Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ *C*. МАССИВЫ

ПОСОБИЕ
по выполнению контрольных
и лабораторных работ по дисциплине
«Вычислительная техника и программирование»
для студентов технических специальностей
дневной и заочной форм обучения

Электронный аналог печатного издания

Гомель 2007

УДК 004.43(075.8)
ББК 32.973-018.1я73
П78

Рекомендовано к изданию научно-методическим советом факультета автоматизированных и информационных систем ГГТУ им. П. О. Сухого
(*протокол № 2 от 12.02.2005 г.*)

Авторы-составители: *О. А. Кравченко*, *Д. А. Литвинов*

Рецензент: доц. каф. «Промышленная электроника» ГГТУ им. П. О. Сухого *Е. А. Храбров*

Программирование на языке *C*. Массивы : пособие по выполнению контрол.
П78 и лаб. работ по дисциплине «Вычислительная техника и программирование» для студентов техн. специальностей днев. и заоч. форм обучения / авт.-сост.: О. А. Кравченко, Д. А. Литвинов. – Гомель : ГГТУ им. П. О. Сухого, 2007. – 38 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://gstu.local/lib. – Загл. с титул. экрана.

ISBN 978-985-420-574-8.

Дается понятие массива, включены правила описания массивов в программах на языке *C*. Рассматриваются основные алгоритмы обработки одномерных массивов. Приводятся примеры программ на языке *C* для всех рассмотренных алгоритмов.

Для студентов технических специальностей дневной и заочной форм обучения.

УДК 004.43(075.8)
ББК 32.973-018.1я73

ISBN 978-985-420-574-8

1. МАССИВЫ В ЯЗЫКЕ *C*

1.1. Понятие массива

Массив – это совокупность элементов одного типа, имеющих одно имя и отличающихся только индексом (порядковым номером). Массивы могут состоять из арифметических данных, символов, строк, структур, указателей. Доступ к отдельным элементам массива осуществляется по имени массива и индексу (порядковому номеру) элемента.

При объявлении массива в программе определяется ***имя массива***, ***тип его элементов***, ***размерность*** и ***размер***. ***Размерность***, или количество измерений массива, определяется количеством индексов при обращении к элементам массива. Массивы бывают одномерные, двухмерные, трехмерные и т. д. ***Размер массива*** – это количество его элементов по соответствующим размерностям. Общий вид объявления массива:

<имя_типа> <имя_массива> [k1] [k2] … [kn];

где ***k1, k2, …, kn*** – количество элементов массива – константы или константные выражения по ***1, 2, …, n*** измерениям. Причем значения индексов могут изменяться от ***0*** до ***ki – 1***.

Такое объявление массива называют ***статическим***, поскольку предельное количество его элементов известно заранее и оно уже не может быть изменено в ходе выполнения программы. При работе с массивами необходимо следовать следующим правилам:

- современные трансляторы языка *C* не контролируют допустимость значений индексов, это должен делать программист;
- количество измерений массива не ограничено;
- в памяти элементы массива располагаются так, что при переходе от элемента к элементу наиболее быстро меняется самый правый индекс массива, т. е. матрица, например, располагается в памяти по строкам;
- имя массива является указателем-константой на первый элемент массива;
- операций над массивами в *C* нет, поэтому пересылка элементов одного массива в другой может быть реализована только поэлементно с помощью цикла;
- над элементами массива допускаются те же операции, что и над простыми переменными того же типа;
- ввод-вывод значений элементов массива можно производить только поэлементно;

• начальные значения элементам массива можно присвоить при объявлении массива.

Примеры объявления массивов:

int A [10]; //одномерный массив из 10 целочисленных величин
float X [20]; //одномерный массив из 20 вещественных величин
int a[5]={1, 2, 3, 4, 5}; //массив с инициализацией его элементов
int c[]={–1 , 2, 0, –4, 5, –3, –5, –6, 1}; // массив, размерность которого определяется числом инициализирующих элементов

Обращения к элементам одномерного массива могут иметь вид: ***A[0], A[1], A[2],…, A[9], A[2*3]***.

В *C* нет массивов с переменными границами. Но, если количество элементов массива известно до выполнения программы, можно определить его как константу с помощью директивы ***#define***, а затем использовать ее в качестве границы массива, например,

#define n 10;
Main ( )
{ int a[n], b[n]; // Объявление 2-х одномерных массивов

Если количество элементов массива определяется в процессе выполнения программы, используют ***динамическое*** выделение оперативной памяти компьютера.

1.2. Динамические массивы

Если до начала работы программы неизвестно, сколько в массиве элементов, в программе используют динамические массивы. Память под них выделяется с помощью оператора ***new*** во время выполнения программы. Адрес начала массива хранится в переменной, называемой ***указателем***. Например:

int n = 20;
*int *a = new int[n];*

Здесь описан указатель ***a*** на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью оператора ***new***. Выделяется столько памяти, сколько необходимо для хранения ***n*** величин типа ***int***. Величина ***n*** может быть переменной.

Примечание. Обнуление памяти при ее выделении не происходит. Инициализировать динамический массив нельзя.

Обращение к элементу динамического массива осуществляется так же, как и к элементам обычного массива. Например: **a*[0], a[1], …, a[9]***.

Можно обратиться к элементу массива другим способом: ***(a + 9), *(a + i)***, т. к. в переменной-указателе ***a*** хранится адрес начала массива. Для получения адреса, например 9-го его элемента, к этому адресу прибавляется ***9 · sizeof(int)*** (9 умножить на длину элемента типа ***int***), т. е. к начальному адресу ***a*** прибавляется смещение ***9***. Затем с помощью операции ***** (разадресации) выполняется выборка значения из указанной области памяти.

После использования массива выделенная динамическая память освобождается с помощью оператора: ***delete [ ] имя массива***. Так, например, для одномерного массива ***a***:

delete [ ] a;

Время «жизни» динамического массива определяется с момента выделения динамической памяти до момента ее освобождения.

2. АЛГОРИТМЫ ОБРАБОТКИ ОДНОМЕРНЫХ МАССИВОВ

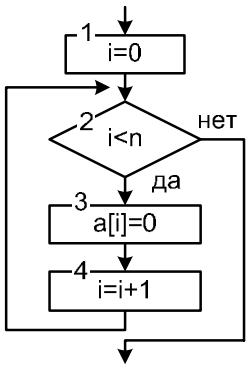
2.1. Инициализация массива

Инициализация массива – это присваивание элементам массива начальных значений. Инициализацию массива можно выполнить на этапе описания массива, как это показано в п. 1.1. Но в том случае, когда начальные значения получают лишь некоторые элементы массива, а остальные вычисляются в процессе выполнения программы, в программе записывают операторы присваивания. Например:

a[0]= –1; a[1]=1.1;

Присваивание всем элементам массива одного и того же значения осуществляется в цикле. Например, чтобы всем элементам массива ***a*** присвоить значение ***0***, можно воспользоваться алгоритмом, изображенном на рис. 2.1.

В представленном алгоритме все элементы массива в цикле последовательно инициализируются значением ***0***.

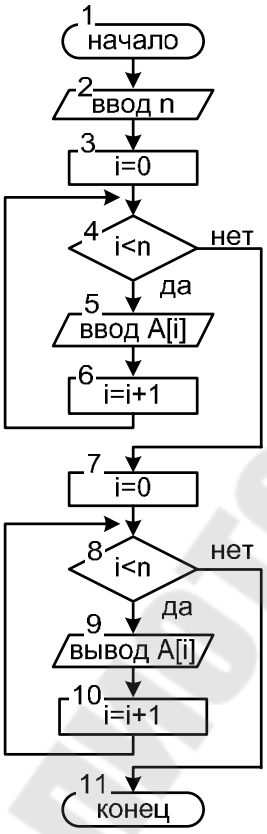



```
    for(i=0;i<n;i++)
       a[i]=0;
// или с помощью цикла while
    i=0;
    while (i<n)
    {
        a[i]=0;
         i=i+1;
    }
```

Рис. 2.1. Алгоритм и фрагмент программы инициализации массива

2.2. Ввод-вывод одномерного массива

Для ввода ***n*** элементов одномерного массива, назовем его ***A***, требуется организовать цикл, для ввода каждого ***i***-го элемента, где ***i = 0, 1, 2,…, n – 1***. Аналогичный цикл требуется организовать и для вывода элементов массива. На рис. 2.2 изображена графическая схема ввода и вывода элементов массива.




```
/* Ввод-вывод статического массива*/
#include <stdio.h>
#define n 50;
 void main()
{
  int n,i;
  float A[n];
  puts("Введите число элементов массива ");
  scanf("%d",&n);
// Ввод массива
  for (i=0; i<n; i++)
    {  printf("Введите число A[%2d]=",i);
       scanf("%f",&A[i]);
    }
  // Вывод массива
  puts("Массив A");
  for(i=0;i<n;i++)
    printf("%6.3f ",A[i]);
  printf("\n");
}
```

Рис. 2.2. Алгоритм и программа ввода-вывода статического массива

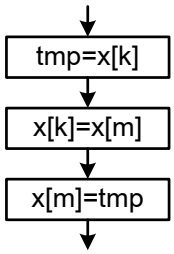
Ввод-вывод динамического массива осуществляется по тому же алгоритму. Из приведенного ниже примера программы ввода и вывода динамического массива видно, что отличие заключается лишь в описании массива.

```
/* Ввод-вывод динамического  массива*/
#include <stdio.h>
void main()
{
 int n,i;
 puts("Введите число элементов массива a");
 scanf("%d",&n);
 float *a=new float[n]; // Описание динамического массива
// Ввод массива
 for (i=0;i<n;i++)
   { printf("Введите число a[%2d]=",i);
     scanf("%f",a+i);      // или  scanf("%f",&a[i]);
   }
// Вывод массива
 puts("Массив a");
 for(i=0;i<n;i++)
   printf("%.3f ",*(a+i)); // или   printf("%.3f ",a[i]);
 printf("\n");
 delete[]  a;              // Освобождение памяти, выделенной под
массив
}
```

2.3. Перестановка двух элементов массива

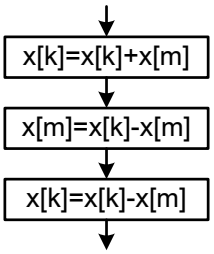
Для перестановки двух элементов массива ***x[]*** с индексами ***k*** и ***m*** необходимо использование дополнительной переменной (***tmp***), для хранения копии одного из элементов (рис. 2.3, *а*), но можно обойтись и без использования дополнительной переменной ***tmp***. В этом случае алгоритм перестановки имеет следующий вид (рис. 2.3, *б*).

