1) Что такое определение функции?
2) Что такое вызов функции?
3) Что такое объявление функции?
4) Что такое формальные параметры?
5) Что такое локальные переменные?

Одномерные массивы

Лекция 7

Что такое массив?

Как ввести 10000 переменных?

Массив – это группа переменных одного типа, расположенных в памяти рядом (в соседних ячейках) и имеющих общее имя. Каждая ячейка в массиве имеет уникальный номер (индекс).

Массив – набор однотипных данных, которые характеризуются общим именем, типом и отличаются друг от друга только числовым индексом.

Надо:

- выделять память
- записывать данные в нужную ячейку
- читать данные из ячейки

Выделение памяти (объявление)

Массив = **таблица**!

```
int A[5];
double V[8];
bool L[10];
char S[80];
```

число элементов

Элементы всегда нумеруются с нуля!

A[0], A[1], A[2], A[3], A[4]

размер через константу

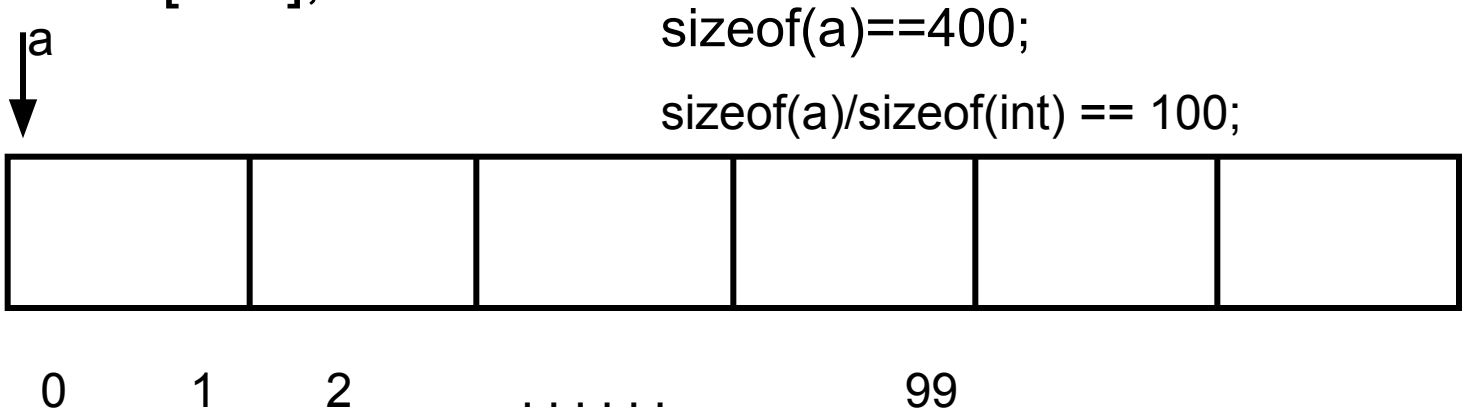
```
const int N=10;
int A[N];
```

Зачем?

Указатели и массивы

- При определении массива ему выделяется память. После этого имя массива воспринимается как константный указатель того типа, к которому относятся элементы массива.

int a[100];



Результатом операции & является адрес нулевого элемента массива:
a==&a==&a[0]

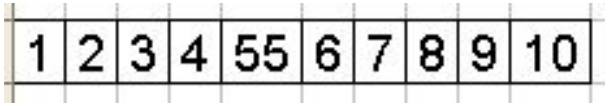
имя_массива[индекс] ==*(имя_массива+индекс)

Примеры

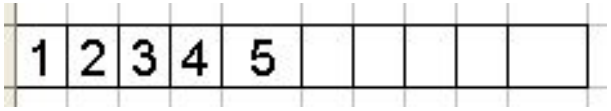
```
for(int i=0;i<n;i++)    //печать массива
cout<<*(a+i)<<" ";

a[1]!=*(a++)
a[1]==*(a+1)

int  a[100]={1,2,3,4,5,6,7,8,9,10};
int* na=a;
```

- Элементы массива можно задавать при его определении:

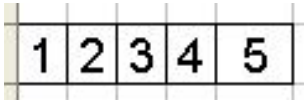
int a[10]={1,2,3,4,55,6,7,8,9,10};



int a[10]={1,2,3,4,5};

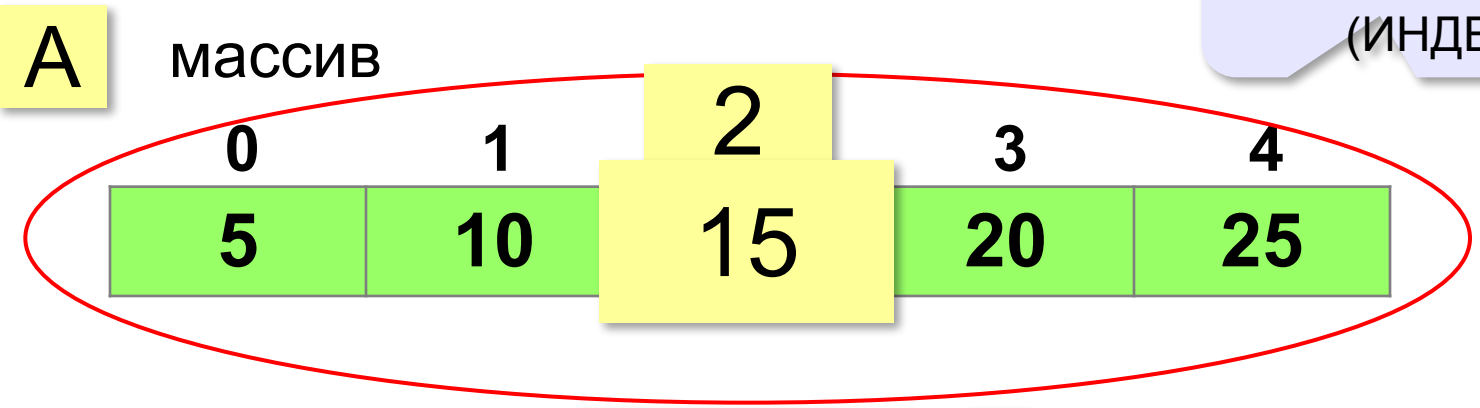


int a[]={1,2,3,4,5};



Обращение к элементу массива

НОМЕР
элемента массива
(ИНДЕКС)



`A[0]` `A[1]` `A[4]`

ЗНАЧЕНИЕ
элемента массива

НОМЕР (ИНДЕКС)
элемента массива: 2

`A[2]`

ЗНАЧЕНИЕ
элемента массива: 15

- Чтобы обратиться к элементу массива, надо указать имя массива и номер элемента в массиве (индекс):

a[0] – индекс задается как константа,

a[55] – индекс задается как константа,

a[i] – индекс задается как переменная,

a[2*i] – индекс задается как выражение.

Как обработать все элементы массива?

Объявление:

```
const int N = 5;
int A[N];
```

Обработка:

```
// обработать A[0]
// обработать A[1]
// обработать A[2]
// обработать A[3]
// обработать A[4]
```

1) если N велико (1000, 1000000)?

2) при изменении N программа не должна меняться!

Как обработать все элементы массива?

Обработка с переменной:

```
i = 0;
// обработать A[i]
i ++;
// обработать A[i]
i ++;
// обработать A[i]
i ++;
// обработать A[i]
i ++;
// обработать A[i]
```

```
i ++;
```

Обработка в цикле:

```
i = 0;
while ( i < N )
  {
  // обработать A[i]
  i ++;
  }
```

Цикл с переменной:

```
for( i = 0; i < N; i++ )
  {
  // обработать A[i]
  }
```

Заполнение массива

```
main()
{
   const int N=10;
   int A[N];
   int i;
   for( i=0; i<N; i++ )
     A[i]=i*i;
}
```

Чему равен `A[9]`?

Ввод с клавиатуры и вывод на экран

Объявление:

```
const int N=10;
int A[N];
```

Ввод с клавиатуры:

```
for ( i=0; i<N; i++ )
  {
   cout  << "A[" << i+1<<
 "]=";
  cin >> A[i];
  }
```

```
A[1] = 5
A[2] = 12
A[3] = 34
A[4] = 56
A[5] = 13
```

```
cout >> "Массив A:\n";
for ( i=0; i<N; i++ )
  cout << A[i] << " ";
```

? Зачем пробел?

Заполнение случайными числами

Задача. Заполнить массив (псевдо)случайными целыми числами в диапазоне от 20 до 100.

```
int irand ( int a, int b )
{
  return a + rand() % (b - a + 1);
}
```

```
for ( i = 0; i < N; i++ )
  {
  A[i] = irand ( 20, 100 );
  cout << A[i] << " ";
  }
```

Чтобы использовать функцию rand(), надо подключить библиотечный файл <stdlib> или <cstdlib> и <ctime> (для вызова в главной функции srand())

Перебор элементов

Общая схема:

```
for ( i = 0; i < N; i++ )
   {
   ... // сделать что-то с A[i]
   }
```

Подсчёт нужных элементов:

Задача. В массиве записаны данные о росте баскетболистов. Сколько из них имеет рост больше 180 см, но меньше 190 см?

```
count = 0;
for ( i = 0; i < N; i++ )
    if ( 180 < A[i] && A[i] < 190 )
       count ++;
```

Перебор элементов

Среднее арифметическое:

```
int count, sum;
count = 0;
sum = 0;
for ( i = 0; i < N; i++ )
  if ( 180 < A[i] && A[i] < 190 ) {
    count ++;
    sum += A[i];
    }
cout << (float)sum / count;
```

? Зачем `float`?

