity(m: myarray;
                   Var k1, k2: Integer);
Var i: Integer;
Begin
  k1:=0; k2:=0;
  For i:=1 To n Do
  If m[i]>0 Then Inc(k1)
  Else If m[i]<0 Then Inc(k2);
End;
```

Есть ли в данном массиве элементы с данным свойством, или Найти первый (последний) элемент, отвечающий заданным условиям

Для решения задач этого типа удобнее использовать циклы с условиями и составлять функции, результат которых имеет логический тип.

Пример

Есть ли отрицательный элемент в массиве?

Решение

Начинаем просматривать массив с первого элемента (i=1). Пока не просмотрен последний (i<=n) и не найден отрицательный (m[i]>=0), будем переходить к следующему (inc(i)). Таким образом, мы закончим просмотр в одном из двух случаев: первый — просмотрели все элементы и не нашли отрицательного, тогда i>n, второй — нашли нужный, при этом i<=n. Опишем функцию, значение которой истина (True), если в массиве есть отрицательный элемент, и ложь (False), если его нет.

```
Function Control1(m: myarray): Boolean;
Var i: Integer;
Begin
  i:=1;
  While (i<=n) And (m[i]>=0) Do Inc(i);
  Control1:=(i<=n)
End;
```

Пример

Найти номер последнего отрицательного элемента массива.

Решение

Последний отрицательный — это первый отрицательный элемент, который встретится при просмотре массива с конца. Если очередной элемент не является отрицательным, то нужно уменьшать значение текущего индекса, пока он не станет меньше номера первого элемента или не будет найден отрицательный элемент. Таким образом, можно модифицировать предыдущую функцию. Но поскольку надо найти номер элемента, тип результата будет целым.

Договоримся, что если в массиве нет отрицательного элемента, то значение функции будет равно 0.

```
Function Control2(m: myarray): Integer;
Var i: Integer;
Begin
  i:=n;
  While (i>=1) And (m[i]>=0) Do Dec(i);
  Control2:=i;
End;
```

Решение задач

1. Найти сумму положительных элементов массива.

2. Найти сумму всех четных элементов массива (или сумму элементов, кратных заданному числу).

3. Найти сумму всех четных элементов массива, стоящих на четных местах, то есть имеющих четные номера.

4. Найти сумму первых пяти элементов массива.

5. Найти сумму элементов с k1-го по k2-й, где k1 и k2 вводятся с клавиатуры. Проверить корректность их ввода.

6. Найти сумму элементов, больших данного числа A (A вводится с клавиатуры).

7. Найти сумму элементов, принадлежащих промежутку от A до B (A и B вводятся с клавиатуры).

Примечание. В задачах 1—7 можно находить не только сумму, но и произведение.

8. Найти максимальный элемент массива и его номер при условии, что все элементы различны.

9. Найти номера всех отрицательных элементов (вывести их на экран), если таких нет, то сообщить об этом.

10. Найти номера всех элементов с максимальным значением.

11. Найти минимальный элемент.

12. Найти количество нечетных элементов.

13. Найти количество отрицательных элементов.

14. Сколько элементов массива превосходят по модулю заданное число A?

15. Найти все элементы, кратные 3 или 5. Сколько их?

16. Есть ли в данном массиве два соседних положительных элемента? Найти номера элементов, образующих первую (последнюю) пару.

17. Есть ли в данном массиве элементы, равные заданному числу? Если есть, то вывести номер одного из них.

18. Найти количество четных чисел среди пяти целых чисел (без использования массива).

19. Если существует треугольник со сторонами a, b и c, то напечатать "ДА", иначе напечатать "НЕТ" (значения a, b и c вводятся с клавиатуры).

2.3.2. Методы работы с элементами одномерного массива

Повторение

1. Что такое массив?

2. Как описывается одномерный массив?

3. Как обратиться к некоторому элементу массива? Что общего между всеми элементами массива? Что у каждого свое?

4. Как можно задать одномерный массив (перечислите некоторые способы)?

5. Как описать процедуру?

6. Как заполнить массив?

7. Как распечатать (вывести на экран) заполненный массив?

Рассмотрим еще несколько типов задач.

Изменение значений некоторых элементов

Пример

Заменить отрицательные элементы массива на их абсолютные величины.

Решение

Для решения задачи опишем процедуру. Ей будем передавать один параметр — массив, который и будет результатом, при этом значения некоторых элементов могут быть изменены.

```
Procedure Substitution1(Var m: myarray);
Var i: Integer;
Begin
  For i:=1 To n Do If m[i]<0 Then  m[i]:=-m[i];
End;
```

Пример

Прибавить к каждому элементу массива число 25.

Решение

Преобразуем предыдущую процедуру.

```
Procedure Substitution2(Var m: myarray);
Var i: Integer;
Begin
  For i:=1 To n Do  m[i]:=m[i] + 25;
End;
```

Пример

Если очередной элемент массива четный, то прибавить к нему первый, если нечетный — прибавить последний. Первый и последний элементы не изменять.

Решение

Просмотрим все элементы массива, кроме первого и последнего, и если очередной элемент четный, то есть делится на 2 без остатка, то увеличим его на значение первого элемента, иначе — увеличим его на значение последнего элемента.

```
Procedure Substitution3 (Var m: myarray);
Var i: Integer;
Begin
  For i:=2 To n-1 Do
  If m[i] Mod 2=0 Then m[i]:=m[i] + m[1]
  Else m[i]:=m[i]+m[n];
End;
```

Создание массива

Рассмотрим два примера.

Пример

Даны два одномерных массива одинаковой размерности. Получить третий массив такой же размерности, каждый элемент которого равен сумме соответствующих элементов данных массивов.

Решение

Пусть даны два массива — A и B, состоящие из пяти элементов. Получим из них массив C, тоже состоящий из пяти элементов, причем первый элемент этого массива равен сумме первых элементов массивов A и B, второй — сумме вторых элементов данных массивов — и так далее.

i	1	2	3	4	5
A	14	2	7	8	9
B	3	6	5	12	4
C	17	8	12	20	13

Для решения этой задачи опишем процедуру, которой передаются три параметра. Первые два — это два исходных одномерных массива, третий — это одномерный массив той же размерности, который является результатом и заполняется по указанному правилу.

```
Procedure Sum_Ar(a, b: myarray; Var c: myarray);
Var i: Integer;
Begin
  For i:=1 To n Do c[i]:=a[i]+b[i];
End;
```

Пример

Даны первый член арифметической прогрессии и ее разность. Записать в массив первые n членов прогрессии.

Решение

Пусть a1 — первый член прогрессии, а k — ее разность, тогда i-й член можно найти по следующему правилу: a[i]:=a[i-1]+k, или a[i]:=a1+k*(i-1); если i=1, то a[i]:=a1. Опишем процедуру, в которую передаются два параметра, а результатом является одномерный массив. Его первый элемент равен первому члену прогрессии, второй — второму — и так далее.