В большинстве случаев предпочтительнее использовать первый способ, поскольку он не содержит дополнительных вычислений, что особенно важно при перестановке вещественных чисел.




```
tmp=x[k];
x[k]=x[m];
x[m]=tmp;
```

а)




```
x[k]=x[k]+x[m];
x[m]=x[k]-x[m];
x[k]=x[k]-x[m];
```

б)

Рис. 2.3. Алгоритм и фрагмент программы перестановки двух элементов массива с использованием:
а – дополнительной переменной; *б* – без нее

Пример 2.1

Переставить первый и последний элемент массива ***x[]*** местами. Количество элементов массива ***n***.

Решение

В *С* нумерация элементов массива начинается с нуля, поэтому номер последнего элемента массива (***n – 1***) .

1 способ: *tmp=x[0]; x[0]=x[n – 1]; x[n – 1]=tmp;*

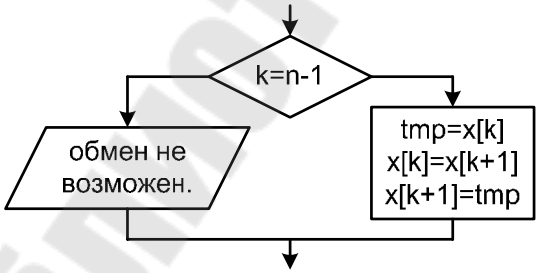
2 способ: *x[0]=x[0]+x[n – 1]; x[n – 1]=x[0] – x[n – 1];*
x[0]=x[0] – x[n – 1];

Пример 2.2

Поменять местами заданный элемент массива ***x[k]*** с последующим.

Решение

При решении этой задачи необходимо учитывать, что если заданный элемент массива ***x[k]*** является последним, то обмен выполнить невозможно, поскольку последующий элемент отсутствует.




```
if(k == n–1)
  puts("Обмен не возможен");
else
{
  tmp=x[k];
  x[k]=x[k+1];
  x[k+1]=tmp;
}
```

Рис. 2.4. Алгоритм и фрагмент программы перестановки заданного элемента массива ***x[k]*** с последующим

При перестановке с предыдущим элементом, обмен невозможен если заданный элемент является первым (***k = 0***).

2.4. Вычисление суммы элементов массива

Часто возникают задачи, требующие вычислить сумму всех или некоторых элементов массива, например, сумму элементов, стоящих в массиве на заданных местах, или сумму элементов, удовлетворяющих некоторому условию (сумму только положительных элементов, сумму ненулевых элементов второй половины массива и т. д.).

Пусть ***a[]*** – заданный массив из ***n*** элементов. Сумма всех его элементов в математической форме выглядит следующим образом:

$$s = a_0 + a_1 + ... + a_{n-1} = \sum_{i=0}^{n-1} a_i \ . \qquad (2.1)$$

Для вычисления суммы элементов части массива, например, с ***in***-го до ***ik***-го. Следует использовать формулу:

$$s = a_{in} + a_{in+1} + ... + a_{ik} = \sum_{i=in}^{ik} a_i \ . \qquad (2.2)$$

Очевидно, что формула (2.2) получается из формулы (2.1) при ***in = 0*** и ***ik = n – 1***.

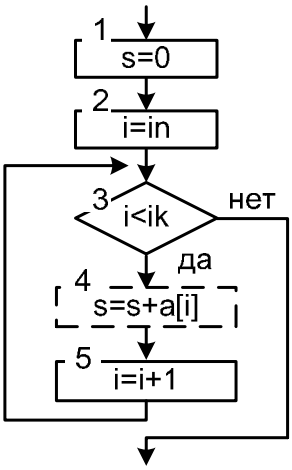
Алгоритм вычисления суммы состоит в следующем:

1) установить значение переменой для накопления суммы (***s***) в нулевое значение (***s = 0***);

2) в цикле изменяя ***i*** от ***in*** до ***ik***, вычислить сумму элементов массива по выражению $s = s + a_i$.

При первой итерации цикла (***i = in***) получим $s = s + a_{in} = 0 + a_{in}$. На второй (***i = in + 1***) – $s = s + a_{in+1} = a_{in} + a_{in+1}$ и т. д. На последней итерации цикла будем иметь $s = s + a_{ik} = a_{in} + a_{in+1} + ... + a_{ik}$, т. е. в цикле по параметру ***i*** «старое» значение ***s***, содержащее накопленную сумму на предыдущей итерации, изменяется на значение a_i. На рис. 2.5 представлен алгоритм и фрагменты программ вычисления суммы элементов массива.

Если в алгоритме (рис. 2.5) в блоке *2* записать ***i = 0***, а в блоке *3* – (***i < n)***, то получим алгоритм вычисления суммы всех элементов массива.

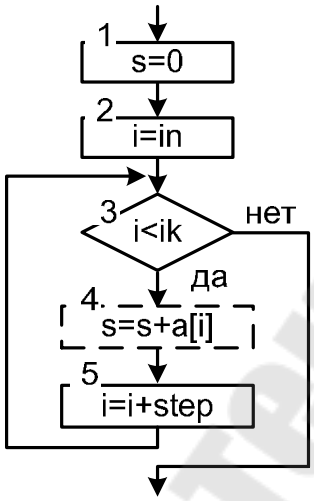



```
*/вычисление суммы элементов массива с in по ik
s=0;
for(i=in;i<ik;i++)
  s=s+a[i];     // или  s+=a[i]; или s+=*(a+i);

*/вычисление суммы всех элементов массива
s=0;
for(i=0;i<n;i++)
    s+=a[i];
```

Рис. 2.5. Графическая схема и фрагмент программы вычисления суммы элементов массива

Рассмотренный алгоритм вычисления суммы можно применить для ***вычисления суммы элементов, стоящих в массиве на заданных местах*** (рис. 2.6). В этом случае шаг изменения параметра цикла определяется переменной ***step***.




```
/* с помощью цикла  for */
s=0;
for(i=in;i<ik;i=i+step)
    s+=a[i];  // или s=s+a[i];
/* с помощью цикла  while */
s=0;   i=in;
while (i<ik)
{
    s+=a[i];
     i=i+step;
}
```

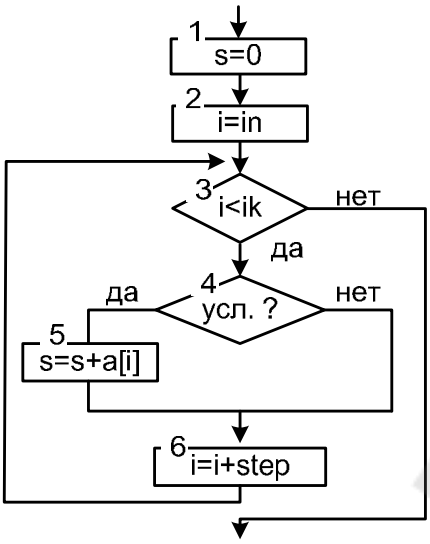
Рис. 2.6. Графическая схема и фрагмент программы вычисления суммы элементов массива, стоящих на заданных местах

Например, чтобы вычислить сумму элементов, стоящих в массиве на ***четных*** местах, необходимо «заставить» ***i*** принимать значения ***1, 3, 5, …*** (поскольку нумерация элементов массива в *C* начинается с нуля, т. е. элемент массива с индексом ***a[0]*** – первый элемент массива). Для этого достаточно в блоке *2* записать ***i = 1***, в блоке *3* – (***i < n)***,

а в блоке *5* записать ***i = i + 2*** (***step = 2***). В программе на языке *C* соответствующий фрагмент будет выглядеть следующим образом:

```
s=0;
for(i=1;i<n;i=i+2)      // или for(i=1;i<n;i+=2)
   s+=a[i];             // или s=s+a[i];
```

Для ***вычисления суммы только тех элементов, которые удовлетворяют некоторому условию***, необходимо в алгоритме вычисления суммы (рис. 2.6) блок *4* заменить на ветвление, которое обеспечивает выполнение команды $s = s + a_i$ только тогда, когда условие выполнено для рассматриваемого элемента массива a_i. В этом случае алгоритм вычисления суммы примет следующий вид (рис. 2.7).




```
/* с помощью цикла  for */
s=0;
for(i=in;i<ik;i+=step)
   if( условие )  s+=a[i];  // или s=s+a[i];
/* с помощью цикла  while */
s=0;   i=in;
while (i<ik)
{
   if( условие )  s+=a[i];
    i=i+step;
}
```

Рис. 2.7. Графическая схема и фрагмент программы вычисления суммы элементов массива стоящих на заданных местах

Применим полученный алгоритм для вычисления суммы положительных элементов массива, стоящих на нечетных местах. Для этого в блоке *2* запишем ***i = 0***, в блоке *3* (***i < n)***, в *4* условие – (***a[i] > 0***), а в блоке *6* изменение параметра цикла (***step = 2***) ***i = i + 2***. Тогда соответствующий фрагмент программы можно записать в виде:

```
s=0;
for(i=0;i<n;i+=2)
    if( a[i]>0 )  s+=a[i];  // или s=s+a[i];
```

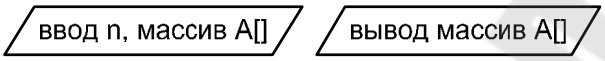
Рассмотрим примеры использования представленных алгоритмов.