среднее арифметическое

Лекция 8

Что будет выведено на экран, если с клавиатуры вводятся подряд числа 1 2 3 4 5 6 7 8 9 10?

```
int a[100], n=10;
…….
for(int i=n-1;i>-1;i--)    cin>>a[i];
for(int i=0;i<n;i++)cout<<*(a+i)<<"  ";
```

- 10 9 8 7 6 5 4 3 2 1

Алгоритмы обработки массивов

Поиск в массиве

Найти в массиве элемент, равный X:

```
i = 0;
while ( A[i] != X )
   i ++;
cout << "A[" << i << "]=" << X;
```

? Что плохо?

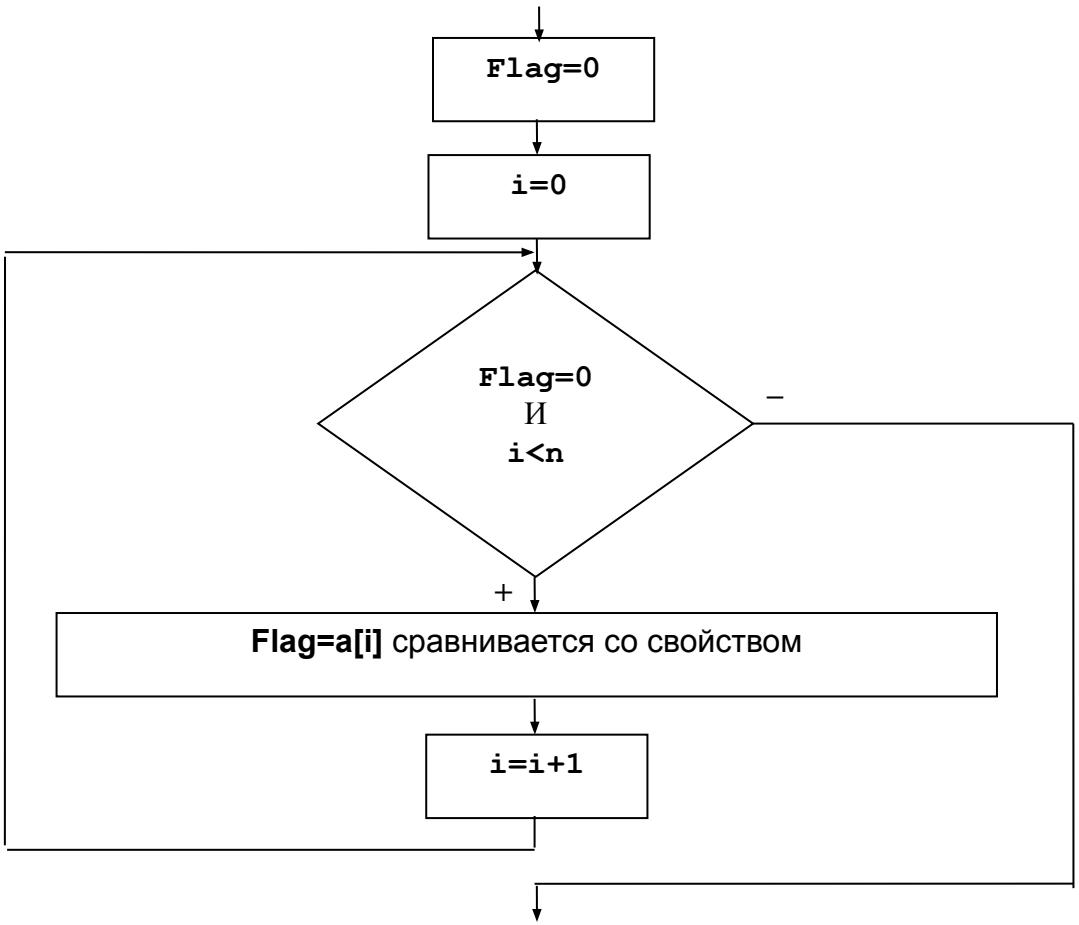
```
i = 0;
while ( i < N  &&  A[i] != X )
   i ++;
if ( i < N )
   cout << "A[" << i << "]=" << X;
else
   cout << "Не нашли!";
```

? Что если такого нет?

Поиск в массиве

Вариант с досрочным выходом:

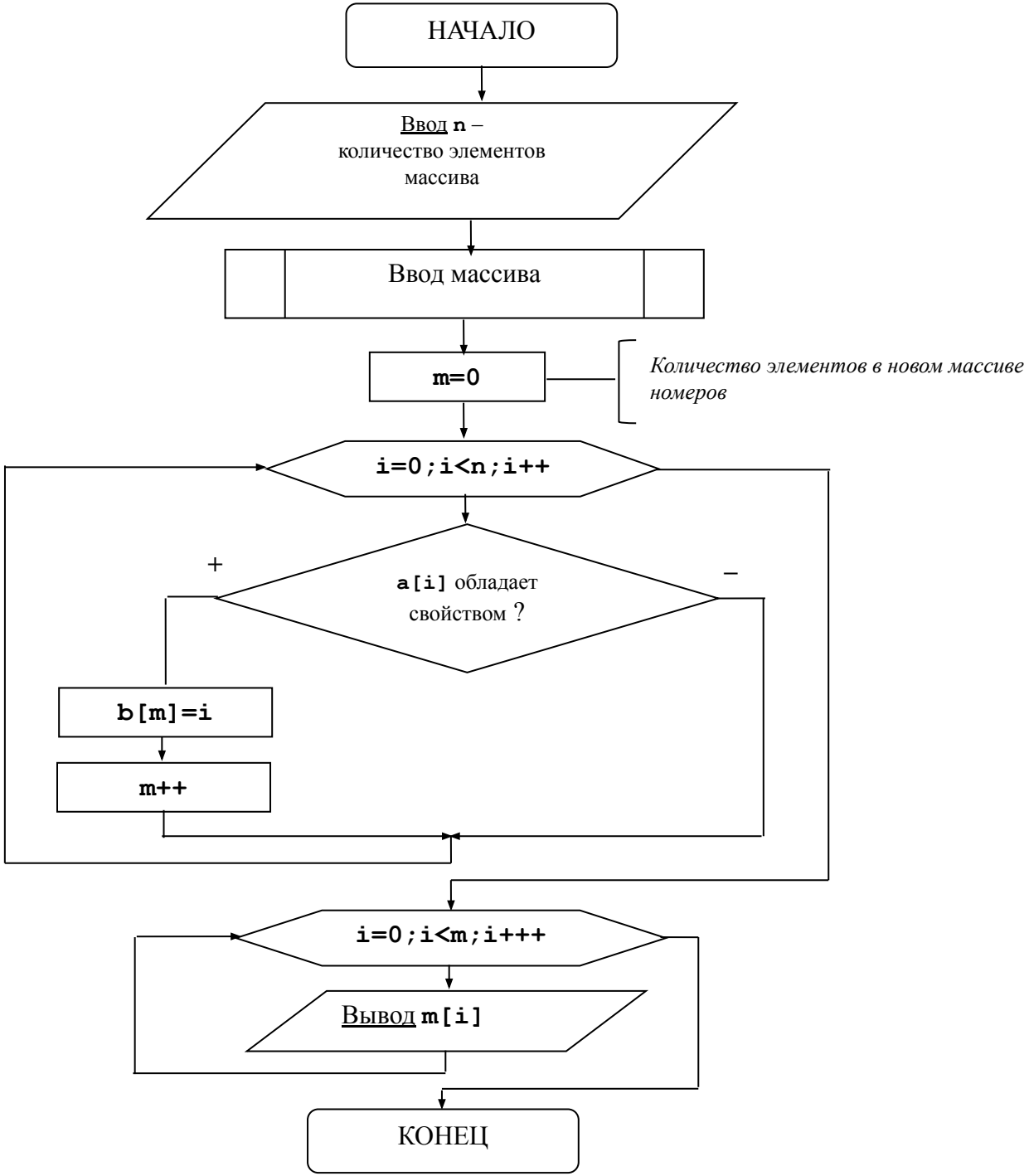
```
nX = -1;
for ( i = 0; i < N && nX==-1; i++ )
  if ( A[i] == X ) nX = i;

if ( nX >= 0 )
  cout << "A[" << nX << "]=" << X;
else
  cout << "Не нашли!";
```


Flag=0
i=0
Flag=0
И
i<n
–
+
Flag=a[i] сравнивается со свойством
i=i+1

Определить, имеется ли в массиве элемент, значение которого больше 1, но меньше 3?

Программа:

```
...
Flag=0;
i=0;
while (Flag==0 && i<n)
{Flag=a[i]>1&&a[i]<3;
i=i+1;
}
if (Flag) printf(“есть”);
    else printf(“нет”);
...
```


НАЧАЛО
Ввод n – количество элементов массива
Ввод массива
m=0
Количество элементов в новом массиве номеров
i=0;i<n;i++
+
a[i] обладает свойством ?
–
b[m]=i
m++
i=0;i<m;i+++
Вывод m[i]
КОНЕЦ

Программа:

Вывести номера всех элементов, значения которых больше значения первого элемента массива.

```
void main;
{ float a[100];
int b[100];
int n,m,l;
scanf(“%d”,&n);
for (i=0;i<n;i++)
    scanf(“%f”,&a[i]);
m=0;
for(i=0;i<n;i++)
    if (a[i]>a[0])
     {b[m]=i;m++;}
for (i=0;i<m;i++)
    printf(“%d  “,b[i]);
printf(“\n”);
}
```

Задачи

«А»: Заполните массив случайными числами в интервале [0,5]. Введите число X и найдите все значения, равные X.