```
Procedure Progress(a1, k: Integer; Var a: myarray);
Var i: Integer;
Begin
  a[1]:=a1;
  For i:=2 To n Do a[i]:=a[i-1]+k
End;
```

Работа с несколькими массивами

Пример

Даны два одномерных массива — A и B. Найти их скалярное произведение.

Решение

Скалярным произведением двух массивов одинаковой размерности называется сумма произведений соответствующих элементов:

a[1]*b[1] + a[2]*b[2] + ... + a[n-1]*b[n-1] + + a[n]*b[n], где n — количество элементов в массивах.

```
Function Sp(a, b: myarray): longitn;
Var i: Integer;
    s: Longint;
Begin
  s:=0;
  For i:=1 To n Do s:=s+a[i]*b[i];
  Sp:=s;
End;
```

Решение задач

1. Изменить знак у максимального по модулю элемента массива.

2. Заменить все четные элементы на их квадраты, а нечетные удвоить.

3. Вычесть из положительных элементов элемент с номером k1, а к отрицательным прибавить элемент с номером k2, нулевые элементы оставить без изменения.

4. К четным элементам прибавить A, а из элементов с четными номерами вычесть B.

5. Отрицательные элементы возвести в квадрат.

6. Даны два целочисленных массива, состоящих из одинакового числа элементов. Получить третий массив той же размерности, каждый элемент которого равен большему из соответствующих элементов данного массива.

Например, даны два массива — A и B, состоящие из пяти элементов. Получим массив C, тоже состоящий из пяти элементов. Первый элемент этого массива равен большему из первых элементов массивов A и B, второй — большему из вторых элементов данных массивов — и так далее.

Таким образом, получим массив C, представленный в таблице.

i	1	2	3	4	5
A	14	2	7	8	9
B	3	6	5	12	4
C	14	6	7	12	9

7. Дан одномерный массив $A(a_1, a_2, \ldots, a_n)$. Найти массив $B(b_1, b_2, \ldots, b_n)$, в котором

a) $b_i=a_i^2+2a_i-1$ для всех $i=1, \ldots, n$;

b) $b_i = \begin{cases} 1\,(\text{True}) & \text{если } A_i \text{ делится на } k, \\ 0\,(\text{False}) & \text{если } A_i \text{ не делится на } k; \end{cases}$

c) $b_i=a_1+a_2+\ldots+a_i$ для всех $i=1, \ldots, n$.

8. Даны первый член геометрической прогрессии и ее знаменатель. Найти и записать в массив первые n членов этой прогрессии.

9. Получить и записать в массив первые n чисел Фибоначчи — первые два числа равны 1, а каждое следующее равно сумме двух предыдущих.

10. Даны два массива. Найти среднее арифметическое элементов каждого и сравнить эти значения.

2.3.3. Удаление элементов из одномерного массива

Повторение

1. Какие из приведенных описаний одномерных массивов являются правильными:

- `Var a: Array[1..20] Of Integer;`
- `Type myarray: Array[1..20];`
  `Var b: myarray;`
- `Var sd: Array[1..n] Of Integer;`
- `Var dd: Array[1] Of Integer;`

2. Сколько элементов в каждом из следующих массивов:

- `Var mb: Array[2..20] Of Integer;`
- `Type myarray1=Array[0..8] Of Integer;`
  `myarray2=Array[-5..5] Of Integer;`
  `Var aa: myarray1;`
  `bb: myarray2;`

3. Используя описание массивов из предыдущего пункта, определить, какие из приведенных обращений к элементам массивов неправильные и почему.

- `MB[3];`
- `myarray1[2];`
- `aa[6];`
- `BB[6];`
- `Aa[0];`
- `bb[0];`
- `bb[-3];`
- `aa[-3].`

Удаление элемента

Пример

Удалить из массива, в котором все элементы различны, максимальный элемент. После удаления максимального элемента массив "уплотнить", сдвинув все следующие за ним элементы влево. Последнему (самому правому) элементу массива присвоить 0.

Решение

Для того чтобы решить данную задачу, необходимо:

- найти номер максимального элемента k;
- сдвинуть все элементы начиная с k-го на один элемент влево;
- последнему элементу присвоить значение 0.

Рассмотрим решение задачи на конкретном примере. Пусть дан одномерный массив, состоящий из 10 элементов: 6, 3, 4, 7, 11, 2, 13, 8, 1, 5.

Номер максимального элемента равен 7 (k=7), то есть начиная с 7-го элемента будем сдвигать элементы на один влево: 7-му элементу присвоим значение 8-го, 8-му присвоим значение 9-го, а 9-му присвоим значение 10-го, на этом сдвиг заканчивается. Таким образом, сдвиг начинается с k-го элемента и заканчивается (n-1)-м (где n — количество элементов в массиве). После этого последнему элементу присвоим 0, тогда массив примет вид: 6, 3, 4, 7, 11, 2, 8, 1, 5, 0.

Примечание. При удалении элемента размерность массива не изменяется.

Составим программу для удаления максимального элемента из одномерного массива, в ней воспользуемся двумя уже знакомыми процедурами инициализации массива и вывода его на печать. Первая заполняет массив случайными числами, а вторая выводит на печать этот массив. Чтобы последний 0 не выводился на экран, мы модифицируем процедуру вывода Print и будем ей передавать не только массив, но и количество элементов, которые надо вывести, начиная с первого.

```
Program Example_40;
Const n=30; dd=51;
Type myarray = Array[1..n]  Of  Integer;
Var  A: myarray;
     k:Integer;
    {k - номер максимального элемента}
Procedure Init2(Var  m:myarray);
{Процедура заполнения (инициализации)
массива случайными числами}
...
Procedure Print1(n1: Integer; m: myarray);
{Процедура вывода (распечатки) массива}
Var i: Integer;
Begin
  For i:=1 To n1 Do Write(m[i]:5);
  Writeln;
End;
Function Maximum(m: myarray): Integer;
Var i, max, maxi: Integer;
Begin
  max:=-32768;
  {Минимальное значение типа Integer равно -32 768}
  For i:=1 To n Do
  {Просмотр всех элементов массива}
  If  m[i]>max Then
  {Если данный элемент больше максимального
   элемента, найденного среди первых
   i-1 элементов, то}
  Begin
    max:=m[i];
   {Новое значение максимального элемента}
    maxi:=i;
   {Номер максимального элемента в массиве}
  End;
  Maximum:=maxi;
End;
Procedure Delete(k1: Integer; Var m: myarray);
Var i: Integer;
Begin
  {Сдвиг элементов на один влево}
  For i:=k1 To n-1 Do
       m[i]:=m[i+1];
```

```
  {i-му элементу присваиваем значение (i+1)-го}
  m[n]:=0;
  {Последний элемент равен 0}
  End;
Begin
  Randomize;
  {Инициализация генератора случайных чисел}
  Init2(A);
  {Заполнение массива A}
  Print1(n,A);
  {Вывод заполненного массива A}
  k:=Maximum(A);
  {Поиск номера максимального элемента}
  Delete(k, A);
  {Удаление элемента с номером k}
  Print1(n-1,A);
  {Вывод нового массива A}
  Readln;
End.
```

Пример

Решить предыдущую задачу, считая, что максимальный элемент может встречаться несколько раз.

Решение

Когда необходимо удалять несколько элементов, то это лучше всего делать с конца массива, так как иначе нужно будет снова возвращаться к элементу, который только что удаляли (эта проблема возникает в том случае, когда подряд идут два максимальных элемента: если первый удалить, то на его место снова встанет максимальный элемент). Просматривать массив с конца можно при помощи цикла с параметром, который имеет следующий вид:

```
For i:=B Downto A Do <тело цикла>
```

Значение переменной i будет уменьшаться на единицу начиная от B до A.

Кроме того, номер максимального элемента запоминать не будем, а просмотрим массив с конца, и если элемент имеет максимальное значение, то удалим его, при этом значение счетчика k будем увеличивать на 1. Для решения этой задачи надо изменить функцию Maximum, сейчас нам нужен не номер, а значение максимального элемента. В программе это будет выглядеть так:

```
Program Example_41;
Const n=30; dd=51;
Type myarray = Array[1..n] Of Integer;
Var A: myarray;
    m, k, i:Integer;
   {m - значение максимального элемента
    k -  количество удаленных элементов}
Procedure Init2(Var  m:myarray);
{Процедура заполнения (инициализации)
 массива случайными числами}
...
Procedure Print1(n1: Integer; m: myarray);
{Процедура вывода (распечатки) массива}
...
Function Maximum(m: myarray): Integer;
Var i, max : Integer;
Begin
  max:=-32768;
  For i:=1 To n Do
  {Просмотр всех элементов массива}
  If m[i]>max Then max:=m[i];
  {Новое значение максимального элемента}
  Maximum:=max;
End;
Procedure Delete(k1: Integer; Var m: myarray);
{Процедура удаления элемента с данным номером}
...
Begin
  Randomize;
  {Инициализация генератора случайных чисел}
  Init2(A);
  {Заполнение массива A}
  Print1(n,A);
  {Вывод заполненного массива A}
  {Поиск значения максимального элемента}
  m:=Maximum(A);k:=0;
  {Просмотр всех элементов начиная с последнего}
  For i:=n Downto 1 Do
  If A[i]=m Then
  {Если данный элемент имеет максимальное
   значение, то }
  Begin
    {Удаляем элемент с номером i}
    Delete(i,A);
    Inc(k);
  End;
 Print1(n-k,A);
 {Вывод нового массива A}
 Readln;
End.
```

Решение задач

1. Удалить первый отрицательный элемент массива, если такой элемент есть.

2. Удалить все отрицательные элементы массива.

3. Удалить все элементы, большие данного числа A (A вводится с клавиатуры).

4. Удалить все четные элементы, стоящие на нечетных местах.

5. Удалить все повторяющиеся элементы, оставив их первые вхождения, то есть в массиве должны остаться только различные элементы.

6. Удалить последний четный элемент массива.

7. Удалить все элементы, кратные 3 или 5.

8. Удалить все элементы массива начиная с k1-го по k2-й (k1 и k2 вводить с клавиатуры). Проверить корректность ввода значений k1 и k2 (k1≤k2); если ввод некорректный, то вывести сообщение об ошибке и закончить работу.

2.3.4. Вставка элементов в одномерный массив

Вставка элемента после элемента с заданным номером

Пример

Вставить число 100 после пятого элемента массива.

Решение

Пусть k — номер элемента, после которого мы должны вставить элемент x (k и x будем вводить с клавиатуры). Вставка осуществляется следующим образом:

- первые k элементов массива остаются без изменений;
- все элементы начиная с (k+1)-го необходимо сдвинуть вправо;
- элементу с номером (k+1) присваиваем значение x.

Рассмотрим конкретный пример. Пусть дан следующий одномерный массив из N (N=10) элементов: 3, –12, 5, 14, 27, –6, 1, –34, 10, –15.

Надо вставить элемент со значением 100 после пятого элемента массива. Мы получим следующий массив: 3, –12, 5, 14, 27, 100, –6, 1, –34, 10, –15.

Таким образом, после вставки в массиве станет 11 элементов, и это надо учесть при описании типа myarray:

```
Type myarray = Array[1..n+1] Of Integer;
```

Будем выводить массив два раза — до и после вставки нового элемента, используя уже известную процедуру Print1.

Составим теперь основную программу с использованием новой процедуры Insert1(k1, x1, m), которой передаются: k1 — номер элемента, после которого надо вставить новый элемент, x1 — значение нового элемента, m — массив. Сдвиг элементов будем начинать с конца массива.

```
Program Example_42;
Const n=10; dd=51;
Type myarray=Array[1..n+1] Of Integer;
Var A: myarray;
    x, k:Integer;
    {x - значение нового элемента
     k - номер элемента, после которого
         вставляем}
Procedure Init2(Var m: myarray);
{Процедура заполнения (инициализации) массива
 случайными числами}
...
Procedure Print1(n1: Integer; m: myarray);
{Процедура вывода (распечатки) массива}
...
Procedure Insert1(k1, x1: Integer; Var m: myarray);
Var i: Integer;
Begin
  {Сдвиг элементов на одну позицию назад}
  For i:=n Downto k1+1 Do m[i+1]:=m[i];
  m[k1+1]:=x1;
  {Вставка элемента после k1-го}
End;
Begin
  Init2(A);
  Print1(n,A);  {Вывод начального массива
                 из n элементов}
  Writeln('Номер элемента, после которого
           вставлять, ');
  Writeln('и значение нового элемента');
  Readln(k,x);
  Insert1(k,x,A);
  Print1(n+1,A);{Вывод массива после вставки
                 нового элемента}
  Readln;
End.
```

Вставка элемента перед элементом с данным номером

Пример

Вставить число 100 перед пятым элементом массива.

Решение

Эта задача немного отличается от предыдущей: в предыдущей мы сдвигали вправо все элементы, стоящие после k-го, то есть с (k+1)-го, а на его место записывали новый элемент, в этой — сдвигаем все элементы с k-го, а затем на его место записываем новый.

Пусть дан следующий одномерный массив из N (N=10) элементов: 3, –12, 5, 14, 27, –6, 1, 34, 10, –15. Надо вставить элемент со значением 100 перед пятым элементом массива. Получим следующий массив: 3, –12, 5, 14, 100, 27, –6, 1, 34, 10, –15.