Пример 2.3

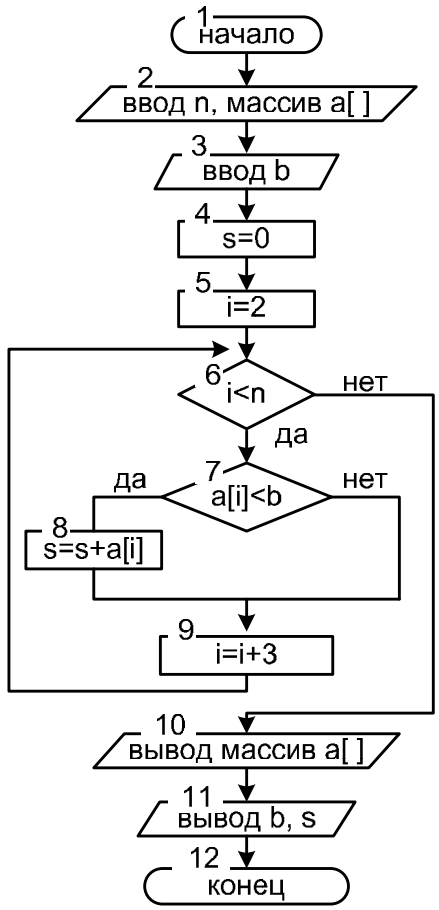
В одномерном массиве ***a*** размерностью ***n*** вычислить сумму элементов массива, меньших заданного значения ***B*** и стоящих на местах, кратных трем.

Решение

Объединим алгоритмы ввода-вывода массива (рис. 2.2) и вычисления суммы (рис. 2.7). Для сокращения записи графической схемы алгоритма ввода и вывода массива здесь и в дальнейшем используем простые блоки вида:



В алгоритме для вычисления искомой суммы рассматриваются только те элементы, которые в массиве стоят на местах, кратных трем, при этом необходимо учитывать, что нумерация элементов массива в *C* начинается с нуля, т. е. элемент массива с индексом ***a[0]*** – это первый элемент массива. Таким образом, элементы, стоящие на местах, кратных трем – a_2, a_5, a_8, …, индекс элемента массива (он же – параметр цикла) должен последовательно принимать значения ***2***, ***5***, ***8***, **…**, т. е. изменяться от ***2*** с шагом 3, что и достигается изменениями в блоках *2* и *6* алгоритма вычисления суммы (рис. 2.7). Так, в блоке *2* запишем ***i = 2***, в блоке *3* (***i < n)***, а в блоке *6* – (***step = 3***) ***i = i + 3***. Для суммирования из рассмотренных элементов только тех, которые меньше заданного ***B***, используется ветвление с условием $a_i < B$ (блок *4*). Окончательный алгоритм вычисления суммы заданных элементов примет следующий вид (рис. 2.8). В задаче будем использовать динамический способ задания массива. В данном примере для обращения к элементам массива используются указатели. Как уже отмечалось в разделе 1.1, имя массива является указателем на его первый элемент.



Используемые переменные:

n – число элементов массива;
a[] – динамический массив;
s – сумма элементов массива;
B – заданное число;
i – параметр цикла;

```
#include <stdio.h>
main()
{
 int n,i;
 float s, B;

 puts("Введите число элементов
массива a");
 scanf("%d",&n);

 float *a=new float[n];

 for (i=0;i<n;i++)
   { printf("Введите число
a[%2d]=",i);
     scanf("%f",a+i);
   }
 puts("Введите B");
 scanf("%f",&B);
 s=0;
 for(i=2;i<n;i+=3)
     if(*(a+i)<B) s+=*(a+i);
 puts("Массив a");
 for(i=0;i<n;i++)
   printf("%.1f ",*(a+i));
 printf("\n");
 printf("Сумма чисел, меньших %.1f,
стоящих на местах, кратных 3, рав-
на %.2f\n",B,s);
 delete[] a; // освобождение памяти
 return(0);
       }
```

Рис. 2.8. Графическая схема и программа примера 2.3

2.5. Подсчет количества элементов массива, удовлетворяющих заданному условию

Подсчет количества элементов массива, удовлетворяющих заданному условию, производится по алгоритмам, аналогичным вычислению суммы. Отличие заключается в том, что вместо добавления

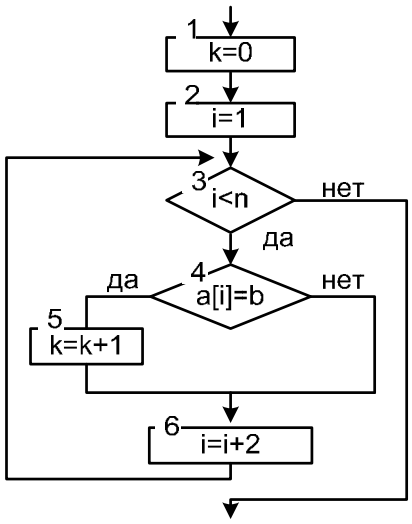
элемента массива к сумме переменная-счетчик (***k***) увеличивается на единицу (***k = k + 1***). Таким образом, если в графических схемах алгоритмов рис. 2.5–2.7 вместо ***s = 0*** и ***s = s + a$_i$*** записать ***k = 0*** и ***k = k + 1***, то получим алгоритмы подсчета количества элементов массива.

Пример 2.4

В одномерном массиве ***a*** размерностью ***n*** вычислить количество элементов, равных заданному числу ***B*** и стоящих на четных местах.

Решение

Графическая схема алгоритма решения задачи и фрагмент программы изображены на рис. 2.9.




```
/* с помощью цикла  for */
k=0;
for(i=1;i<n;i+=2)
    if(a[i]==b ) k++;  // или k=k+1;

/* с помощью цикла  while */
k=0;   i=1;
while (i<n)
{
    if(a[i]==b)  k++;
     i=i+2;
}
```

Рис. 2.9. Графическая схема и программа для примера 2.4

Следующий пример показывает, как в одном алгоритме находить сумму и количество элементов, удовлетворяющих заданному условию.

Пример 2.5

В одномерном массиве ***a*** размерностью ***n*** вычислить среднее арифметическое положительных элементов второй половины массива, стоящих на нечетных местах.

Решение

Среднее арифметическое чисел (***sr***) – частное от деления их суммы (***s***) на их количество (***k***): ***sr = s/k***, где ***≠ 0***. Таким образом, задача сводится к нахождению суммы и количества положительных эле-

ментов второй половины массива, стоящих на нечетных местах. Для решения данной задачи применим алгоритм, приведенный на рис. 2.7, в соответствии с которым можно найти сумму и количество части элементов массива (второй половины), удовлетворяющих заданному условию (положительных элементов).

Определим номер ***in*** того элемента, с которого будем просматривать элементы второй половины массива. Поскольку по условию задачи обрабатываются элементы, стоящие на нечетных местах, то начальное значение ***in*** тоже должно быть четным (поскольку нумерация элементов массива начинается с нуля). Значение переменной ***in*** можно определить по формуле (2.3), где пара квадратных скобок ***[ ]*** определяет операцию вычисления целой части числа:

$$in = \begin{cases} \left[\frac{n}{2}\right] + 1, \text{ если } \left[\frac{n}{2}\right] - \text{нечетное число;} \\ \left[\frac{n}{2}\right], \text{если } \left[\frac{n}{2}\right] - \text{четное число.} \end{cases} \qquad (2.3)$$

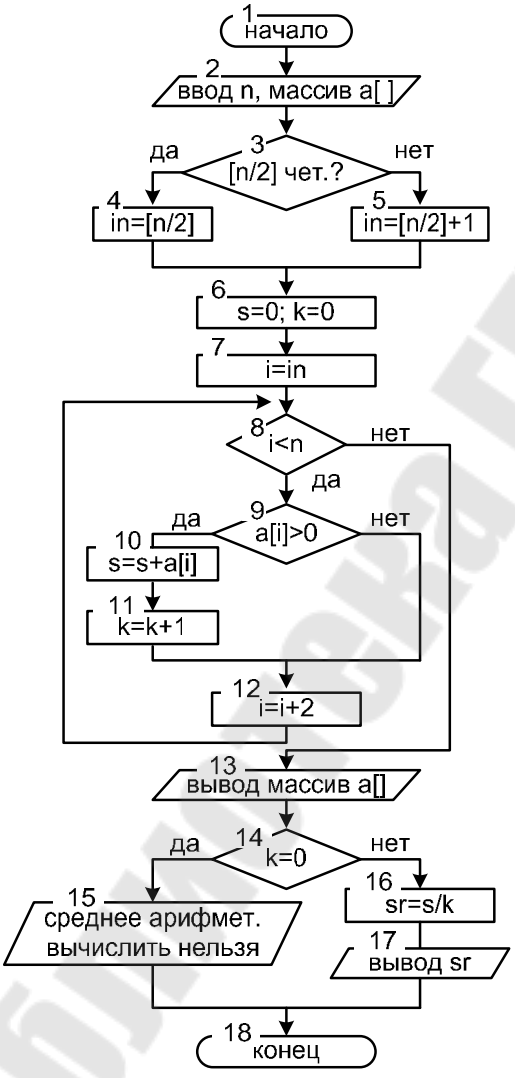
Значение ***ik*** совпадает с ***n***, поскольку массив надо просматривать до конца. Параметр цикла ***i*** (номер элемента массива) необходимо изменять с шагом 2, чтобы обеспечить четные значения индекса (значения номеров элементов массива), т. е. ***i = i + 2***. Вычисление суммы и количества будем вычислять в одном цикле по параметру ***i***, что значительно сократит алгоритм.

При вычислении среднего арифметического ***sr = s/k*** необходимо избежать возможного деления на ноль, поскольку если во второй половине массива на нечетных местах не окажется положительных элементов, то ***k*** будет равным нулю. Например, для массива ***X = {2, 0, –6, 7, 9, 0, –14, –5, 0, –4, –32}; n = 11; in = [n/2] + 1 = 6***, в цикле по ***i*** будут просматриваться только выделенные элементы, а среди них нет положительных чисел, поэтому переменная ***k*** останется равной нулю. В графической схеме (рис. 2.10) для этой цели используется ветвление, в котором проверяется условие ***k = 0***. Если это условие выполнено, то выводится сообщение о том, что среднее значение не может быть вычислено, в противном случае вычисляется и выводится значение ***sr***.

В приведенном ниже фрагменте программы четность [***n/2***] определяется по остатку от деления ***n*** на ***2*** с помощью операции ***%*** – определение остатка целочисленного деления.