Пример:

```
Массив:
1 2 3 1 2
Что ищем:
2
Нашли: A[2]=2, A[5]=2
```

Пример:

```
Массив:
1 2 3 1 2
Что ищем:
6
Ничего не нашли.
```

Задачи

«B»: Заполните массив случайными числами в интервале [0,5]. Определить, есть ли в нем элементы с одинаковыми значениями, стоящие рядом.

Пример:

```
Массив:
1 2 3 3 2 1
Есть: 3
```

Пример:

```
Массив:
1 2 3 4 2 1
Нет
```

Задачи

«С»: Заполните массив случайными числами. Определить, есть ли в нем элементы с одинаковыми значениями, не обязательно стоящие рядом.

Пример:

```
Массив:
3 2 1 3 2 5
Есть: 3, 2
```

Пример:

```
Массив:
3 2 1 4 0 5
Нет
```

Максимальный элемент

```
M=A[0];
for ( i=1; i<N; i++ )
  if ( A[i]>M )
    M=A[i];
cout << M;
```

Как найти его номер?

```
M=A[0]; nMax=0;
for ( i=1; i<N; i++ )
  if ( A[i]>M ) {
    M=A[i];
    nMax=i;
    }
cout << "A[" << nMax << "]=" << M;
```

Что можно улучшить?

Максимальный элемент и его номер

По номеру элемента можно найти значение!

```
nMax = 0;
for ( i = 1; i < N; i++ )
  if ( A[i] > A[nMax] )
    nMax = i;
cout << "A[" << nMax << "]=" << A[nMax];
```

Задачи

«A»: Заполнить массив случайными числами и найти минимальный и максимальный элементы массива и их номера.

Пример:

```
Массив:
1 2 3 4 5
Минимальный элемент: A[1]=1
Максимальный элемент: A[5]=5
```

«B»: Заполнить массив случайными числами и найти два максимальных элемента массива и их номера.

Пример:

```
Массив:
5 5 3 4 1
Максимальный элемент: A[1]=5
Второй максимум: A[2]=5
```

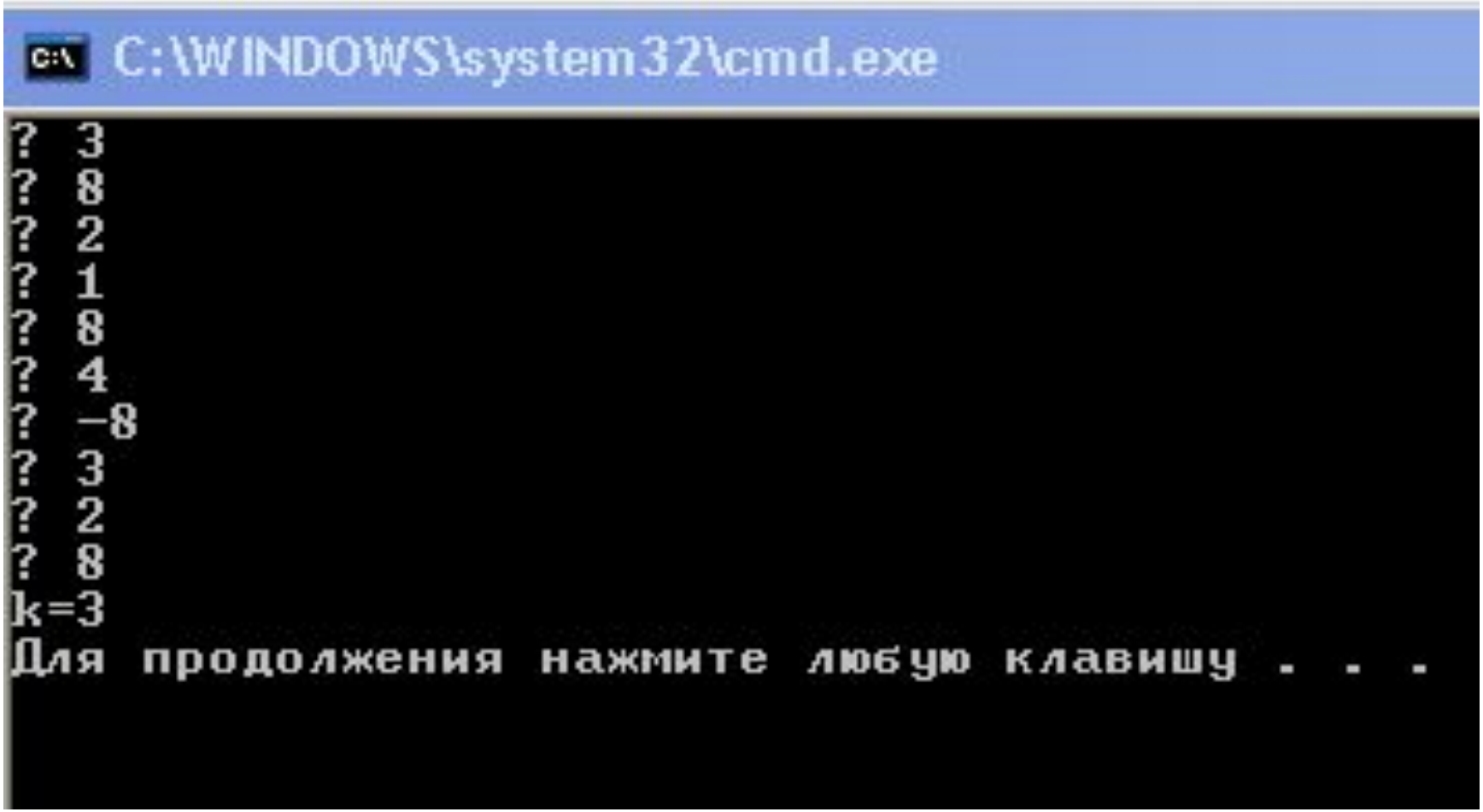
Задачи

«С»: Введите массив с клавиатуры и найдите (за один проход) количество элементов, имеющих максимальное значение.

Пример:

```
Массив:
3 4 5 5 3 4 5
Максимальное значение 5
Количество элементов 3
```

```
#include <iostream>
#include<cstdlib>
using namespace std;
void main()
{
int const N=10;
int a[N];
int i,max,k=0;
for(i=0; i<N; i++)
{
cout<<"? ";
cin>>a[i];
}
max=a[0]; k=1;
for(i=1; i<N; i++)
  if(a[i]==max) k++;
  else
        if(a[i]>max) {max=a[i]; k=1;}

cout<<"k="<<k<<endl;
}
```


C:\WINDOWS\system32\cmd.exe
? 3
? 8
? 2
? 1
? 8
? 4
? -8
? 3
? 2
? 8
k=3
Для продолжения нажмите любую клавишу . . .

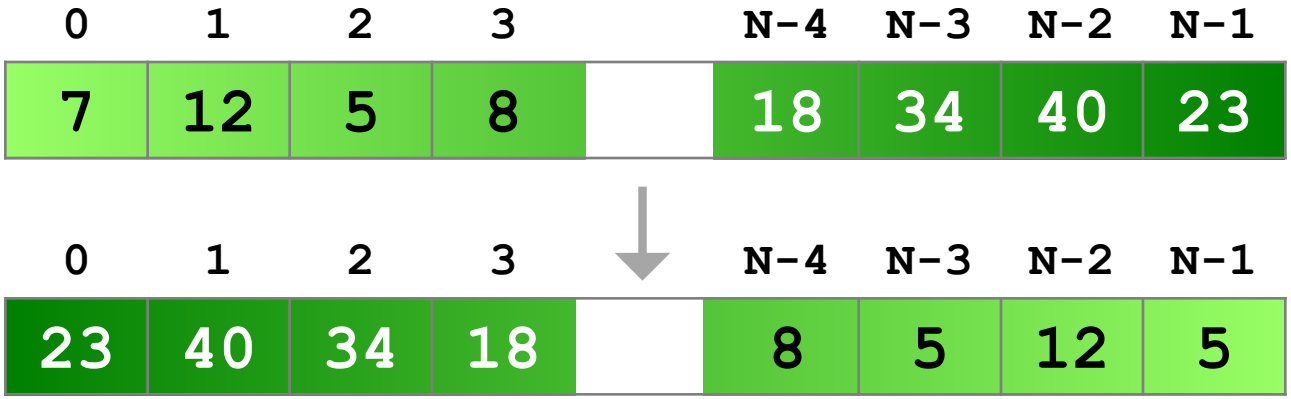
Поиск номера первого максимального/минимального элементов массива

```
M = A[0]; nM = 0;
for ( i = 1; i < N; i++ )
  if ( A[i] < M )
{
    M = A[i];
    nM = i;    // номер первого максимального
    nM = i; //Номер первого минимального
    }
cout << "A[" << nM << "]=" << M;
```

Поиск номера последнего максимального/минимального элементов массива

```
M = A[0]; nM = 0;
for ( i = 1; i < N; i++ )
  if ( A[i] >= M )
{
    M = A[i];
    nM = i;   // номер последнего максимального
    nM = i; //Номер последнего минимального
    }
cout << "A[" << nM << "]=" << M;
```