```
Program Example_43;
Const n=10; dd=51;
Type myarray= Array[1..n+1] Of Integer;
Var  A: myarray;
     x, k:Integer;
    {x - значение нового элемента,
     k - номер  элемента, после которого
         вставляем}
Procedure Init2(Var  m:myarray);
{Процедура заполнения (инициализации) массива
 случайными числами}
...
Procedure Print1(n1: Integer; m: myarray);
{Процедура вывода (распечатки) массива}
...
Procedure Insert2(k1, x1: Integer; Var m: myarray);
Var i: Integer;
Begin
  {Сдвиг  на одну позицию вправо}
  For i:=n Downto k1 Do m[i+1]:=m[i];
  m[k1]:=x1; {Вставка x1  на место элемента
              с номером k1}
End;
Begin
  Init2(A);
  Print1(n,A);  {вывод начального массива}
  Writeln('Номер элемента, перед которым
           вставлять, ');
```

```
  Writeln('и значение нового элемента');
  Readln(k,x);
  Insert2(k,x,A);
  Print1(n+1,A); {Вывод массива после вставки
                  нового элемента}
  Readln;
End.
```

Вставка нескольких элементов

Предположим, что необходимо вставлять не один элемент, а по одному элементу после всех элементов с заданным свойством.

Пример

Вставить данное число после всех элементов массива, кратных 3.

Решение

Первое, на что необходимо обратить внимание, — это описание массива: на сколько элементов может увеличиться массив? Максимальное количество элементов, после которых будет вставлен новый элемент, совпадает с количеством элементов массива, ведь может случиться так, что все элементы массива обладают заданным свойством. Поэтому массив может увеличиться максимум в два раза, а значит, соответствующее описание будет следующим:

```
Type myarray = Array[1..2*n] Of Integer;
```

Если мы будем просматривать элементы массива с начала и вставлять новый элемент после элемента с заданным свойством, то номер последнего элемента каждый раз будет меняться, кроме того, придется пропускать ("перепрыгивать") новый (вставленный) элемент, поэтому решение будет не очень эффективным.

Удобнее просматривать массив с конца, тогда вставляемый элемент мешать не будет.

Составим программу.

```
Program Example_44;
Const n=10; dd=51;
Type myarray=Array[1..2*n] Of Integer;
Var A: myarray;
    x, k, i :Integer;
   {x - значение нового элемента,
    k - счетчик вставленных элементов}
Procedure Init2(Var m:myarray);
{Процедура заполнения (инициализации) массива
случайными числами}
...
Procedure Print1(n1: Integer; m: myarray);
{Процедура вывода (распечатки) массива}
...
Procedure Insert3(k1, x1: Integer;
                  Var m: myarray);
Var i: Integer;
Begin
  {Сдвиг элементов на одну позицию назад,
   n+k - это в данный момент номер последнего
   элемента}
  For i:=n+k Downto k1+1 Do m[i+1]:=m[i];
    m[k1+1]:=x1;
    {Вставка элемента на место - после k1-го}
    Inc(k); {Увеличение счетчика вставленных
             элементов}
End;
Begin
  Init2(A);
  Print1(n,A);
  Writeln('Введите вставляемое число');
  Readln(x);
  k:=0;
  For i:= n Downto 1 Do
  If A[i] Mod 3=0 Then Insert3(i,x,A);
  Print1(n+k,A); {Вывод массива после вставки
                  в него всех элементов}
  Readln;
End.
```

Решение задач

1. Вставить элемент с данным значением после первого отрицательного элемента массива.

2. Вставить элемент с данным значением перед последним отрицательным элементом массива.

3. Вставить в массив два элемента с данными значениями: первый — после максимального элемента, второй — перед максимальным элементом (удобнее всего вставлять элементы именно в таком порядке).

4. Вставить по одному элементу с данным значением перед всеми элементами массива, кратными заданному числу.

5. Вставить по одному элементу с данным значением перед всеми отрицательными элементами массива.

6. Вставить два элемента с данными значениями: первый — после всех элементов, больших данного числа P, а второй — перед всеми элементами, большими данного числа P (P вводится с клавиатуры).

7. Вставить элемент со значением A перед всеми элементами, большими A, а элемент со значением B — после всех элементов, меньших B.

2.3.5. Перестановки элементов массива

Перестановка двух элементов

Пример

Поменять местами значения двух элементов с номерами k1 и k2 (где k1 и k2 вводятся с клавиатуры).

Решение

Опишем процедуру, которой будем передавать номера переставляемых элементов и массив.

```
Procedure Swap(k1, k2: Integer; Var m: myarray);
Var x: Integer;
Begin
  x:=m[k1]; m[k1]:=m[k2]; m[k2]:=x;
End;
```

Примечание. Задача о перестановке двух элементов с заданными свойствами сводится к этой задаче — надо только найти их номера.

Перестановка нескольких элементов (части) массива

Пример

Дан одномерный массив A, состоящий из 2n элементов. Поменять местами его половины.

Решение

Пусть массив A состоит из 10 элементов, то есть n=5: 1, 12, 23, 3, 7, 13, 27, 6, 9, 11. Тогда если мы поменяем местами его половины, то получим такой массив A: 13, 27, 6, 9, 11, 1, 12, 23, 3, 7. Заметим, что мы меняем местами элементы с номерами 1 и n+1, 2 и n+2 и так далее; последняя пара — n и 2n. Легко заметить и то, что элемент с номером i меняется местами с элементом с номером n+i. Поэтому, используя процедуру Swap из примера, можно в основной программе применить цикл:

```
For i:=1 To n Do Swap(i,i+n,A);
```

Решение задач

1. Поменять местами:

a) первый и максимальный элементы массива;

b) второй и минимальный элементы массива;

c) первый и последний отрицательный элементы массива.

2. Дан одномерный массив A, состоящий из 2n элементов. Поменять его половины следующим образом: первый элемент поменять с последним, второй — с предпоследним и так далее.

3. Дан одномерный массив B, состоящий из 2n элементов. Переставить его элементы по следующему правилу:

a) b[n+1], b[n+2],..., b[2n], b[1], b[2],..., b[n];

b) b[n+1], b[n+2],..., b[2n], b[n], b[n-1],..., b[1];

c) b[1], b[n+1], b[2], b[n+2],..., b[n], b[2n];

d) b[2n], b[2n-1],..., b[n+1], b[1], b[2],..., b[n].

4. Дан одномерный массив. Переставить в обратном порядке элементы массива, расположенные между минимальным и максимальным элементами.

2.4. ДВУМЕРНЫЕ МАССИВЫ

2.4.1. Описание. Работа с элементами

Повторение

1. Как описать одномерный массив?

2. Как задать значения его элементов?

3. Как вывести массив на экран?

4. Как найти сумму элементов массива?

5. Как найти номер заданного элемента массива?

6. Как найти количество элементов массива с заданными свойствами?

7. Как определить, обладают ли все элементы массива некоторым свойством?

Описание

Двумерные массивы можно представить в виде прямоугольной таблицы или матрицы.

Рассмотрим матрицу A размерностью 2×3 (состоящую из двух строк по три элемента в каждой):

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

Положение каждого элемента определяется двумя числами: номером строки, в которой находится элемент, и номером столбца. Например, a_{12} — это элемент, стоящий в первой строке и во втором столбце.

Имеется несколько способов объявления двумерных массивов.

Способ 1. В Паскале двумерный массив можно описать как одномерный, элементами которого являются одномерные массивы. Например, для матрицы A, приведенной выше:

```
Const n=2; m=3;
Type omyarray=Array[1..m] Of real;
     dmyarray=Array[1..n] Of  omyarray;
Var v: omyarray;
    a: dmyarray;
```

В данном случае переменная v объявлена как одномерный массив из трех элементов вещественного типа. Переменная a описана как двумерный массив из двух строк, в каждой по три элемента.

Способ 2. Описание массива A можно сократить, исключив определение типа omyarray в определении типа dmyarray:

```
Const n=2; m=3;
Type dmyarray=Array[1..n] Of Array[1..m] Of
     <тип элементов>;
Var a: dmyarray;
```

Способ 3. Еще более краткое описание массива A можно получить, указывая диапазоны изменения индексов для каждой размерности массива:

```
Const n=2; m=3;
Type dmyarray=Array[1..n,1..m] Of <тип элементов>;
Var a: dmyarray;
```

Если нет необходимости описывать тип, то можно просто объявить массив в разделе описания переменных:

```
Var  a:  Array[1..n,1..m] Of <тип элементов>;
```

Рассмотренные выше методы решения задач обработки одномерных массивов могут применяться и для обработки двумерных массивов. Поскольку положение элемента в двумерном массиве описывается двумя ин-

дексами (первый — номер строки, второй — номер столбца), программы для решения большинства матричных задач строятся на основе вложенных циклов. Обычно внешний цикл организуется по строкам матрицы, то есть в нем выбирается требуемая строка матрицы, а внутренний цикл — по столбцам матрицы, в нем выбирается элемент строки. Для присваивания значений элементам массива могут быть использованы операторы присваивания и операторы ввода.

Пример

В приведенном ниже примере осуществляются ввод и вывод двумерного массива A размерностью 10×15. Формирование и вывод массива производятся в двух процедурах, которые последовательно вызываются из основной программы. Надо заметить, что формирование двумерного массива можно осуществлять всеми тремя способами, описанными для одномерных массивов, то есть: вводом с клавиатуры, посредством генератора случайных чисел или с помощью файла. Пусть в нашем примере элементы задаются посредством генератора случайных чисел.

```
Program Example_45;
Const n=10; m=15;
Type dmyarray = Array[1..n,1..m] Of Integer;
Var A: dmyarray;
Procedure Init(Var x: dmyarray);
{Процедура формирования массива}
Var i, j: Integer;
Begin
  For i:=1 To n Do
  For j:= 1 To m Do x[i,j]:=-25+Random(51);
End;
Procedure Print(x: dmyarray);
{Процедура вывода массива на экран}
Var i, j:  Integer;
Begin
  For i:=1 To n Do
  Begin    {Вывод i-й строки массива}
    For j:=1 To m Do Write(x[i,j]:5);
   Writeln; {Переход  на начало следующей строки}
  End;
End;
Begin
  Init(A);
  Writeln('Массив A:');
  Print(A);
  Readln;
End.
```

Можно выделить несколько видов задач на двумерные массивы.

Найти сумму элементов

Иногда необходимо найти сумму всех элементов, иногда — только некоторых, удовлетворяющих определенному условию. Мы рассмотрим более сложный пример.

Пример

Сформировать одномерный массив, каждый элемент которого равен сумме отрицательных элементов соответствующей строки заданной целочисленной матрицы.

Решение

Опишем одномерный массив, размерность которого равна количеству строк в двумерном массиве.

```
Const n=10; m=15;
Type omyarray=Array [1..n] Of Integer;
     dmyarray=Array[1..n,1..m] Of Integer;
Var  B: omyarray;
     A: dmyarray;
```

Формировать одномерный массив по заданному правилу будем в процедуре. Ей передаются два параметра — исходный двумерный массив и одномерный массив, который является результатом. В теле процедуры используются вложенные циклы. Внешний цикл определяет номер строки, который совпадает с номером элемента одномерного массива. Здесь же задаются начальные значения элементов одномерного массива, равные 0. Во внутреннем цикле анализируется каждый элемент выбранной строки. Если очередной элемент отрицательный, то он добавляется к сумме всех предыдущих отрицательных элементов выбранной строки матрицы.

```
Procedure Sum(x: dmyarray; Var y: omyarray);
Var i, j: Integer;
Begin
  For i:=1 To n Do
  Begin
    y[i]:=0; {Присваивание начальных значений
              элементов массива суммы}
    For j:=1 To m Do
    {Накопление суммы  отрицательных}
    If x[i,j]<0 Then y[i]:=y[i]+x[i,j];
  End;
End;
```

В основной программе после вызова процедуры Sum(A,B) остается только вывести на экран одномерный массив B, в котором записаны суммы отрицательных элементов каждой строки.

Нахождение количества элементов с данным свойством

Задачи на нахождение номеров элементов с заданными свойствами и на нахождение количества таких элементов во всем массиве решаются практически так же, как и для одномерных массивов. Необходимо лишь добавить второй цикл ко второму индексу.

Пример

Найти максимальный элемент массива и его индексы.

Решение

Так как элементы могут повторяться, то договоримся, что будем запоминать только индексы первого максимального элемента. Опишем процедуру, которой пе-

редается массив. Ее результатом являются значение максимального элемента и индексы первого элемента с максимальным значением.

```
Procedure Maximum(x: dmyarray; Var max, maxi,
                  maxj: Integer);
Var i, j: Integer;
Begin
  max:=x[1,1]; maxi:=1; maxj:=1;
 {Начальные значения}
  For i:=1 To n Do
  For j:=1 To m Do If x[i,j]>max Then
  Begin {Новые значения}
    max:=x[i,j];
    maxi:=i; maxj:=j;
  End;
End;
```

Пример

Найти количество отрицательных элементов в каждой строке.

Решение

Рассмотрим несколько вариантов решения этой задачи.

Можно хранить количество отрицательных элементов каждой строки в одномерном массиве соответствующей размерности.

```
Procedure Q_1(x: dmyarray; Var y: omyarray);
Var i, j: Integer;
Begin
  For i:=1 To n Do
  Begin
    y[i]:=0;
   For j:=1 To m Do If x[i,j]<0 Then Inc(y[i]);
  End;
End;
```

Можно использовать счетчик, находить количество отрицательных элементов строки и сразу выводить найденное значение на экран.

```
Procedure Q_2(x: dmyarray);
Var i, j, k: Integer;
Begin
  For i:=1 To n Do
  Begin
   k:=0;
   For j:=1 To m Do If x[i,j]<0 Then Inc(k);
   Writeln(i,'  -   ', k);
   {Вывод номера строки и количества
   отрицательных элементов}
  End;
End;
```

Работа с несколькими массивами

Пример

Составить программу вычисления произведения двух квадратных целочисленных матриц — A и B — размером 5×5. Элементы результирующей, также целочисленной, матрицы C (размером 5×5) определяются по формуле

$$C[i,j]=\sum_{k=1}^{n} a[i,k]\cdot b[k,j]$$, где n — размерность.

Решение

Формировать матрицы будем с помощью генератора случайных чисел, вычислять значения элементов результирующей матрицы C — посредством вложенных циклов. Во внутреннем цикле (по параметру k) будет накапливаться сумма, определяющая элемент c[i,j].