Примечание. Результат операции целочисленный, т. к. ***n*** и ***2*** – целочисленные операнды.

Используемые переменные:
n – число элементов массива;
a[] – статический массив;
in – первый четный номер второй половины массива;
s – сумма элементов массива, удовлетворяющих условию;
k – количество элементов массива, удовлетворяющих условию;
sr – среднее значение элементов массива, удовлетворяющих условию;
i – параметр цикла;




```
#include <stdio.h>
main()
{
 float a[20];
 int n, i , in, k;
 float s, sr;

 puts("Введите число элементов массива a");
 scanf("%d",&n);

 for (i=0;i<n;i++)
   { printf("Введите число a[%2d]=",i);
     scanf("%f",&a[i]);
   }


 if ((n/2)%2==0) in=n/2;
 else in=n/2+1;

 s=0; k=0;
 for(i=in;i<n;i+=2)
     if(a[i]>0) { s+=a[i]; k++;}

 puts("Массив a");
 for(i=0;i<n;i++)
   printf("a[%2d]=%6.2f \n", i, a[i]);

 if (k==0)
   puts("Среднее арифметическое вычислить нельзя!");
   else
   {
     sr=s/k;
     printf("Среднее ариф. положж. элементов на нечетных местах второй полов. массива =%6.3f\n", sr);
   }
 return(0);
}
```

Рис. 2.10. Графическая схема и программа для примера 2.5

2.6. Вычисление произведения элементов массива

Формулы, по которым вычисляется произведение элементов массива, аналогичны формулам вычисления сумм:

$$p = a_0 \cdot a_1 \cdot \ldots \cdot a_{n-1} = \prod_{i=0}^{n-1} a_i ; \qquad (2.4)$$

$$p = a_{in} \cdot a_{in+1} \cdot \ldots \cdot a_{ik} = \prod_{i=in}^{ik} a_i . \qquad (2.5)$$

Поэтому вычисление произведения элементов массива выполнятся по алгоритмам, аналогичным вычислению суммы. Отличие заключается в том, что начальное значение произведения ***p*** должно быть равным ***1***, а в цикле по параметру ***i*** надо вычислять $\boldsymbol{p = p*a_i}$. Таким образом, если в графических схемах алгоритмов рис. 2.5–2.7 вместо $\boldsymbol{s = 0}$ и $\boldsymbol{s = s + a_i}$ записать $\boldsymbol{p = 1}$ и $\boldsymbol{p = p*a_i}$, то получим алгоритмы вычисления произведения элементов массива.

Пример 2.6

В одномерном массиве ***a*** размерностью ***n*** вычислить среднее геометрическое ненулевых элементов массива.

Решение

Среднее геометрическое ***k*** элементов массива – это корень степени ***k*** из произведения этих элементов. Таким образом, сначала необходимо вычислить произведение ***P*** ненулевых элементов массива и их количество ***k***, а затем среднее геометрическое ***Sg*** по формуле:

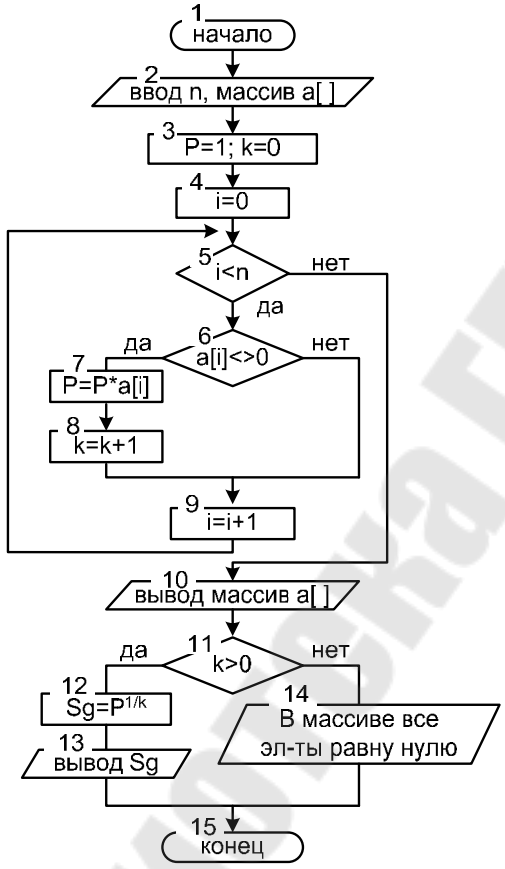
$$Sg = \sqrt[k]{P} = P^{\frac{1}{k}} . \qquad (2.6)$$

Например, если элементы массива равны $\boldsymbol{A = \{1, 0, 2, 4, 0\}}$, то $\boldsymbol{P = 8}$, $\boldsymbol{k = 3}$, $\boldsymbol{Sg = \sqrt[3]{P} = 2}$.

Графическая схема алгоритма решения задачи изображена на рис. 2.11. В приведенном алгоритме в цикле по ***i*** *(*блоки *5–9*) помимо вычисления произведения вычисляется и количество ненулевых элементов массива. После цикла с помощью ветвления проверяется, есть ли в массиве ненулевые элементы ($\boldsymbol{k > 0}$ – условие наличия в массиве ненулевых элементов), в этом случае вычисляется и выводится среднее геометрическое. В противном случае выводится сообщение:

«*В массиве все элементы равны нулю*». В программе переменные ***P*** и ***Sg*** имеют вещественный тип двойной точности (***double***), т. к. произведение вещественных чисел может быть очень большим числом.

Используемые переменные:
n – число элементов массива;
a[] – статический массив;
P – произведение ненулевых элементов массива;
k – количество ненулевых элементов массива;
Sg – среднее геометрическое элементов массива;
i – параметр цикла;




```
#include <stdio.h>
#include <math.h>

main()
{
 float a[20];
 int n, i , k;
 double P, Sg;
 puts("Введите число элементов массива a");
 scanf("%d",&n);
 for (i=0;i<n;i++)
   { printf("Введите число a[%2d]=",i);
     scanf("%f",&a[i]);
   }
 P=1; k=0;
 for(i=0;i<n;i++)
    if(a[i]!=0) {P*=a[i]; k++;}
 puts("Массив a");
 for(i=0;i<n;i++)
  printf("a[%2d]=%6.2f \n", i, a[i]);
 if(k>0)
 {
     if(P>0) Sg=powl(P,1.0/k);
      else Sg= –powl(fabs(P),1.0/k);
      printf("Среднее геометрическое ненулевых элементов массива =%.4lf \n", Sg);
     printf("P=%.4lf  k=%d \n", P, k);
  }
  else puts("В массиве все элементы равны нулю! ");
 return(0);
}
```

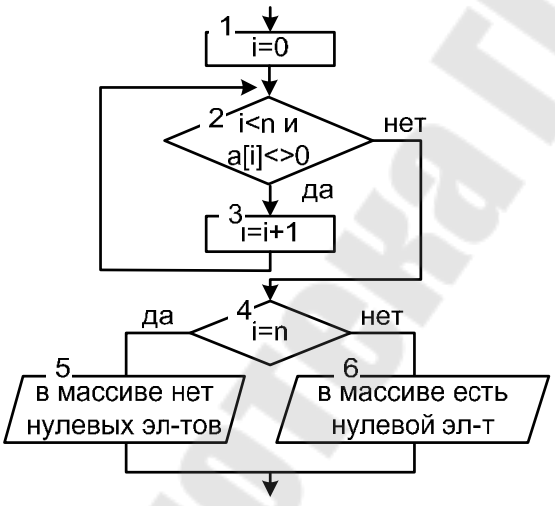
Рис. 2.11. Графическая схема и программа для примера 2.6

В программе для возведения ***P*** в степень ***1/k*** используется функция ***powl(основание, степень)***, первый аргумент которой может быть только положительным числом. Поэтому для отрицательного ***P*** использовано выражение $-|P|^{1/k}$, запись которого на языке *C* имеет вид: ***–powl(fabs(P), 1.0/k)***.

2.7. Поиск элементов, обладающих заданным свойством

При поиске элементов, обладающих заданным свойством, необязательно просматривать все элементы массива. Например, требуется определить, есть ли в массиве хотя бы один нулевой элемент. Для ответа на этот вопрос достаточно в цикле просматривать элементы массива до тех пор, пока не закончится массив или не встретится равный нулю элемент. Если, например, уже третий элемент равен нулю, то остальные элементы просматривать нет необходимости.

В таких случаях для просмотра массива обычно используется оператор цикла ***while*** со сложным условием. Графическая схема для рассматриваемого примера изображена на рис. 2.12. После цикла достаточно проверить, чему равно ***i***. Если окажется, что ***i = n***, т. е. были просмотрены все элементы, то в массиве нет нулевых элементов.



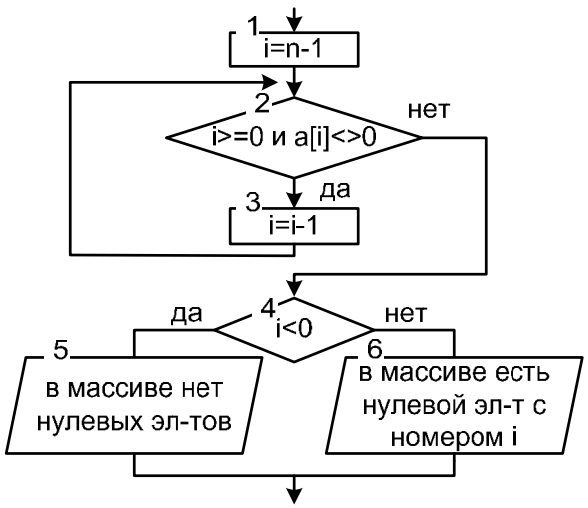

```
i=0;
while(i<n && a[i]!=0) i=i+1;

if(i == n)
   puts("В массиве нет нулевых элементов");
else
   puts("В массиве есть нулевой элемент");
```

Рис. 2.12. Графическая схема и фрагмент программы поиска нулевого элемента в массиве

Встречаются задачи, в которых требуется не только определить, есть ли элемент, обладающий заданным свойством в массиве, но и номер (индекс) такого элемента. Например, найти максимальный

элемент в части массива, находящейся после последнего нуля. Решение задачи следует начать с вычисления индекса последнего нулевого элемента. Для определения индекса самого правого элемента, обладающего заданным свойством, массив следует просматривать с конца до тех пор, пока не закончатся элементы и текущий элемент не станет равен нулю (рис. 2.13).




```
i=n–1;
while(i>=0 && a[i]!=0) i=i–1;

if(i<0)
   puts("В массиве нет нулевых элементов");
 else
   printf("Индекс последнего нуля – %d \n", i);
```

Рис. 2.13. Графическая схема и фрагмент программы поиска номера последнего нулевого элемента в массиве

Номер (индекс) первого встретившегося нулевого элемента можно узнать по значению параметра цикла ***i***. Этот номер можно использовать в дальнейших вычислениях, например, как номер начального элемента для поиска максимума.