Реверс массива



«Простое» решение:

остановиться на середине!

```
for( i = 0; i < N/2 ; i++ )
   {
   // поменять местами A[i] и A[N+1-i]
   }
```

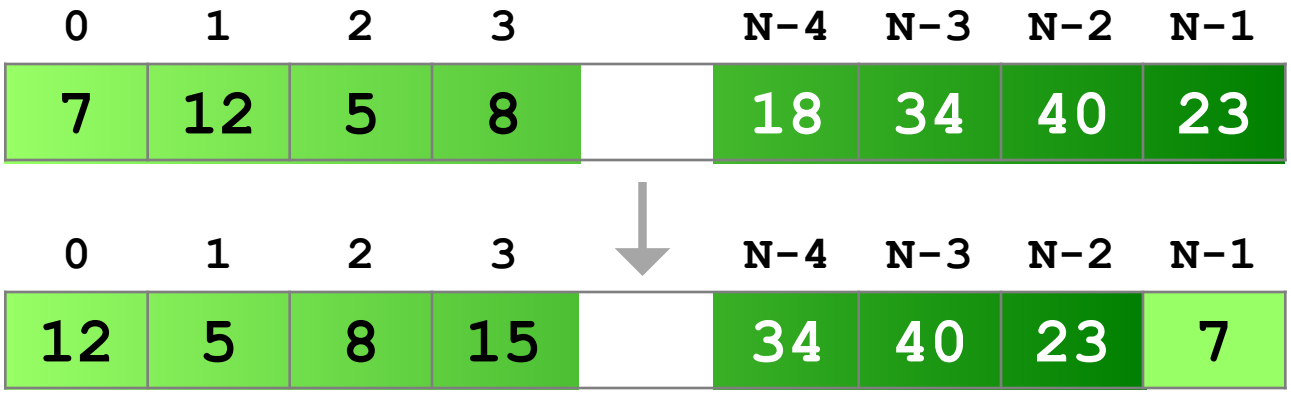
? Что плохо?

Реверс массива

```
for ( i=0; i<(N/2); i++ )
   {
   c=A[i];
   A[i]=A[N-1-i];
   A[N-1-i]=c;
   }
```

*Как обойтись без переменной c?

Циклический сдвиг элементов



«Простое» решение:

```
for ( i = 0; i < N-1; i++ )
  A[i] = A[i+1];
```

? Почему не до N-1?

? Что плохо?

Задачи

«A»: Заполнить массив случайными числами и выполнить циклический сдвиг элементов массива вправо на 1 элемент.

Пример:

```
Массив:
1 2 3 4 5 6
Результат:
6 1 2 3 4 5
```

«B»: Массив имеет четное число элементов. Заполнить массив случайными числами и выполнить реверс отдельно в первой половине и второй половине.

Пример:

```
Массив:
1 2 3 4 5 6
Результат:
3 2 1 6 5 4
```

Задачи

«С»: Заполнить массив случайными числами в интервале [-100,100] и переставить элементы так, чтобы все положительные элементы стояли в начала массива, а все отрицательные и нули – в конце. Вычислите количество положительных элементов.

Пример:

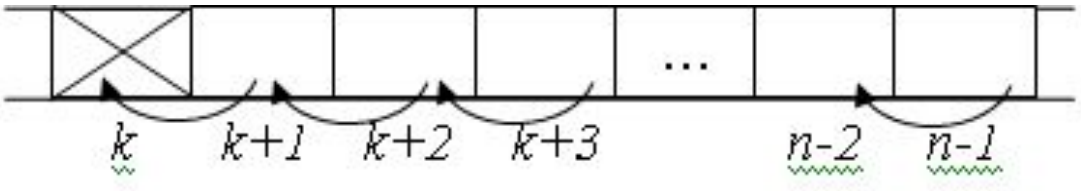
```
Массив:
20 -90 15 -34 10 0
Результат:
20 15 10 -90 -34 0
Количество положительных элементов: 3
```

Удаление элементов массива

- Удалить элемент с номером К
- Удалить все элементы с заданными свойствами

Задачи удаления элементов одномерного массива

- Удалить элемент с номером *k* из одномерного массива.



Вариант 1:

```
for (i=k+1;i<n;i++)
    a[i-1]=a[i];  // a[i] – указывает, что сдвигаем
n--;
```

Вариант 2:

```
for (i=k;i<n-1;i++)
    a[i]=a[i+1];  // a[i] – указывает, куда сдвигаем
n--;
```

- Удалить все элементы массива, обладающие заданным свойством.

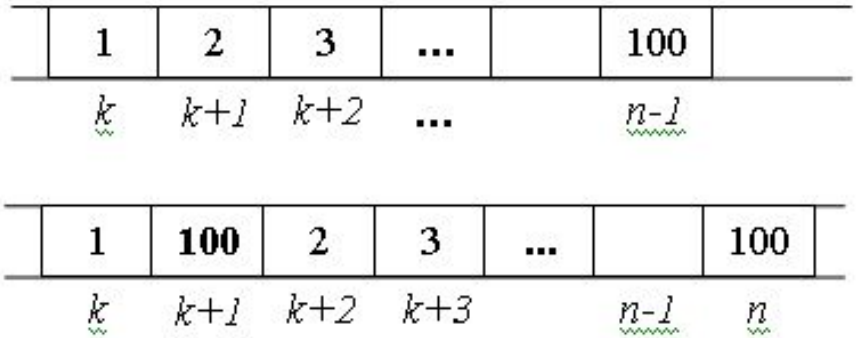
for (k=n-1;k>-1;k--)

if (a[k] *обладает свойством***)**

{(1) *или* **(2);}**

Задачи вставки элемента в одномерный массив

- После элемента с номером *k* вставить заданную величину; например, 100:



Вариант 1:

```
for(i=n-1;i>k;i--) // все элементы, начиная с k+1-го
    a[i+1]=a[i];       // сдвигаются на один вправо
                       //i – номер элемента, который сдвигается вправо
a[k+1]=B; // на k+1-е место вставляется новый элемент
n++; // размер массива увеличивается на единицу
```

Вариант 2:

В этой реализации параметр отвечает за место элемента, который сдвигается вправо:

```
for(i=n;i>k+1;i--)
    a[i]=a[i-1];       // a[i] указывает, куда сдвигаем
a[k+1]=B;
n++;
```

- Перед элементом с номером *k* вставить заданную величину.

Вариант 1:

```
for(i=n-1;i>=k;i--) // все элементы, начиная с k-го
    a[i+1]=a[i];      // сдвигаются на один вправо
a[k]=B; // на k-е место вставляется новый элемент
n++; // размер массива увеличивается на единицу
```

Вариант 2:

```
for(i=n;i>k;i--)
    a[i]=a[i-1];
a[k]=B;
n++; // размер массива увеличивается на единицу
```

- После элементов с заданными свойствами вставить указанную величину. Например, вставить число 100, после всех элементов, которые кратны 3.

```
for (k=n-1;k>-1;k--) // просмотр начинается с
                                  // конца
   if (a[k]%3==0)
   {for (i=n-1;i>k;i--) // сдвиг на один элемент
    a[i+1]=a[i];    // вправо
   a[k+1]=100; n++;
   }
```

За основу взята презентация

ПОЛЯКОВ Константин Юрьевич
д.т.н., учитель информатики
ГБОУ СОШ № 163, г. Санкт-Петербург
kpolyakov@mail.ru
ЕРЕМИН Евгений Александрович
к.ф.-м.н., доцент кафедры мультимедийной дидактики и ИТО ПГГПУ, г. Пермь
eremin@pspu.ac.ru

Источники иллюстраций

- old-moneta.ru
- www.random.org
- www.allruletka.ru
- www.lotterypros.com
- logos.cs.uic.edu
- ru.wikipedia.org
- иллюстрации художников издательства «Бином»

I. Описать синтаксис понятий языка C++:

1) определение функции

2) вызов функции

3) объявление функции

II. Что вычисляет данная функция?

```
int Fun(int a, int b)
{
  do {
    if (a>b) a= a%b; else b=b%a;
      } while (a!=0 && b!=0);
return a+b;
}
```

Сортировка

Лекция 9

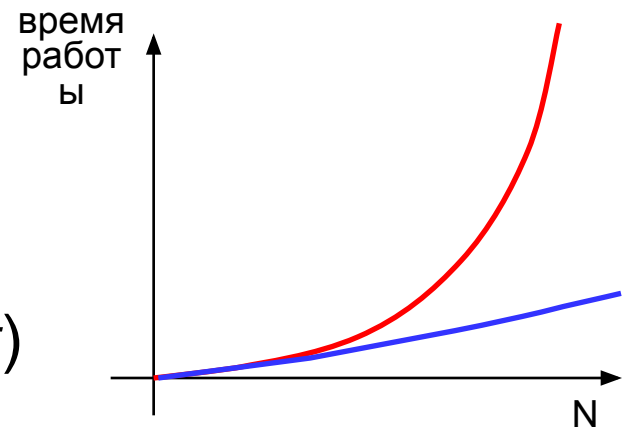
Что такое сортировка?

Сортировка – это расстановка элементов массива в заданном порядке.