```
Program Example_46;
Const n=5;
Type dmyarray=Array[1..n,1..n] Of Integer;
Var A, B, C: dmyarray;
Procedure Init(Var x: dmyarray);
...
Procedure Print(x: dmyarray);
...
Procedure Mult(x,y: dmyarray; Var z: dmyarray);
Var  k, i, j:  Integer;
Begin
  For i:=1 To n Do
  For j:=1 To n Do
  Begin
    z[i,j]:=0;
    For k:=1 To n Do z[i,j]:=z[i,j]+x[i,k]*y[k,j];
  End;
End;
Begin
  Writeln('матрица A:'); Init(A); Print(A);
  Writeln('матрица B:'); Init(B);  Print(B);
  Mult(A,B,C);
  Writeln('матрица C:');Print(c);
  Readln;
End.
```

Определить, отвечает ли заданный массив некоторым требованиям

Пример

Определить, есть ли в данном массиве элемент, равный 0.

Решение

Опишем логическую функцию, значение которой равно "истина", если такой элемент есть, и "ложь" — в противном случае. Будем просматривать элементы массива, и если найден искомый, то присвоим функции значение True, иначе — False.

```
Function Check1(x: dmyarray): Boolean;
Var i, j: Integer;
    t: Boolean;
Begin
  t:=False; {Предполагаем, что искомого
             элемента в массиве нет}
  i:=1;
  While (not t) And (i<=n) Do
  Begin
    j:=1;
    While (j<=m) And (x[i,j]<>0) Do Inc(j);
    t:=(j<=m);
    {Определяем, найден ли искомый элемент;
     если просмотрена вся строка, то
     значение j=m+1 и t:=False,
     иначе элемент найден и t:=True}
```

```
    Inc(i);
  End;
  check1:=t;
End;
```

Пример

Определить, является ли данная квадратная матрица симметричной относительно своей главной диагонали.

Решение

Если для всех i=1, ..., n и j=1, ..., n, i>j выполняется равенство a[i,j]=a[j,i], то матрица является симметричной. Поэтому можно составить следующую функцию:

```
Function Check2(x: dmyarray): Boolean;
Var i, j: Integer;
    t: Boolean;
Begin
  t:= True;
  {Предположим, что матрица симметрична}
  i:=2;
  While t And (i<n) Do
  Begin
    j:=1;
    While (j<i) And (x[i,j]=x[j,i]) Do Inc(j);
    t:=(j=i);
    Inc(i);
  End;
  Check2:=t;
End;
```

Таким образом, если встретится хотя бы одна такая пара, что соответствующие элементы не будут равны, то функция вернет значение "ложь" (False).

Решение задач

1. Найти сумму и количество элементов в каждом столбце матрицы, удовлетворяющих заданному условию (хранить эти значения в одномерных массивах). Найти сумму и количество элементов:

a) кратных k1 или k2;

b) попадающих в промежуток от A до B;

c) являющихся простыми числами;

d) положительных, лежащих выше главной диагонали.

2. Найти сумму элементов в строках с k1-й по k2-ю.

3. Найти номера:

a) всех максимальных элементов (по строкам);

b) первых отрицательных элементов каждой строки (столбца);

c) последних отрицательных элементов каждой строки (столбца).

4. Найти количество элементов в каждой строке, больших (меньших) среднего арифметического элементов данной строки.

5. Найти произведение двух двумерных массивов A и B, если массив A имеет размерность n×m, а B — m×n. Укажите размерность результирующего массива и правило нахождения элемента с индексами i и j.

6. Даны две квадратные матрицы — A и B. Вывести на экран ту из них, у которой след (сумма элементов главной диагонали) меньше.

7. Изменить функцию в примере (Check1) так, чтобы просмотр элементов заканчивался в случае, когда найден нулевой элемент.

8. Изменить функцию в примере (Check2) так, чтобы просмотр пар прекращался тогда, когда найдена пара неравных элементов.

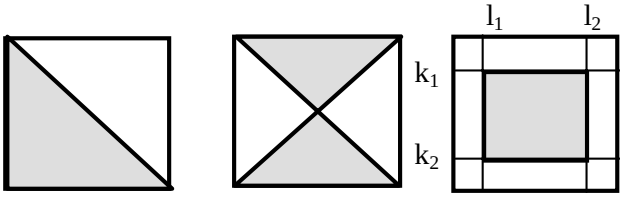
9. Определить:

a) есть ли в данном массиве отрицательный элемент;

b) есть ли в данном массиве два одинаковых элемента;

c) есть ли данное число A среди элементов массива;

d) есть ли в заштрихованной части массива элемент, равный A (массив имеет размерность n×n):



10. Определить, является ли матрица магическим квадратом. В магическом квадрате суммы элементов по всем горизонталям, вертикалям и двум диагоналям равны.

Решите задачу, добавив следующее условие: сумма указанных элементов должна быть равна данному числу A.

11. Определить, есть ли в данной матрице строка (столбец):

a) состоящая только из положительных элементов;

b) состоящая только из положительных или нулевых элементов;

c) состоящая только из элементов, больших числа A;

d) состоящая только из элементов, принадлежащих промежутку от A до B.

2.4.2. Двумерные массивы. Работа с элементами

Повторение

1. Как изменить значение одного элемента в одномерном массиве (с заданным номером, с заданным свойством)?

2. Что такое двумерный массив?

3. Как его описать?

4. Как заполнить двумерный массив? Приведите примеры заполнения двумерного массива различными способами.

5. Как вывести двумерный массив на экран?

Рассмотрим несколько видов задач.

Изменение значений некоторых элементов, обладающих заданным свойством

Решение таких задач похоже на решение задач для одномерных массивов.

Пример

В массиве размерностью n×m к элементам четных столбцов прибавить элемент первого столбца соответствующей строки.

```
Procedure Substitution1(Var x: dmyarray);
Var i, j: Integer;
Begin
  For i:=1 To n Do
  For j:=1 to m Div 2 Do
  {рассмотрим четные столбцы матрицы}
  Inc(x[i,2*j],x[i,1]);
End;
```

Пример

Заменить все отрицательные элементы на противоположные.

```
Procedure Substitution2(Var x: dmyarray);
Var i, j: Integer;
Begin
  For i:=1 To n Do
  For j:=1 To m Do x[i,j]:=abs(x[i,j]);
End;
```

Заполнение двумерного массива по правилу

Пример

Составить программу, запрашивающую координаты ферзя на шахматной доске и показывающую поля доски, находящиеся под боем (на доске нет других фигур).

Решение

Заметим, что шахматную доску удобно представить в виде двумерного массива размером 8×8. Координаты ферзя можно задать двумя числами (номером строки и номером столбца), но в шахматах принято указывать букву и число. Буква указывает номер строки, а число — номер столбца. Поэтому не будем отступать от традиций и введем координаты именно таким образом. В программе сделаем проверку правильности ввода, и если все правильно, то переведем букву в соответствующее ей число (a – 1, b – 2, c – 3, d – 4, e – 5, f – 6, g – 7, h – 8), тогда будет удобнее работать.

Для решения задачи полезно знать следующие свойства шахматной доски. Все диагонали делятся на восходящие и нисходящие:

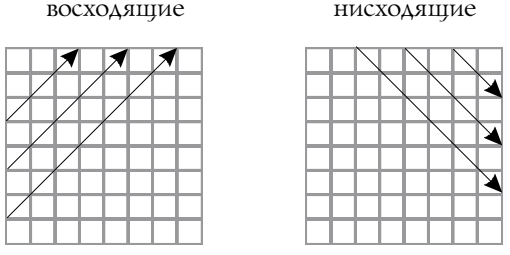


Рис. 7

Каждая диагональ обладает следующими свойствами:

- для элементов любой восходящей диагонали сумма номеров строки и столбца постоянна, причем для разных диагоналей — разная, то есть i+j=const1;
- для элементов нисходящих диагоналей разность номеров строки и столбца тоже постоянна и для разных диагоналей разная, то есть i–j=const2.

Это необходимо знать для того, чтобы определить номера диагоналей, на которых находится ферзь. Вся программа будет такой:

```
Program Example_47;
Const n=8;
Type dmyarray=Array[1..n,1..n] Of Integer;
Var  A: dmyarray;
     c: Char;
     str, stl: Integer;
     {str -  номер строки, stl - номер столбца}
Function Place(ch: Char): Integer;
Var k: Integer;
Begin
  Case ch Of
  'a': k:=1;  'b': k:=2;  'c': k:=3;  'd': k:=4;
  'e': k:=5;  'f': k:=6;  'g': k:=7;  'h': k:=8;
  End;
  Place:=k;
End;
Procedure Init(k, l: Integer;
               Var x: dmyarray);
{k  - номер строки,  l - номер столбца,
где стоит ферзь}
Var i, j: Integer;
Begin
  For i:=1 To n Do
  For j:=1 To n Do   {Под боем клетки,
  находящиеся с клеткой, на которой стоит
  ферзь, на одной вертикали, горизонтали,
  восходящей или нисходящей диагонали.
  Клетки, находящиеся под боем,
  мы помечаем 1, остальные помечаем 0.}
  If (i=k) Or (j=l) Or (i+j=k+l) Or (i-j=k-l)
  Then x[i,j]:=1 Else x[i,j]:=0;
  x[k,l]:=2; {На этой клетке стоит ферзь}
End;
Procedure Print (x: dmyarray);
Var i, j: Integer;
Begin
  For i:=1 To n Do
  Begin
    For j:=1 To n Do
    Case x[i,j] Of
    0: Write(' ':3);
    1: Write('*':3);
    2: Write('F':3);
  End;
  Writeln;
End;
End;
Begin
  Writeln('Введите координаты ферзя');
  Readln(c, stl);
  If (c<'a') Or (c>'h') Or (stl<1) Or (stl>n)
  Then Writeln ('Некорректный ввод')
  Else
```

```
  Begin
    str:= Place(c);
    Init(str,stl,A);
    Print(A);
  End;
  Readln;
End.
```

Можно функцию Place составить иначе:

```
Function Place(ch: Char): Integer;
Var k: Integer;
Begin
  Place:=Ord(ch) - Ord('a') +1;
End;
```

В этом случае она будет более рациональна.

Пример

Заполнить массив A размером n×m следующим образом (по "змейке"):

Например, при n=6 и m=8:

1	2	3	4	5	6	7	8
16	15	14	13	12	11	10	9
17	18	19	20	21	22	23	24
32	31	30	29	28	27	26	25
33	34	35	36	37	38	39	40
48	47	46	45	44	43	42	41

Решение

Для того чтобы заполнить массив указанным образом, надо вывести правило заполнения. В данном случае правило будет таким: если ряд нечетный (то есть когда номер строки — нечетное число), то A[i,j]=(i-1)*m+j, иначе (то есть когда строка четная) A[i,j]=i*m-j+1. В соответствии с этим правилом составляем процедуру заполнения массива:

```
Procedure Fill(Var x: dmyarray);
Var i, j: Integer;
Begin
  For i:=1 To n Do
  For j:=1 To m Do
  If i Mod 2=1 Then x[i,j]:=(i-1)*m+j
  Else x[i,j]:=i*m-j+1;
End;
```

Решение задач

1. В каждой строке двумерного массива сменить знак максимального по модулю элемента на противоположный.

2. Последний отрицательный элемент каждого столбца двумерного массива заменить нулем.

3. Положительные элементы двумерного массива умножить на первый элемент соответствующей строки, а отрицательные — на последний; то есть положительные элементы первой строки умножаем на первый элемент первой строки, а отрицательные — на последний элемент первой строки, то же самое и с остальными строками.

4. Заменить все элементы строки с номером k и столбца с номером l на противоположные по знаку (элемент, стоящий на пересечении, не изменять).

5. К элементам столбца k1 прибавить элементы столбца k2.

6. Написать программу, запрашивающую координаты коня и определяющую поля шахматной доски, находящиеся под боем.

7. Ввести координаты ферзя и коня и определить:

a) если конь ходит первым, то бьет ли он ферзя;

b) бьет ли ферзь коня, если первый ход ферзя.

8. Составить программу заполнения и вывода на экран таблицы умножения.

9. Даны два двумерных массива одинаковой размерности. Создать третий массив той же размерности, каждый элемент которого равен сумме соответствующих элементов первых двух.

10. Даны два двумерных массива — A и B — одинаковой размерности. Создать массив C, где каждый элемент равен 1, если соответствующие элементы A и B имеют одинаковый знак, иначе элемент равен 0.

11. Составить программу вывода на экран арифметического квадрата. В арифметическом квадрате первый столбец и первая строка заполнены 1, а каждый из остальных элементов равен сумме своих соседей сверху и слева.

12. Заполнить массив A размерности n×m, как показано на рисунке (приведены примеры для n=5 и m=7):

a)

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

b)

1	0	2	0	3	0	4
0	5	0	6	0	7	0
8	0	9	0	10	0	11
0	12	0	13	0	14	0

13. Заполнить массив B размерности n×n, как показано на рисунке (приведены примеры для n=6):

a)

1	12	13	24	25	36
2	11	14	23	26	35
3	10	15	22	27	34
4	9	16	21	28	33
5	8	17	20	29	32
6	7	18	19	30	31

b)

1	3	4	10	11	21
2	5	9	12	20	22
6	8	13	19	23	30
7	14	18	24	29	31
15	17	25	28	32	35
16	26	27	33	34	36

2.4.3. Вставка и удаление элементов

Повторение

1. Какие задачи вставки элементов мы решали ранее для одномерных массивов?

2. Как вставить один элемент после данного?

3. Как вставить новые элементы после всех элементов, удовлетворяющих заданному условию?

4. Как вставить один элемент перед данным?

5. Как вставить заданный элемент перед всеми элементами массива, удовлетворяющими некоторому условию?

6. Как удалить один элемент?

7. Как удалить несколько элементов?

Вставка строки

Мы уже рассматривали операции вставки для одномерных массивов. Обобщим их для двумерных.

Пример

Вставить строку из нулей после строки с номером k.

Решение

Для решения этой задачи необходимо:

1. Первые k строк оставить без изменения.
2. Все строки после k-й сдвинуть на одну вниз, это лучше начать с последней строки и идти до (k+1)-й.
3. Присвоить значения элементам (k+1)-й строки.

Кроме того, необходимо обратить внимание на размерность массива. Так как мы вставляем строку, то необходимо иметь одну строку "в запасе".