2.8. Поиск в упорядоченном массиве

Упорядоченность элементов массива позволяет значительно увеличить скорость его обработки, за счет снижения числа проверяемых элементов массива. В таких алгоритмах массив проверяется, пока выполняется (или не выполняется) дополнительное условие, определяющее досрочный выход из цикла. Также при составлении алгоритма необходимо учитывать, возрастающим или убывающим является проверяемый массив, что оказывает влияние на то, как удобнее обрабатывать массив – с начала или с конца. В общем случае алгоритм обработки упорядоченного массива имеет следующий вид (рис. 2.14).

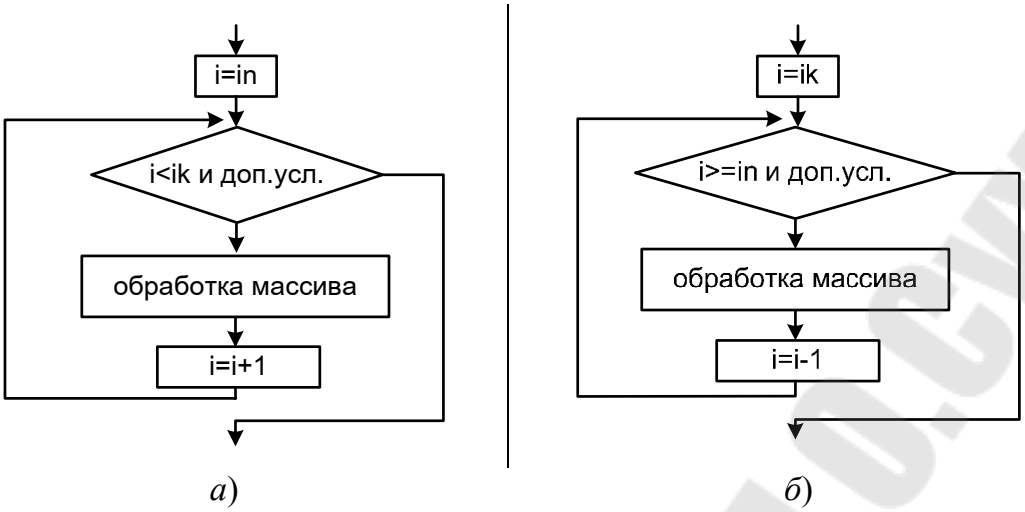


Рис. 2.14. Графический алгоритм обработки упорядоченного массива:
а – с перебором с начала; *б* – с конца

Как видно из блок-схемы, дополнительное условие управляет досрочным выходом из цикла. Пока дополнительное условие истина и не конец массива $i < n$, цикл выполняется, как только одно из условий будет не выполнено, происходит выход из цикла.

Пример 2.7

В возрастающем одномерном массиве ***X*** с количеством элементов ***n*** определить, есть ли число, равное ***A***, и на какой позиции оно находится; если числа ***A*** нет, определить место, на котором оно должно находиться, чтобы не нарушить упорядоченность массива.

Решение

В данной задаче обработку массива будем проводить с начала. Выход из цикла по дополнительному условию будет выполнен, если в массиве найден элемент, больший либо равный ***A*** (***k = 1***). Для индикации наличия в массиве элемента, равного ***A***, введем вспомогательную переменную ***f*** с начальным значением ***f = 0***. При обнаружении элемента ***A*** переменная ***f = 1***. Для определения номера позиции числа ***A*** в массиве введем дополнительную переменную ***poz*** с начальным значением ***n***, т. е. предполагая, что все элементы массива меньше ***A***. При обнаружении в массиве числа, большего или равного ***A***, в переменной ***poz*** сохраняется его индекс – ***i***. После выхода из цикла по значению переменной ***f*** определяется наличие и место переменной ***A*** в массиве. Описанный алгоритм поиска и программа представлены на рис. 2.15.

Используемые переменные:

n – число элементов массива;

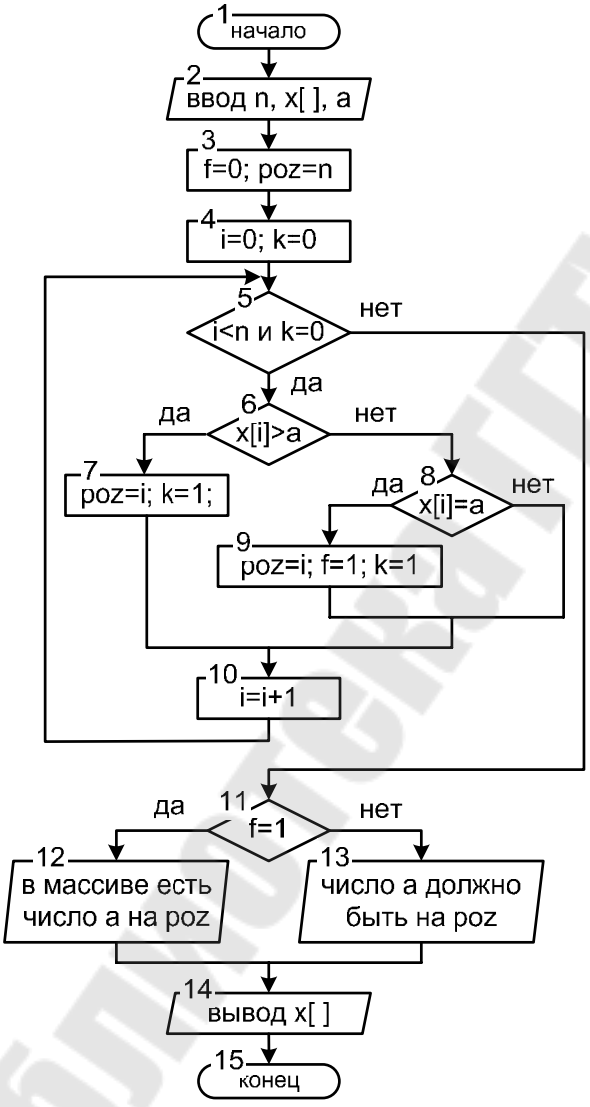
a[] – статический массив;

k – переменная для досрочного выхода из цикла при нахождении элемента, большего или равного a;

f – вспомогательная переменная для индикации наличия в массиве числа, равного a;

poz – номер элемента массива, на котором должно находиться число a;

i – параметр цикла;




```
#include <stdio.h>

main()
{
  int f, k, n, poz, i, x[10], a;
  puts("Введите число элементов
массива:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
   printf("x[%2d]=",i);
   scanf("%d",&x[i]);
  }
  puts("Введите число a:");
  scanf("%d",&a);
  f=0; poz=n; k=0;
  for(i=0;i<n&&k==0;i++)
  {
   if(x[i]>a) { poz=i;k=1;}
   else
   {
    if(x[i]==a)
      {poz=i; f=1; k=1;}
   }
  }
  if(f==1)
   printf("В массиве есть число
=%d, на позиции-%d\n", a, poz);
  else
   printf("Число %d должно на-
ходиться на позиции-%d\n",a,
poz);
 for(i=0;i<n;i++)
 printf("x[%d]=%d\n", i, x[i]);
  return 0;
}
```

Рис. 2.15. Графический алгоритм и программа для примера 2.7

Описанный алгоритм можно дополнить предварительным сравнением последнего элемента массива ***X[n–1]*** с числом ***A***, если ***X[n–1] = A*** – то заданное число находится на последнем месте, а в случае выполнения ***X[n-1] > A*** число ***A*** должно находиться в массиве на позиции ***n***. Если ни одно из этих условий не выполнено, то это означает, что необходимо выполнить поиск числа ***A*** в массиве.

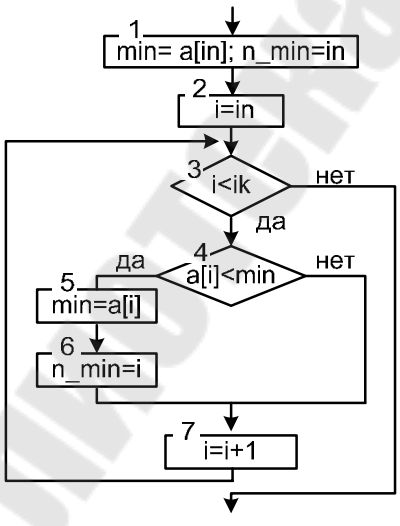
2.9. Поиск минимального и максимального элемента массива и его порядкового номера (индекса)

Пусть требуется найти минимальный элемент (***min***) и его индекс (***n_min***) во всем массиве (***in = 0*** и ***ik = n***) или какой-то его части (с ***in***-го по ***ik***-й), в этом случае алгоритм решения задачи можно записать так:

1) в качестве начального значения переменной ***min*** выберем любой из рассматриваемых элементов (обычно выбирают первый). Тогда ***min*** = a_{in}, ***n_min = in;***

2) затем в цикле по параметру ***i***, начиная со следующего элемента (***i = in + 1,…, ik***), будем сравнивать элементы массива a_i текущим минимальным ***min***. Если окажется, что текущий (***i***-й) элемент массива меньше минимального (a_i < ***min***), то переменная ***min*** принимает значение a_i, а ***n_min*** – на ***i***: ***min*** = a_i, ***n_min = i***.