…по возрастанию, убыванию, последней цифре, сумме делителей, по алфавиту, …

Алгоритмы:

- простые и понятные, но неэффективные для больших массивов
  - **метод пузырька**
  - **метод выбора**
- сложные, но эффективные
  - **«быстрая сортировка»** (*QuickSort*)
  - сортировка «кучей» (*HeapSort*)
  - сортировка слиянием (*MergeSort*)
  - пирамидальная сортировка



Идея сортировок

Массив разбивают на две части:

отсортированную и неотсортированную.

Количество элементов в отсортированной части растет, а в неотсортированной – уменьшается.

При формировании задачи сортировки одномерного массива может использоваться следующая терминология:

1. упорядочить массив по возрастанию: **a1<a2<…<an**;
2. упорядочить массив по убыванию: **a1>a2>…>an**;
3. отсортировать массив по неубыванию (такая формулировка возможна, если среди элементов есть одинаковые):

   a1<a2<a3=a4=a5<a6<a7=a8<…<an

4. отсортировать массив по невозрастанию (такая формулировка возможна, если среди элементов есть одинаковые):

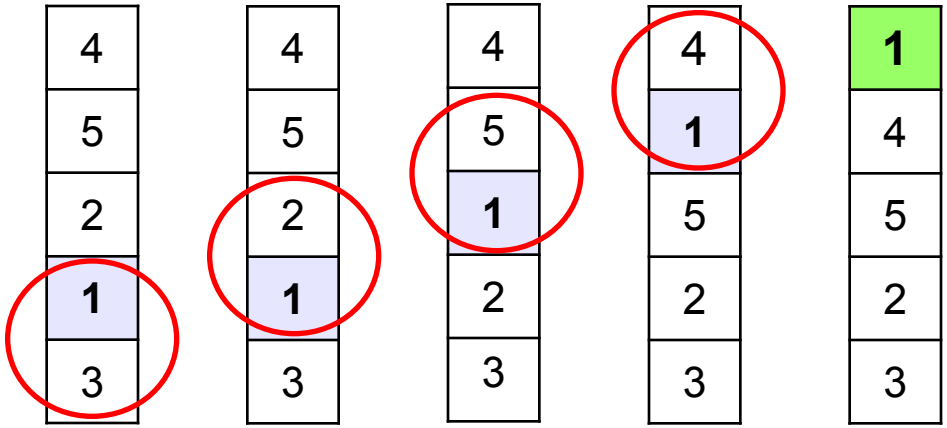
   a1>a2>a3=a4=a5>a6=a8>…>an

Метод пузырька (сортировка обменами)

Идея: пузырек воздуха в стакане воды поднимается со дна вверх.

Для массивов – **самый маленький** («легкий» элемент перемещается вверх (**«всплывает»**).

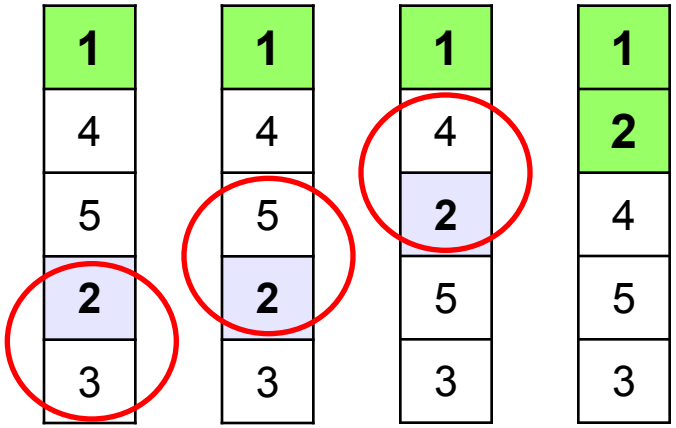
1-й проход:



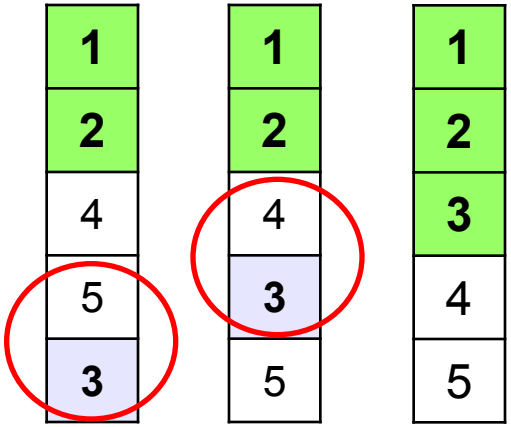
- сравниваем два соседних элемента; если они стоят «неправильно», меняем их местами
- за 1 проход по массиву один элемент (самый маленький) становится на свое место

Метод пузырька

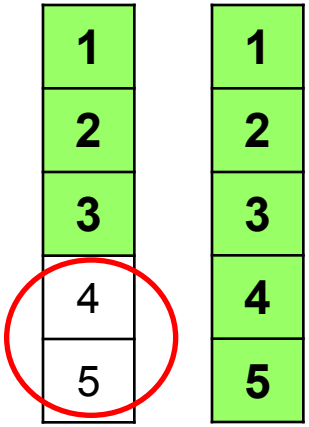
2-й проход:



3-й проход:



4-й проход:



Для сортировки массива из **N** элементов нужен **N-1** проход (достаточно поставить на свои места **N-1** элементов).

Метод пузырька

1-й проход:

```
сделать для j от N-2 до 0 шаг -1
  если A[j+1]<A[j] то
    // поменять местами A[j] и A[j+1]
```

единственное отличие!

2-й проход:

```
сделать для j от N-2 до 1 шаг -1
  если A[j+1]<A[j] то
    // поменять местами A[j] и A[j+1]
```

Метод пузырька

```
for ( i=0; i<N-1; i++ )
  for ( j=N-2; j>=i ; j-- )
    if ( A[j] > A[j+1] )
      {
      // поменять местами A[j] и A[j+1]
     R=A[j]; A[j]=A[j+1]; A[j+1]=R;
      }
```

Как написать метод «камня»?

Программа:

```
int a[100],b[100];
int n,i,k,R;
...
for (i=n;i>1;i--)
for (k=0;k<i-1;k++)
{if (a[k]>a[k+1])
    {R=a[k];a[k]=a[k+1];a[k+1]=R;}
    if (b[k]<b[k+1])
    {R=b[k];b[k]=b[k+1];b[k+1]=R;}
}
```

Пример:

i	k	Массив a	Массив b
4		2;-1;3;10	10;4;-3;2
	0	-1;2;3;10	10;4;-3;2
	1	-1;2;3;10	10;4;2;3
3	0		
	1		
2			

Фрагмент программы:

```
int a[100];
int n,i,k,R;
int F;
i=n; // длина неотсортированной части массива
do
{F=0; // массив является отсортированным
for (k=0;k<i-1;k++)
        if (a[k]>a[k+1])
            {R=a[k]; a[k]=a[k+1]; a[k+1]=R;
            F=1; // массив был неотсортированным }
    i--;}
while (F&&i>1);
```

Задачи

«A»: Напишите программу, в которой сортировка выполняется «методом камня» – самый «тяжёлый» элемент опускается в конец массива.

«B»: Напишите вариант метода пузырька, который заканчивает работу, если на очередном шаге внешнего цикла не было перестановок.

«C»: Напишите программу, которая сортирует массив по убыванию суммы цифр числа. Используйте функцию, которая определяет сумму цифр числа.

Метод выбора (минимального элемента)

Идея: найти минимальный элемент и поставить его на первое место.

```
сделать для i от 0 до N-2
  // найти номер nMin минимального
  // элемента из A[i]..A[N-1]
  если i != nMin то
    // поменять местами A[i] и A[nMin]
```

Метод выбора (минимального элемента)

```
for ( i = 0; i < N-1; i++ )
  {
  nMin = i;
  for ( j = i+1; j < N; j++ )
    if ( A[j] < A[nMin] )
      nMin = j;
  if ( i != nMin )
    {
    // поменять местами A[i] и A[nMin]
    }
  }
```

? Как поменять местами два значения?

Задачи

«А»: Массив содержит четное количество элементов. Напишите программу, которая сортирует первую половину массива по возрастанию, а вторую – по убыванию. Каждый элемент должен остаться в «своей» половине.

Пример:

```
Массив:
5 3 4 2 1 6 3 2
После сортировки:
2 3 4 5 6 3 2 1
```

Задачи

«B»: Напишите программу, которая сортирует массив и находит количество различных чисел в нем.

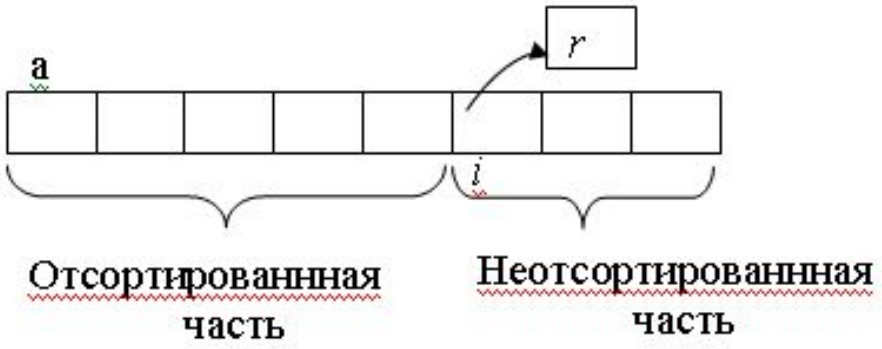
Пример:

```
Массив:
5 3 4 2 1 6 3 2 4
После сортировки:
1 2 2 3 3 4 4 5 6
Различных чисел: 5
```

«C»: Напишите программу, которая сравнивает число перестановок элементов при использовании сортировки «пузырьком» и методом выбора. Проверьте ее на разных массивах, содержащих 1000 случайных элементов, вычислите среднее число перестановок для каждого метода.