```
Const n=5; m=7;
Type dmyarray=Array[1..n+1,1..m] Of Integer;
Var A: dmyarray;
```

Теперь опишем процедуру вставки:

```
Procedure Insert(k1: Integer; Var x: dmyarray);
Var i, j: Integer;
Begin
  For i:=n Downto k1+1 Do
  For j:=1 To m Do x[i+1,j]:=x[i,j];
  {Элементу столбца с номером j присваиваем
   элемент этого же столбца, но из предыдущей
   строки}
  For j:=1 To m Do x[k1+1,j]:=0;
End;
```

Так как число строк меняется, то процедуру Print надо изменить. Она должна выводить указанное количество строк начиная с первой:

```
Procedure Print1(n1: Integer; x: dmyarray);
Var i, j: Integer;
Begin
  For i:=1 To n1 Do
  Begin
   For j:=1 To m Do Write(x[i,j]:4);
   Writeln;
  End;
End;
```

Часть основной программы будет такой:

```
Begin
  Init(A);
  Print(n,A);
  Writeln('Введите номер строки, после
           которой надо вставить новую
           строку');
  Readln(k);
  Insert(k,A);
  Print1(n+1,A);
  Readln;
End.
```

Примечания

1. Если необходимо вставить новую строку после строки, удовлетворяющей некоторому условию, то надо лишь найти номер этой строки — и задача сведется к решению уже рассмотренной.

2. Если надо вставлять новые строки после всех строк с заданным условием, то надо учесть это при описании массива. Заметим, что удобнее просматривать строки с последней и ввести счетчик вставленных строк.

3. Вставка новой строки перед строкой с номером k изменится только тем, что сдвигать назад надо не с (k+1)-й строки, а с k-й.

4. Если надо вставлять столбцы, то размерность массива увеличивается по столбцам, а все остальное практически не меняется: надо сдвинуть столбцы вправо и на данное место записать новый столбец.

Удаление строки

Пример

Удалить строку с номером k.

Решение

Для того чтобы удалить строку с номером k, необходимо:

- Сдвинуть все строки начиная с данной на одну вверх.
- Последнюю строку "обнулить", то есть всем элементам последней строки присвоить значение 0.

Описание массивов оставим прежним (размером n×m). Также в программе будем использовать процедуру вывода Print1 из предыдущего параграфа. Будем выводить на экран сначала все строки, а второй раз, после удаления, на одну меньше. Теперь опишем процедуру удаления строки с данным номером:

```
Procedure Delete(k1: Integer;
                 Var x: dmyarray);
Var i, j: Integer;
Begin
  For i:=k1 To n-1 Do
  For j:=1 To m Do x[i,j]:=x[i+1, j];
  For j:=1 To m Do x[n, j]:=0;
End;
```

Примечания

1. Задачу удаления строки, удовлетворяющей заданному условию, можно решить этим же способом, достаточно только найти ее номер.

2. Используя данную процедуру, можно решить еще одну задачу — удаление всех строк, которые обладают некоторым свойством.

Решение задач

1. Вставить первую строку после строки, в которой находится первый встреченный максимальный элемент (здесь и далее массив просматривается слева направо и сверху вниз).

2. Вставить второй столбец после первого же столбца, в котором все элементы положительны. Если такого столбца нет, то сообщить об этом.

3. Вставить нулевую строку и нулевой столбец перед строкой и столбцом, где находится первый минимальный элемент.

4. Вставить после всех строк, в которых есть заданное число A, последнюю строку.

5. Вставить перед всеми столбцами, в которых нет отрицательных элементов, второй столбец.

6. Вставить перед всеми строками, в которых есть 0, первую строку, а после всех столбцов, в которых есть отрицательные элементы, — первый столбец.

7. Удалить столбец, в котором находится минимальный элемент. Если такой элемент встречается несколько раз, то удалить все столбцы.

8. Удалить строку с номером k и столбец с номером l.

9. Удалить все столбцы, в которых нет нулевого элемента.

10. Удалить все строки и столбцы, на пересечении которых стоят отрицательные элементы.

2.4.4. Перестановка элементов массива

Повторение

1. Как поменять местами два элемента одномерного массива?

2. Как переставить две равные (по количеству элементов) части массива?

Перестановка двух элементов

Пример

Поменять местами два элемента массива A с заданными координатами (номерами строки и столбца).

Решение

Можно эту задачу решить несколькими способами.

Первый способ аналогичен перестановке элементов в одномерном массиве, когда в процедуру передаются индексы элементов и массив, в котором надо их поменять. Тогда процедура может быть такой:

```
Procedure Swap1(k1, l1, k2, l2: Integer;
                Var x: dmyarray);
Var c: Integer;
Begin
  c:=x[k1,l1]; x[k1,l1]:=x[k2,l2];x[k2,l2]:=c;
End;
```

Второй способ. Вспомним процедуру Swap, которая меняет местами значения двух целых переменных.

```
Procedure Swap(Var x, y: Integer);
Var z: Integer;
Begin
  z:=x; x:=y; y:=z;
End;
```

А теперь обратимся к ней, передавая данные элементы: Swap(A[k1,l1], A[k2,l2]).

Рассмотрим задачу о перестановке двух столбцов (строк), так как многие задачи используют это действие.

Пример

Поменять местами столбцы с номерами l1 и l2.

Эту задачу также можно решить несколькими способами. Составим процедуру, в которую будем передавать номера столбцов, и массив, в котором надо их переставить. Кроме того, добавим проверку корректности ввода данных номеров, так как если столбца с данным номером нет, то и переставлять ничего не надо. В самой процедуре можно использовать, например, процедуру Swap.

```
Procedure Swap2(l1, l2: Integer;
                Var x: dmyarray);
Var i: Integer;
Begin
  If ((l1<1) Or (l1>m)) Or ((l2<1) Or (l2>m))
  Then Writeln('Ввод неправильный')
  Else For i:=1 To m Do Swap(x[i,l1], x[i,l2]);
End;
```

Если применять первую процедуру Swap1, то после проверки правильности ввода данных будет такое обращение:

```
For  i:=1 To m Do Swap1(i,l1,i,l2,x);
```

Решение задач

1. Поменять местами первый максимальный и последний минимальный элементы (здесь и далее массив просматривается слева направо и сверху вниз).

2. В каждой строке поменять местами первый элемент и максимальный по модулю.

3. В каждой строке переставить первый отрицательный и последний положительный, если таких нет, то сообщить об этом.

4. Переставить вторую и предпоследнюю строки.

5. Поменять местами первую строку и строку, в которой находится первый нулевой элемент.

6. В двумерном массиве переставить строки следующим образом: первую с последней, вторую — с предпоследней и так далее. Если число строк нечетное, то средняя останется неизменной.

7. Дан двумерный массив A. Расставить его столбцы в следующем порядке:

a) последний, предпоследний, ..., второй, первый.

b) первый, последний, второй, предпоследний, третий, ...

8. Дан двумерный массив. Начиная с первой строки сдвинуть все строки на две вниз, а последние две перенести на место первых двух строк.

9. Сдвинуть столбцы двумерного массива на k вправо, а последние k столбцов поставить на место первых.

10. Сдвинуть столбцы двумерного массива на k влево, а первые k столбцов поставить на место последних.