Графическая схема алгоритма и фрагмент программы поиска минимального элемента в массиве приведены на рис. 2.16.



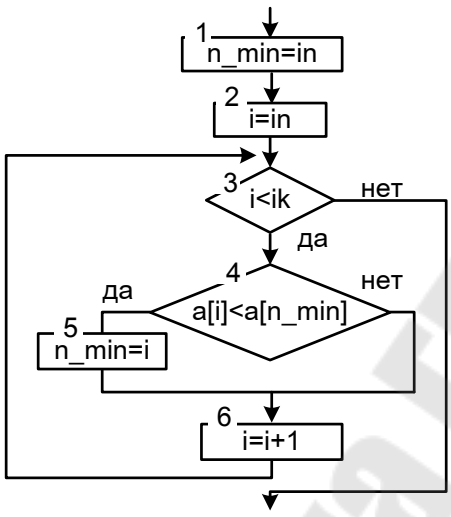

```
min=a[in];
n_min=in;
for(i=in+1; i<ik; i++)
  if(a[i]<min)
  {
     min=a[i];
     n_min=i;
  }
```

Рис. 2.16. Графический алгоритм и фрагмент программы поиска минимального элемента в массиве

Заметим, что при наличии в массиве нескольких минимальных элементов найден будет первый из них (самый левый минимальный элемент) при просмотре массива слева направо. Если в неравенстве $a_i < min$ знак $>$ поменять на знак $\geq$, то будет найден последний из них (самый правый минимальный элемент).

Для поиска максимального элемента ***max*** и его индекса ***n_max*** используется аналогичный алгоритм, в котором сначала надо принять $max = a_{in}$, $n\_ max = in$, вместо неравенства $a_i < min$ используется неравенство $a_i > max$. При выполнении условия $a_i > max$ записать в $max = a_i$ и в $n\_ max = i$.

Для поиска в массиве экстремума можно не использовать вспомогательную переменную ***min*** (***max***). В этом случае минимальный элемент массива определяется только по его индексу ***n_min*** (***n_max***) (рис. 2.17).



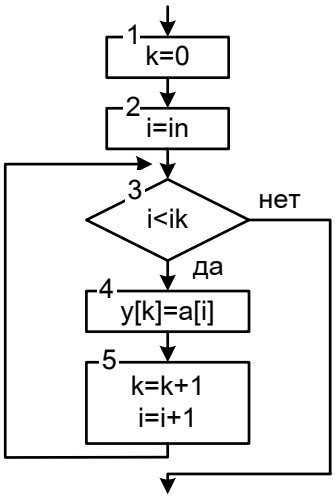

```
/*поиск минимального элемента*/
  n_min=in;
  for(i=in+1; i<ik; i++)
    if(a[i]<a[n_min])
      n_min=i;
/*поиск максимального элемента*/
  n_max=in;
  for(i=in+1; i<ik; i++)
    if(a[i]>a[n_max])
      n_max=i;
```

Рис. 2.17. Графический алгоритм и фрагмент программы поиска минимального элемента в массиве по его индексу

Пример использования рассмотренных алгоритмов представлен в приложении.

2.10. Копирование массивов

В ряде задач для организации дополнительных или промежуточных вычислений требуется создание копии всего массива или части его элементов. Для этого можно воспользоваться алгоритмом, представленным на рис. 2.18.




```
k=0;
for(i=in;i<ik;i++)
{
  y[k]=a[i];
  k++;
}
```

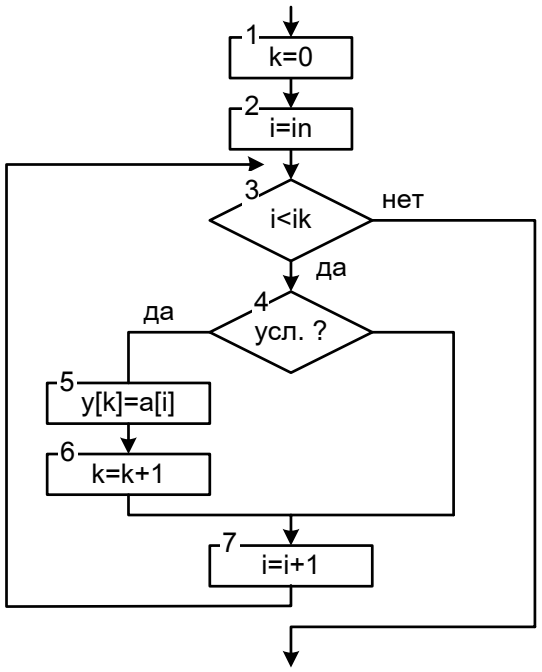
Рис. 2.18 Алгоритм и фрагмент программы создания копии массива

В зависимости от параметров ***in*** и ***ik*** в массив ***y[ ]*** копируются элементы из исходного массива ***a[ ]***. Так, для копирования всех элементов массива ***a[ ]*** необходимо задать ***in = 0***, ***ik = n*** (***n*** – количество элементов массива ***a[ ]***). При копировании части массива, например с ***3*** по ***9***, принимаем ***in*** = ***2*** (поскольку нумерация элементов массива в *C++* начинается с нуля) и ***ik = 9***.

2.11. Формирование нового массива

В задачах формирования нового массива требуется создать массив из элементов существующего массива (массивов), удовлетворяющих заданному условию. В новый массив элементы заносятся последовательно, начиная с нулевого индекса. Максимально число элементов в формируемом массиве может достигать количества элементов в исходном массиве (массивах), минимальное значение равняется нулю. В этом случае считается, что новый массив не сформирован.

При формировании новых массивов удобно использовать динамические массивы, поскольку число его элементов заранее не известно. Алгоритм создания нового массива схож с алгоритмом копирования (рис. 2.19).




```
k=0;
for(i=in;i<ik;i++)
{
  if (условие)
  {
    y[k]=a[i];
    k++;
  }
}
```

Рис. 2.19. Алгоритм и фрагмент программы формирования нового массива

Для последовательной записи элементов в новый массив используется дополнительная переменная ***k*** *– счетчик элементов в новом массиве*. Начальное значение этой переменной принимается равным *нулю*, т. е. считается, что в новом массиве нет элементов. При обнаружении в исходном массиве элемента, удовлетворяющего заданному условию, его значение заносится в новый массив на позицию ***k***, а после счетчик элементов увеличивается на единицу (***k = k + 1***). Таким образом, после обработки всего исходного массива по значению счетчика ***k*** можно определить, сформирован ли новый массив (***k > 0***) и сколько в нем элементов (***k***).

Пример 2.8

Даны два одномерных массива ***X*** и ***Y***. Необходимо сформировать массив ***Z*** из положительных элементов массива ***X***, стоящих на четных местах и элементов массива ***Y***, больших первого элемента массива ***X***.

Решение

Если число элементов массива ***X*** есть ***n***, а массива ***Y*** есть ***m***, то с учетом того, что из первого массива выбираются элементы, стоящие только на четных местах, максимальное число элементов в новом массиве ***Z*** может достигать ***m + n/2*** элементов. Поэтому для массива ***Z***

с помощью оператора динамического выделения памяти (***new***) выделим ***m + [n/2]*** ячейки памяти (***[n/2]*** – целая часть от деления). Начальное значение счетчика элементов нового массива ***k*** принимается равным ***0***.

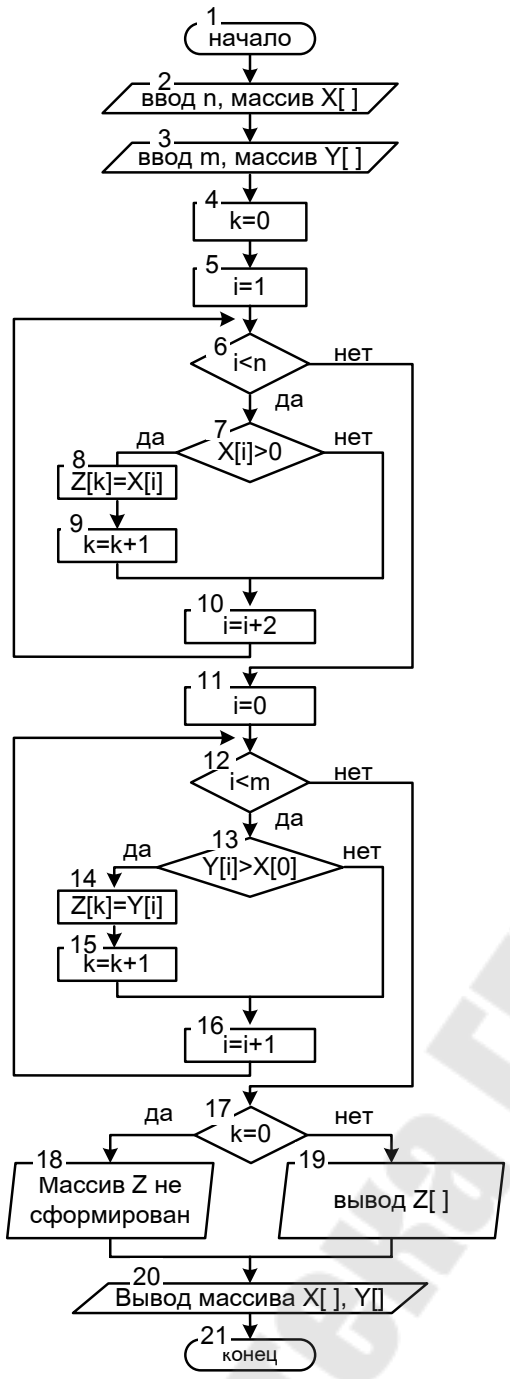
При обработке массива ***X*** необходимо проверять только элементы, стоящие на четных местах, т. е. параметр цикла ***i*** изменяется от ***in = 1*** до ***ik = n*** с шагом ***2***. Условие отбора элементов из первого массива ***X[i] > 0***. При обработке массива ***Y*** учитываются все его элементы, т. е. параметр цикла ***i*** изменяется от ***in = 0*** до ***ik = m*** с шагом ***1***. Условие отбора элементов из второго массива – ***Y[i] > X[0]***.