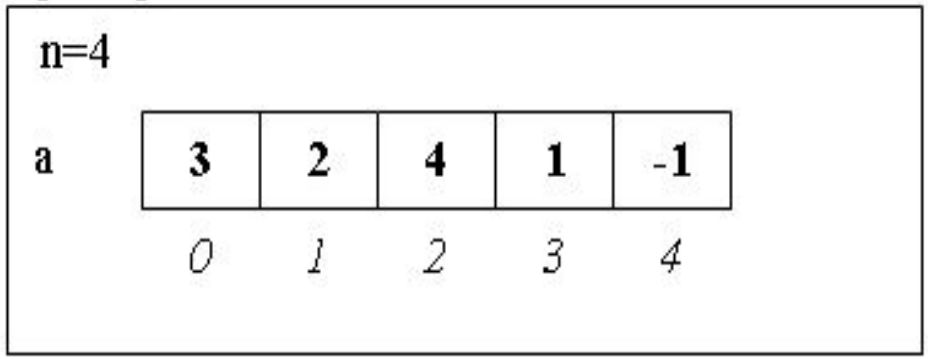
Метод простых вставок

Идея метода: левая часть массива является отсортированной, правая часть – неотсортированной. Для первого элемента неотсортированной части массива ищется место в отсортированной, и новый элемент вставляется в нужное место отсортированной части массива:



Фрагмент программы:

```
int a[100];
int j,i,n,r;
for (i=1;i<n;i++) // цикл по номерам в неотсортированной части
{  r=a[i];j=i-1;
while(j>=0&&r<=a[i])
    {a[j+1]=a[j];j--;}
    a[j+1]=r;
}
```



i	*r*	*j*	Массив *a*
1	2	0	2;3;4;1;-1
		-1	
2	4	1	2;3;4;1;-1
3	1	2	1;2;3;4;-1
		1	
		0	
4	-1	3	-1;1;2;3;4

Метод вставками с барьером

```
void sort_vs(int *a, int n)
{int i,j;
 for (i=2; i<n; i++)
 {
     a[0]=a[i];
     j=i-1;
     while(a[j]<a[0]) a[j+1]=a[j--];
     a[j+1]=a[0];

 }
}
```

Метод вставками с барьером

```
int main()
{
  int n, i;
  int b[100];
  Read(b,n);
   getchar();
  b[n]=b[0];
  n++;
  sort_vs(b,n);
  for(i=0;i<n-1; i++) b[i]=b[i+1];
  n--;
  Write(b,n);
  getchar();
}
```

Шейкер-сортировка

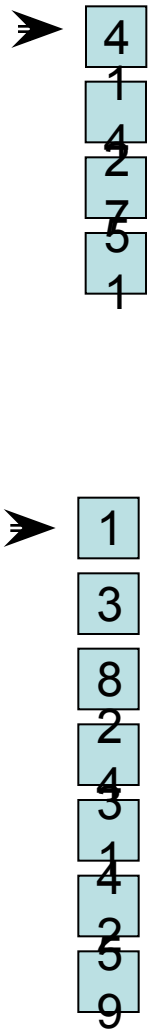
Это модификация метода «пузырька», которая учитывает два дополнительных требования:

1) устранение «лишних» просмотров массива, т.е. если массив уже отсортирован за первые проходы, последующие проходы не делаем. Пример: 12,3,5,7,9,10.
2) смена направлений прохода массива: сначала проходим от начала к концу, а затем – от конца к началу, потом снова от начала к концу и т.д. Это позволяет уменьшить число проходов по массиву. Пример: 5,7,9,10,12,3.

Шейкер-сортировка

```
void Shaker_Sort(int n, int *a)
{
    int  j,k,l,r;
    int  x;

     l=1; //левая граница
     r=n-1; //правая граница
    do
    {
// Обратный проход
          for( j=r; j>=l;j--)
               if (a[j-1]>a[j])
                {
                  x=a[j-1]; a[j-1]=a[j];
                  a[j]=x;
                  k=j;        /*
    фиксирование места
    последнего обмена   */
                }
          l=k+1; // левая граница
// Прямой проход
          for(j=l; j>=r; j++)
               if (a[j-1]>a[j])
               {
                  x=a[j-1]; a[j-1]=a[j];
                  a[j]=x;
                  k=j;  /* фиксирование
    места   последнего обмена */
               }
          r=k-1; // правая граница
}
 while (l<=r); /* До тех пор пока
    левая граница небольше
    правой*/
}
```

Алгоритм слияния упорядоченных массивов

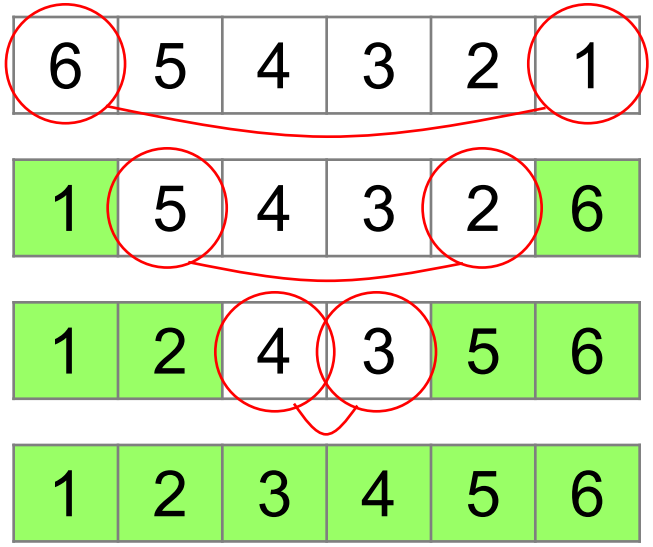



```
void merge(int a[],int b[], int na,int nb, int *c,int
&nc)
{
  int  ia,ib,ic;
    ia = 0;
    ib = 0;
    ic = 0;
    while (ia <= na && ib <= nb) do
    {
        if (a[ia]<b[ib])  c[ic] = a[ia++];
           else c[ic] = b[ib++];
        ic++;
    }
    while (ia <= na) do
     { c[ic] = a[ia];
        ia++; ic++;
     }
    while (ib <= nb) do
     {c[ic] = b[ib];
      ib++; ic++;
     }
    nc = ic;
}
```

Быстрая сортировка (*QuickSort*)

Ч.Э.Хоар

Идея: выгоднее переставлять элементы, который находятся дальше друг от друга.

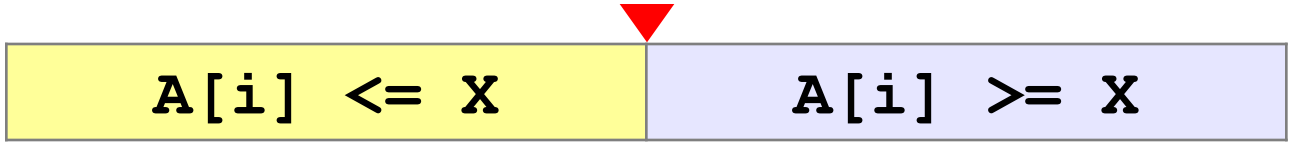


Для массива из **N** элементов нужно всего **N/2** обменов!

Быстрая сортировка

Шаг 1: выбрать некоторый элемент массива X

Шаг 2: переставить элементы так:



при сортировке элементы не покидают « свою область»!

Шаг 3: так же отсортировать две получившиеся области

Разделяй и властвуй (англ. *divide and conquer*)

Как лучше выбрать X?

Медиана – такое значение **X**, что слева и справа от него в отсортированном массиве стоит одинаковое число элементов (*для этого надо отсортировать массив…*).

Быстрая сортировка

Разделение:

1)выбрать средний элемент массива (**X=67**)