Описанный алгоритм формирования нового массива и программа представлены на рис. 2.20.

Используемые переменные:

x[] – статический (исходный) массив;
n – число элементов массива X;
y[] – статический (исходный) массив;
m – число элементов массива;
z[] – динамический (формируемый) массив;
k – счетчик элементов нового массива Z;
i – параметр цикла;

```
#include <stdio.h>
main()
{
  int  k, n, m, i, x[10], y[10];
  puts("Введите число элементов массива X:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
   printf("x[%2d]=",i);
   scanf("%d",&x[i]);
  }
  puts("Введите число элементов массива Y:");
  scanf("%d",&m);
  for(i=0;i<m;i++)
  {
   printf("y[%2d]=",i);
   scanf("%d",&y[i]);
  }
  int *z=new int[15]; // выделение памяти под массив Z
  k=0;
  for(i=1;i<n;i+=2)
  {
   if(x[i]>0)
   {
```




```
    z[k]=x[i];
    k++;
   }
  }
 for(i=0;i<m;i++)
 {
  if(y[i]>x[0])
  {
   z[k]=y[i];
   k++;
  }
 }
 puts("Массив X:");
 for(i=0;i<n;i++)
  printf("x[%d]=%d\n",i,x[i]);
 puts("Массив Y:");
 for(i=0;i<m;i++)
  printf("y[%d]=%d\n",i,y[i]);
 if(k==0)
  puts("Массив Z не сформиро-
ван");
 else
 {
   puts("Массив Z:");
   for(i=0;i<k;i++)
   printf("z[%d]=%d\n",i,z[i]);
 }
 delete[] z; // освобождение памяти
}
```

Рис. 2.20. Графический алгоритм и программа для примера 2.8

Литература

1. Кравченко, О. А. Программирование ввода-вывода данных и линейных вычислительных алгоритмов на языке *C* : практ. пособие к выполнению лаб. и контрол. работ по дисциплине «Вычислительная техника и программирование» для студентов техн. специальностей днев. и заоч. форм обучения / О. А. Кравченко, А. М. Мартыненко. – Гомель : ГГТУ им. П. О. Сухого, 2005. – 33 с.
2. Кравченко О. А., Программирование разветвляющихся и циклических алгоритмов на языке *C* : пособие по выполнению лаб. и контрол. работ по дисциплине «Вычислительная техника и программирование» для студентов техн. специальностей днев. и заоч. форм обучения / О. А. Кравченко, Е. В. Коробейникова. – Гомель : ГГТУ им. П. О. Сухого, 2005. – 33 с.
3. Информатика. Базовый курс : учеб. пособие / под ред. С. В. Симоновича. – 2-е изд. – Санкт-Петербург : Питер, 2006. – 639 с. : ил.
4. Павловская, Т. А. *C*/*C* ++. Программирование на языке высокого уровня / Т. А. Павловская. – Санкт-Петербург : Питер, 2006. – 460 с. : ил.
5. Павловская, Т. А. *C#*. Программирование на языке высокого уровня / Т. А. Павловская. – Сант-Петербург : Питер, 2006. – 432 с. : ил.
6. Острейковский, В. А. Информатика : учеб. для вузов / В. А. Острейковский. – Москва : Высш. шк., 2000. – 511 с. : ил.
7. Информатика : учебник / под ред. Н. В. Макаровой. – Москва : Финансы и статистика, 1998.
8. Касаткин, А. И. Профессиональное программирование на языке СИ: от Turbo *C* к Borland *C*++ : справ. пособие / А. И. Касаткин, А. Н. Вальвачев. – Минск : Выш. шк., 1992. – 240 с.
9. Топп, У. Структуры данных в *C*++ / У. Топп, У. Форд ; пер. с англ. – Москва : БИНОМ, 1994. – 816 с.
10. Крячков, А. В. Программирование на *C* и *C*++. Практикум : учеб. пособие для вузов / А. В. Крячков, И. В. Сухина, В. К. Томшин. – Москва : Горячая линия – Телеком, 2000. – 344 с.
11. Страуструп, Б. Язык программирования Си++ / Б. Страуструп ; пер. с англ. – Москва : Радио и связь, 1991. – 352 с.

ПРИЛОЖЕНИЕ

Примеры решения задач по обработке одномерных массивов

Задача 1. Вычисление сумм, количеств и произведений элементов массива

Предполагается, что задан массив чисел. Программа должна:

1) вводить размерность и элементы массива;
2) вводить некоторые дополнительные числа;
3) выполнять действия в соответствии с условием задачи;
4) выводить исходные данные и результаты вычислений.

Исходные данные для отладки программы выбрать самостоятельно. Массив объявить как **статический**.

Задание

В одномерном массиве ***A*** размерностью ***n***, найти количество чисел, меньших заданного ***X***, и произведение отрицательных чисел, стоящих на четных местах.

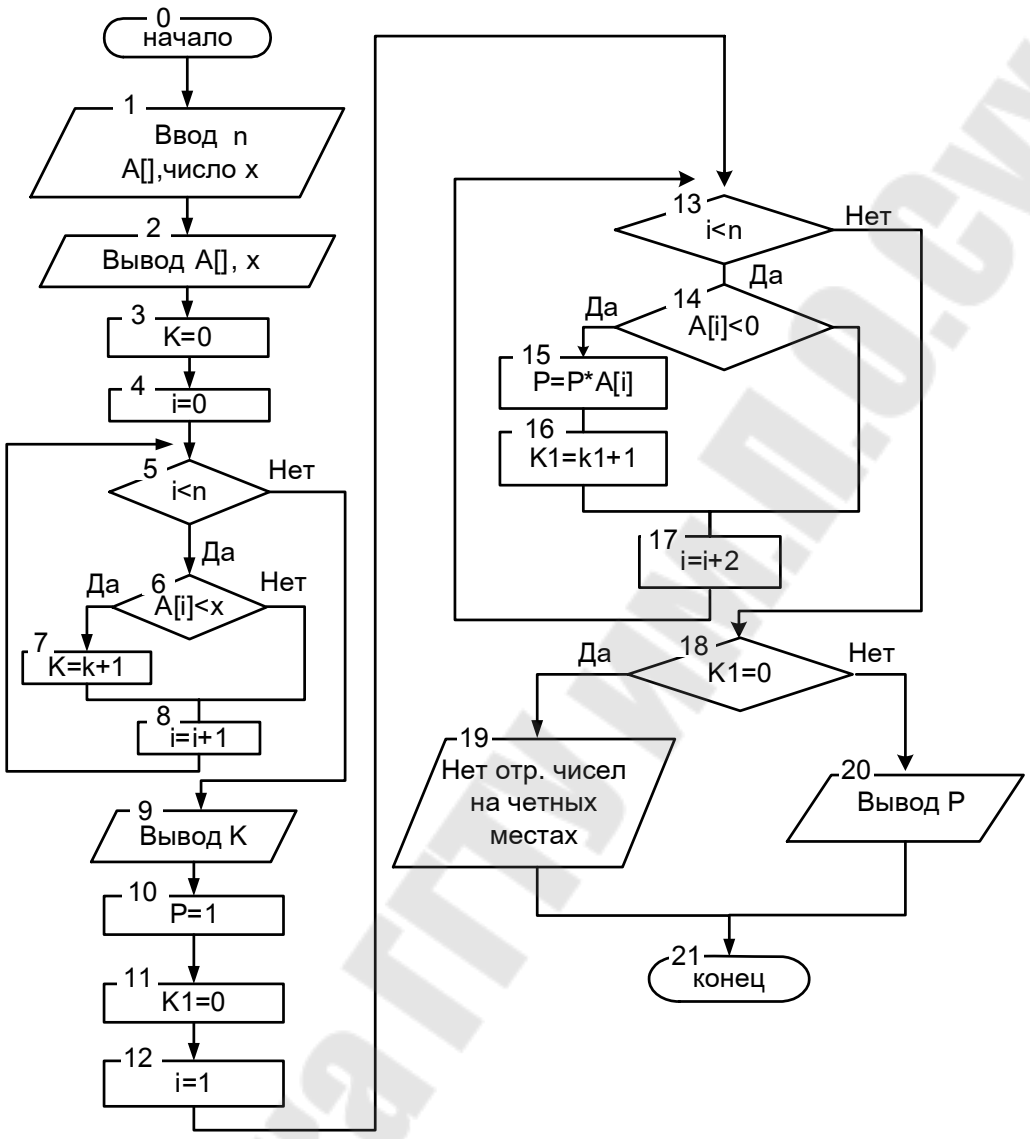
Решение

Таблица П.1

Соответствие переменных

Переменные в задаче	**Имя на языке *C***	**Тип**	**Комментарий**
A[]	*A[]*	*float*	Одномерный статический массив
n	*n*	*int*	Количество элементов массива
P	*P*	*float*	Произведение отрицательных чисел
X	*x*	*float*	Заданное число
k	*k*	*int*	Количество чисел, меньших *X*
–	*k1*	*int*	Количество отрицательных чисел
–	*i*	*int*	Номер элемента массива; параметр цикла

Графическая схема алгоритма



Описание алгоритма

1. Блоки *1–2* – вводятся исходные данные.

2. Блоки *3–9* – вычисление и вывод количества ***K***, меньших заданного ***X***.

3. Блоки *10–18* – вычисление количества ***K1*** и произведения ***P*** отрицательных элементов массива, стоящих на четных местах.

4. Блоки *18–20* – проверка на наличие в массиве отрицательных чисел на четных местах и в случае выполнения проверки вывод их произведения ***P***.

Листинг программы

```
#include <stdio.h>
#include <math.h>
main ()
{
 float P, A[10], x;                  //Описание переменных
 int i, k1, k, n;
 puts (" Введите число x");
 scanf ("%f", &x);                   // Ввод x
 puts (" Введите число элементов массива A");
 scanf ("%d", &n);
 for (i=0; i<n; i++)                 // Ввод массива
 {
  printf("Введите число A[%d]=", i);
  scanf("%f", &A[i]);
 }
 puts("Массив A");                   // Вывод массива
 for (i=0; i<n; i++)
  printf("%.2f ", A[i]);
 printf("\n");
 printf(" x=%.3f \n", x);            // Вывод x
 k=0;
 for (i=0; i<n; i++)
   if (A[i]<x) k=k+1;                // Определение кол-ва чисел <x
 printf(" k=%d\n" ,k);
 P=1;
 k1=0;
 for (i=1; i<n; i=i+2)
    if (A[i]<0)
    {                                // Вычисление произведения
     P=P*A[i];                       // отрицательных чисел,
     k1=k1+1;                        // стоящих на четных местах
    }
  if (k1==0)
     printf("В массиве на чётных местах нет отрицательных
чисел \n");
   else
  printf(" P=%6.2f \n" ,P);
 }
```

Тесты

1) *Массив A*
3.00 6.00 10.00 -7.00 -3.00 -7.00 -5.00 17.00 6.00 10.00
x=10.000
k=7
P= 49.00
2) *Массив A*
2.00 9.00 4.00 6.00 7.00 8.00
x=0.000
k=0
В массиве на чётных местах нет отрицательных чисел

Задача 2. Вычисление сумм, количеств и произведений элементов массива

Предполагается, что задан массив чисел. Программа должна:
1) вводить размерность и элементы массива;
2) вводить некоторые дополнительные числа;
3) выполнять действия в соответствии с условием задачи;
4) выводить исходные данные и результаты вычислений.

Исходные данные для отладки программы выбрать самостоятельно. Массив объявить как **динамический**.

Задание

В одномерном массиве ***A*** размерностью ***n*** найти максимальный элемент, расположенный между первым и последним нулевыми элементами массива.

Решение

Таблица П.2

Соответствие переменных

Переменные в задаче	**Имя на языке *C***	**Тип**	**Комментарий**
max	*max*	*int*	Максимальное число
A[]	*A[]*	*int*	Одномерный динамический массив
n	*n*	*int*	Количество элементов
–	*t1*	*int*	Индекс первого нулевого элемента
–	*t2*	*int*	Индекс последнего нулевого элемента
–	*i*	*int*	Номер элемента массива; параметр цикла

Описание алгоритма

1. Блоки *1–2* – ввод-вывод исходного массива ***A[]***.

2. Блоки *3–5* – поиск первого нулевого элемента в массиве.

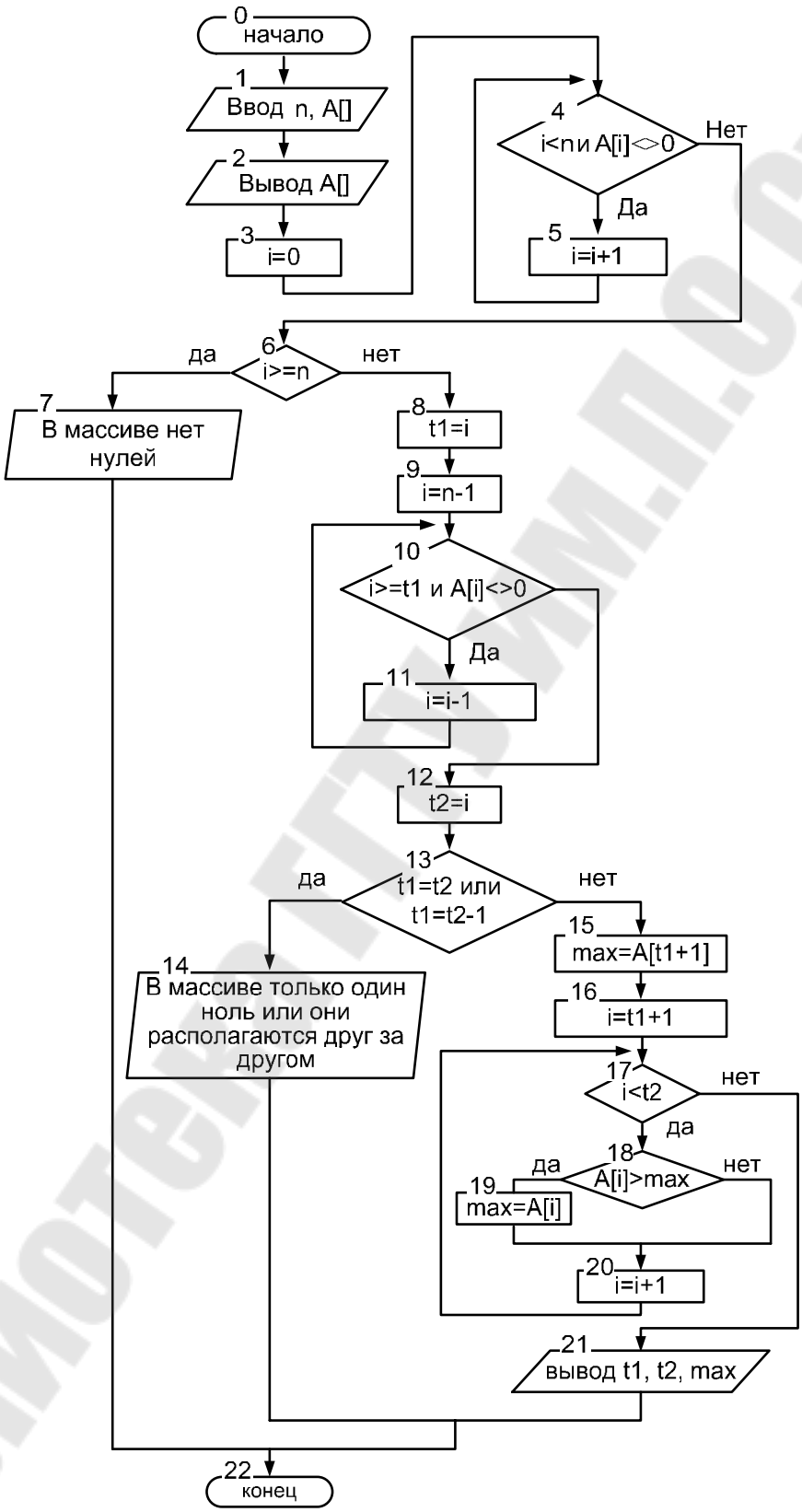
3. Блоки *6–7* – проверка на наличие в массиве хотя бы одного нулевого элемента и в случае их отсутствия вывод сообщения «*В массиве нет нулей*» и переход на конец алгоритма.

4. Блоки *8–12* – поиск последнего нулевого элемента в массиве и сохранение позиции первого нуля в ***t1***, а последнего в ***t2***.

5. Блоки *13–14* – проверка расположения нулевых элементов в массиве и в случае ошибки вывод сообщения «*В массиве только один ноль или они располагаются друг за другом*» » и переход на конец алгоритма.

6. Блоки *15–21* – в случае выполнения проверки (блок *13*) поиск максимального элемента между крайними нулевыми элементами и вывод полученного результата.

Графическая схема алгоритма



Листинг программы

```
#include <stdio.h>
main()
{
  int i,t1,t2,n,max;
  puts("Введите число элементов массива A");
  scanf ("%d", &n);
  int*A=new int[n];   //выделение памяти под массив
  for(i=0;i<n;i++)    //ввод массива
  {
    printf("a[%2d]=",i);
    scanf("%d",&A[i]);
  }
  puts("Массив A:");
  for(i=0;i<n;i++)    //вывод массива
    printf("a[%d]=%d\n",i,A[i]);
  i=0;
  while(i<n && A[i]!=0) i=i+1;  //поиск позиции первого нуля
  if(i>=n)
    printf("В массиве нет нулей \n");
  else
  {
    t1=i;
    i=n-1;
    while(i>=t1 && A[i]!=0) i=i-1; //поиск позиции последнего нуля
    t2=i;
    if(t1==t2 || t1==t2-1)
       printf("В массиве только один ноль или они располагаются друг
за другом\n");
    else
    {
       max=A[t1+1];
       for(i=t1+1;i<t2;i++)
        if(A[i]>max) max=A[i];   //поиск максимального элемента
       printf("t1=%d t2=%d max=%d \n",t1,t2,max);
    }
  }
 delete[] A;          // освобождение динамической памяти
}
```

Тесты

1) *Массив А*:
a[0]=1
a[1]=3
a[2]=0
a[3]=x6
a[4]=-3
a[5]=7
a[6]=4
a[7]=0
a[8]=1
t1=2 t2=7 max=7

2) *Массив А*:
a[0]=0
a[1]=1
a[2]=-2
a[3]=4
a[4]=0
t1=0 t2=4 max=4

3) *Массив А*:
a[0]=1
a[1]=2
a[2]=0
a[3]=5
a[4]=0
a[5]=3
a[6]=8
a[7]=0
a[8]=3
a[9]=1
t1=2 t2=7 max=8

4) *Массив А*:
a[0]=1
a[1]=2
a[2]=3
a[3]=4
В массиве нет нулей

5) *Массив А*:
a[0]=1
a[1]=0
a[2]=2
a[3]=3
a[4]=4
В массиве только один ноль или они располагаются друг за другом
t1=1 t2=1 max=0

6) *Массив А*:
a[0]=1
a[1]=2
a[2]=0
a[3]=0
a[4]=5
a[5]=2
В массиве только один ноль или они располагаются друг за другом

Содержание

Учебное электронное издание комбинированного распространения

Учебное издание

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ *C*. МАССИВЫ

Пособие
по выполнению контрольных и лабораторных работ по дисциплине «Вычислительная техника и программирование» для студентов технических специальностей дневной и заочной форм обучения

Электронный аналог печатного издания

Авторы-составители: **Кравченко** Ольга Алексеевна
Литвинов Дмитрий Александрович

Редактор *Н. В. Гладкова*
Компьютерная верстка *Н. Б. Козловская*

Подписано в печать 23.03.07.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Цифровая печать. Усл. печ. л. 2,32. Уч.-изд. л. 2,3.
Изд. № 38.
E-mail: ic@gstu.gomel.by
http://www.gstu.gomel.by

Издатель и полиграфическое исполнение:
Издательский центр учреждения образования
«Гомельский государственный технический университет
имени П. О. Сухого».
ЛИ № 02330/0131916 от 30.04.2004 г.
246746, г. Гомель, пр. Октября, 48.