78	6	82	67	55	44	34

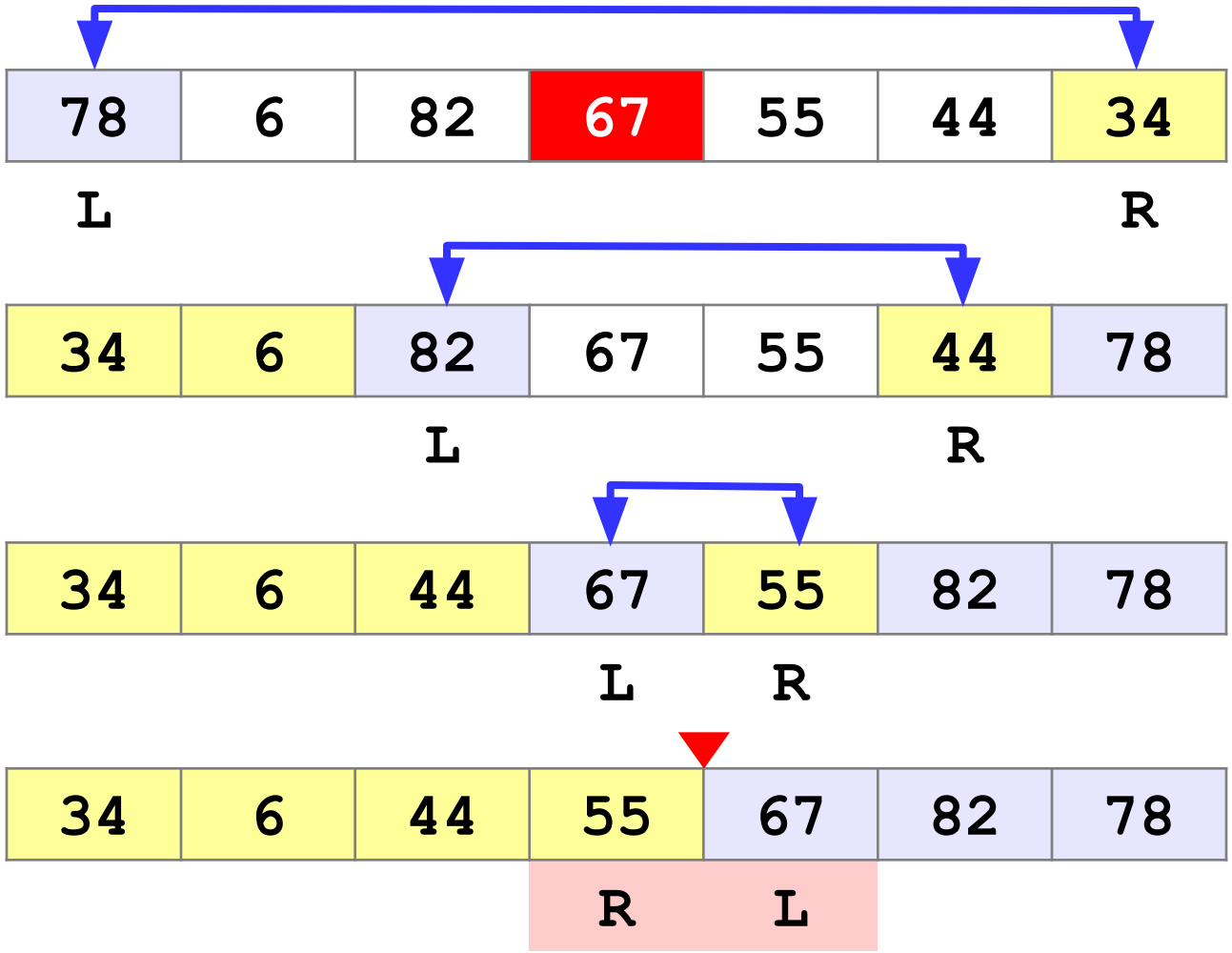
2)установить **L = 1, R = N**

3)увеличивая **L**, найти первый элемент **A[L]**, который **>= X** (должен стоять справа)

4)уменьшая **R**, найти первый элемент **A[R]**, который **<= X** (должен стоять слева)

5)если **L<=R** то поменять местами **A[L]** и **A[R]** и перейти к п. 3
иначе **стоп**.

Быстрая сортировка



L > R : разделение закончено!

Быстрая сортировка

Основная программа:

```
const int N=7;
int A[N];
...
main()
{
  // заполнить массив
  qSort( 0, N-1 ); // сортировка
  // вывести результат
}
```

глобальные данные

процедура сортировки

Быстрая сортировка

```
void qSort( int nStart, int nEnd )
{
  int L, R, c, X;
  L=nStart; R=nEnd;
  X=A[(L+R)/2]; // или X=A[irand(L,R)];
  do {          // разделение
    while ( A[L]<X ) L++;
    while ( A[R]>X ) R--;
    if ( L<=R ) {
      c=A[L]; A[L]=A[R]; A[R]=c;
      L++; R--;
      }
    } while ( L<=R );
  if (nStart<R) qSort ( nStart, R );
  if (L< nEnd) qSort ( L, nEnd );
}
```

Что плохо?

Быстрая сортировка

Передача массива через параметр:

```
void qSort( int A[], int nStart,
                     int nEnd )
{
  ...
 if (nSta A, R) qSort ( A, nStart, R );
 if (L< n A, ) qSort ( A, L, nEnd );
}
```

```
main()
{ // заполнить массив
  qSort( A, 0, N-1 ); // сортировка
  // вывести результат
}
```

Быстрая сортировка

Сортировка массива случайных значений:

N	метод пузырька	метод выбора	быстрая сортировка
1000	0,24 с	0,12 с	0,004 с
5000	5,3 с	2,9 с	0,024 с
15000	45 с	34 с	0,068 с

Задачи

«А»: Массив содержит четное количество элементов. Напишите программу, которая сортирует по возрастанию отдельно элементы первой и второй половин массива. Каждый элемент должен остаться в «своей» половине. Используйте алгоритм быстрой сортировки.

Пример:

```
Массив:
5 3 4 2 1 6 3 2
После сортировки:
2 3 4 5 6 3 2 1
```

Задачи

«B»: Напишите программу, которая сортирует массив и находит количество различных чисел в нем. Используйте алгоритм быстрой сортировки.

Пример:

```
Массив:
5 3 4 2 1 6 3 2 4
После сортировки:
1 2 2 3 3 4 4 5 6
Различных чисел: 5
```

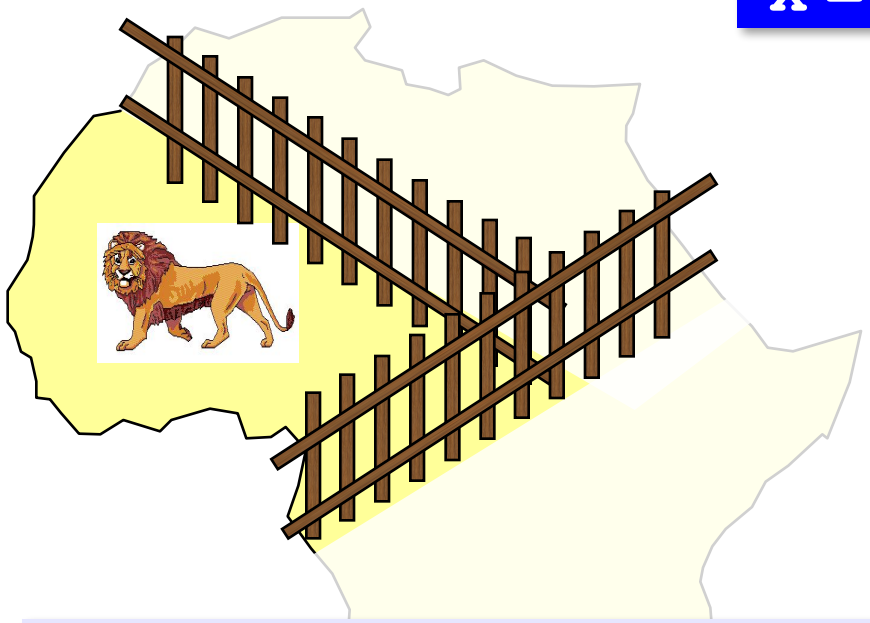
Задачи

«C»: Напишите программу, которая сравнивает число перестановок элементов при использовании сортировки «пузырьком», методом выбора и алгоритма быстрой сортировки. Проверьте ее на разных массивах, содержащих 1000 случайных элементов, вычислите среднее число перестановок для каждого метода.

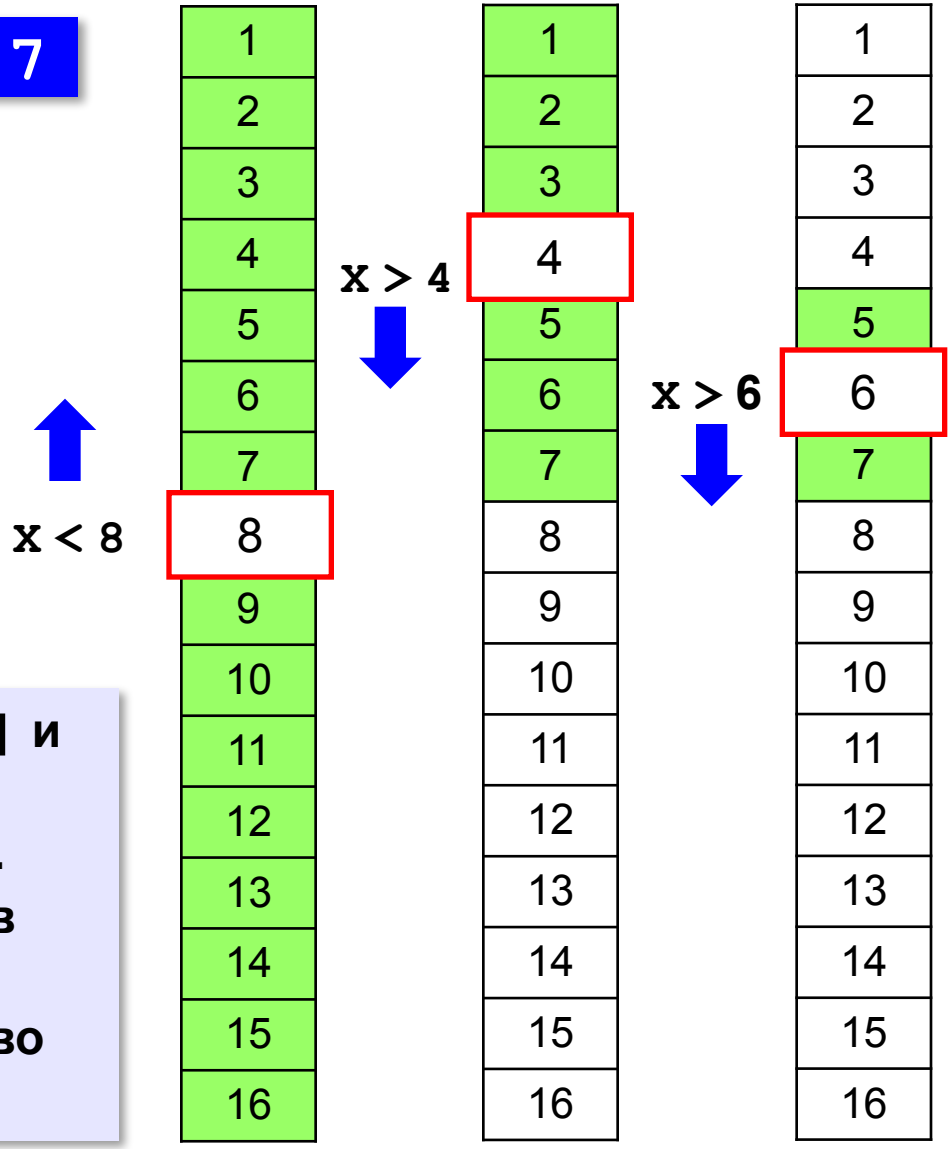
«D»: Попробуйте построить массив из 10 элементов, на котором алгоритм быстрой сортировки показывает худшую эффективность (наибольшее число перестановок). Сравните это количество перестановок с эффективностью метода пузырька (для того же массива).

Двоичный поиск

Двоичный поиск

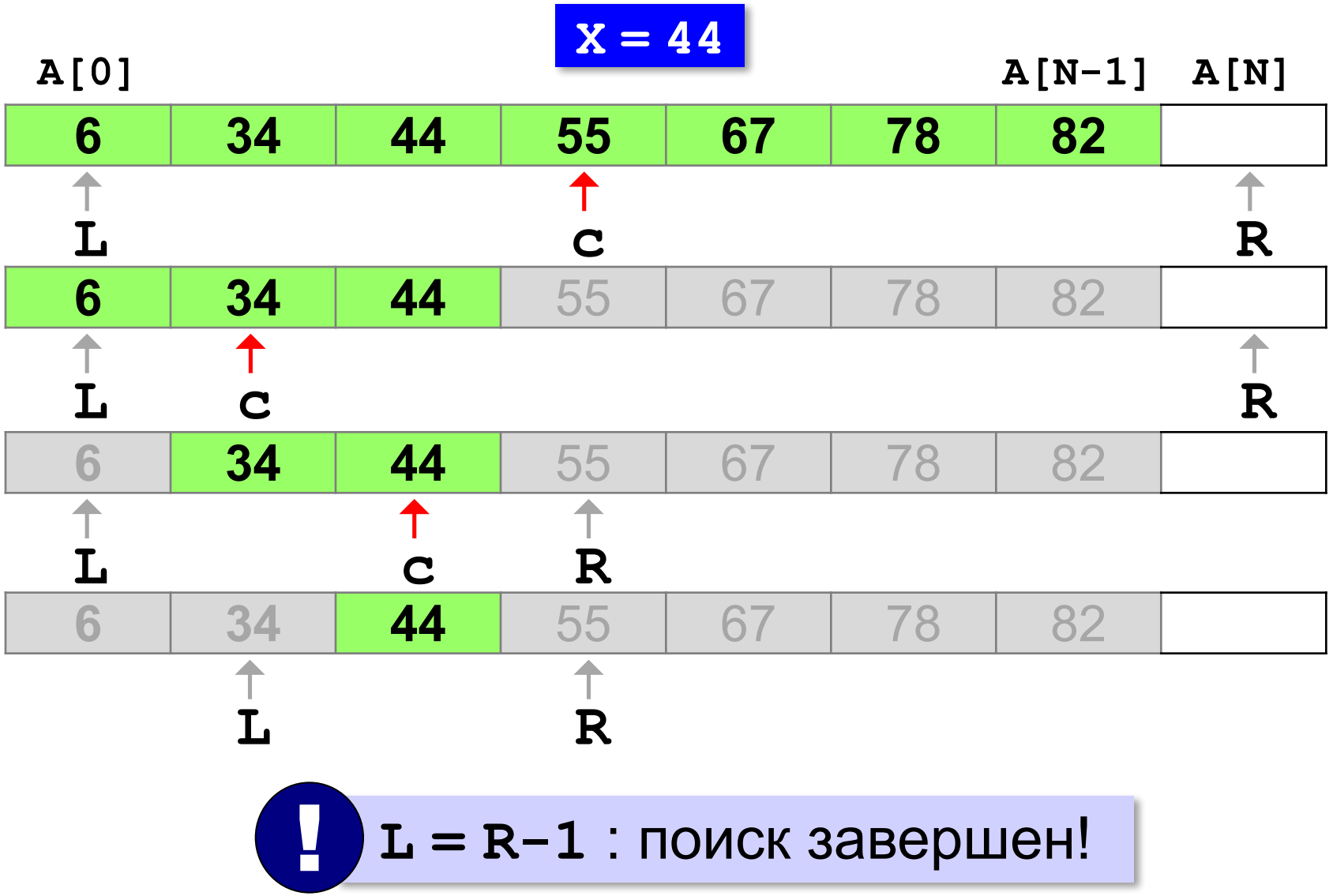


X = 7



1. Выбрать средний элемент `A[c]` и сравнить с X.
2. Если `X = A[c]`, то нашли (стоп).
3. Если `X < A[c]`, искать дальше в первой половине.
4. Если `X > A[c]`, искать дальше во второй половине.

Двоичный поиск



L = R-1 : поиск завершен!

Двоичный поиск

```
int X, L, R, c;
L=0; R=N;          // начальный отрезок
while ( L<R-1 )
   {
   c=(L+R)/2;      // нашли середину
   if ( X<A[c] ) // сжатие отрезка
         R=c;
   else L=c;
   }
if ( A[L]==X )
      printf ( "A[%d]=%d", L, X );
else printf ( "Не нашли!" );
```

Двоичный поиск

Число сравнений:

N	линейный поиск	двоичный поиск
2	2	2
16	16	5
1024	1024	11
1048576	1048576	21

- скорость выше, чем при линейном поиске

- нужна предварительная сортировка

Когда нужно применять?

Задачи

«А»: Заполнить массив случайными числами и отсортировать его. Ввести число X. Используя двоичный поиск, определить, есть ли в массиве число, равное X. Подсчитать количество сравнений.

Пример:

```
Массив:
1 4 7 3 9 2 4 5 2
После сортировки:
1 2 2 3 4 4 5 7 9
Введите число X:
2
Число 2 найдено.
Количество сравнений: 2
```

Задачи

«B»: Заполнить массив случайными числами и отсортировать его. Ввести число X. Используя двоичный поиск, определить, сколько чисел, равных X, находится в массиве.

Пример:

```
Массив:
1 4 7 3 9 2 4 5 2
После сортировки:
1 2 2 3 4 4 5 7 9
Введите число X:
4
Число 4 встречается 2 раз(а).
```

Пример:

```
Массив:
1 4 7 3 9 2 4 5 2
После сортировки:
1 2 2 3 4 4 5 7 9
Введите число X:
14
Число 14 не встречается.
```

Задачи

«С»: Заполнить массив случайными числами и ввести число и отсортировать его. Ввести число X. Используя двоичный поиск, определить, есть ли в массиве число, равное X. Если такого числа нет, вывести число, ближайшее к X.

Пример:

```
Массив:
1 4 7 3 9 2 4 5 2
После сортировки:
1 2 2 3 4 4 5 12 19
Введите число X:
12
Число 12 найдено.
```

Пример:

```
Массив:
1 4 7 3 9 2 4 5 2
После сортировки:
1 2 2 3 4 4 5 12 19
Введите число X:
11
Число 11 не найдено. Ближайшее число 12.
```

Динамическая память

- Динамическая память – это память, выделяемая программе для ее работы за вычетом сегмента данных, стека, в котором размещаются локальные переменные подпрограмм и собственно тела программы.
- Для работы с динамической памятью используют указатели.

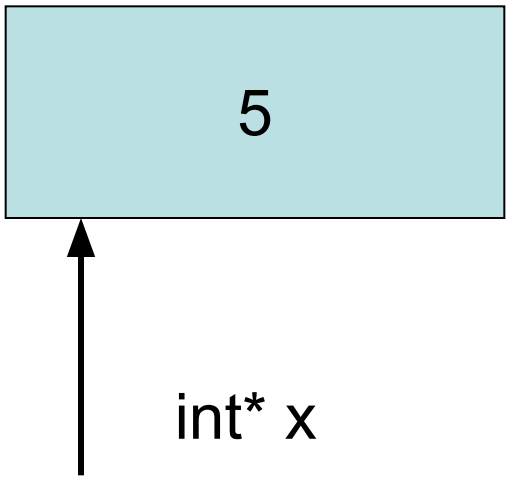
Создание динамических переменных

указатель = **new** имя_типа [инициализатор];

где инициализатор – выражение в круглых скобках.

Пример

```
int* x=new int(5);
```



Удаление динамических переменных

delete указатель;

Пример

delete x;

Динамические массивы

- Операция new при использовании с массивами имеет следующий формат:

new тип_массива

```
int* a = new int[100];
double* b = new double[10];
```

- Операция delete освобождает память, выделенную под массив

```
delete[] a;
delete[] b;
```