Массивы

Часто в задачах с группой переменных одного типа необходимо выполнить одни и те же действия.

В этом случае удобно использовать составную структуру данных – массив.

Массив – совокупность определенного числа пронумерованных однотипных данных, называемых **элементами массива**, имеющих общее имя.

Все элементы массива индексируются последовательно, начиная с нуля.

Поэтому массивы данных удобно обрабатывать в цикле с параметром.

Размещение элементов массива в памяти выполняется последовательно в смежных ячейках.

Массивы

тип элементов массива

Формат описания одномерного массива:

тип идентификатор[константное_выражение];

имя массива

размерность массива – число элементов

Имя массива хранит **адрес первого элемента** массива.

Количество элементов в массиве определяет размер массива и является **константным выражением**.

Обращение к элементам массива производится по имени массива и индексу элемента.

Массивы

Примеры:

Пусть описан одномерный массив:

int a[10];

объявлен массив из 10 элементов целого типа:

a[0], ***a***[1], ***a***[2], ***a***[3], ***a***[4], ***a***[5],***a***[6], ***a***[7], ***a***[8], ***a***[9]

индекс последнего элемента на 1 меньше размерности массива

float array[15];

массив вещественных переменных

array[0], ***array***[1], ***array***[2], … ***array***[14]

Задание значений элементов

При объявлении массива под каждый элемент массива в памяти будет выделено необходимое количество ячеек – **sizeof(тип)**, содержимое которых – мусор.

Способы задания значений элементов:

1. Используя операторы присваивания, можно каждому элементу присвоить свое значение:

Пусть например: ***int*** a[4];

a[0]=2;

a[1]=4;

a[2]=6;

a[3]=8;

Массивы

2. Объявление массива с одновременной инициализацией значений элементов

тип имя_массива[размерность]={*знач0, знач1, ..., значN-1*};

Примеры:

int mass[10]={2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

mass[0] mass[1] mass[9]

float f[8]={2.5, 3.5, 6.3, 8.1};

это равнозначно заданию массива

float f[8]={2.5, 3.5, 6.3, 8.1, 0, 0, 0, 0};

т.е. недостающие элементы обнуляются

Массивы

Возможно объявление массива без указания

```
#include<iostream. >
#include <stdlib.h>
using namespace std;
int main ()
{ int n=10, A[] = {3, 5, 1, 6, 2, 4, 8, 3, 7, 2};
  for (int i=0; i<n; i++)
      cout<<A[i]<<"  ";
  cout<<endl;
 system("pause");
 }
```

```
D:\ковчег\Алутина\Программирование\К занятиям\Задания к занятиям
3  5  1  6  2  4  8  3  7  2
Для продолжения нажмите любую клавишу . . .
```

Задание значений элементов

```
#include<iostream.  >
#include <stdlib.h>
using namespace std
int main ()
{ int n=10, A[10] = {3, 5, 1, 6, 2, 4};
  for (int i=0; i<n; i++)
      cout<<A[i]<<"   ";
  cout<<endl;
 system("pause");
 }
```

заданы значения
только 6-ти элементов

```
D:\ковчег\Алутина\Программирование\К занятиям\Зада
3  5  1  6  2  4  0  0  0  0
Для продолжения нажмите любую клавишу . .
```

Задание значений элем

3. Можно задавать значения эле
с клавиатуры:

Пример. Ввод с клавиатуры
массива:

```
#include<iostream>
#include <stdlib.h>
using namespace std;
int main ()
{ const int n=10;    int  A[n];
 cout<<"VVodite znacheniya elementov
 for (int i=0; i<n; i++)
    {cout<<"A["<<i<<"]=";
     cin>>A[i];
     cout<<endl;       }
 for (int i=0; i<n; i++)
   cout<<A[i]<<"   ";
 cout<<endl;
system("pause");
}
```

```
#include<iostream.h >
#include <stdlib.h>
using namespace std;
int main ()
{ const int n=10;
  int  A[10];
  cout<<"VVodite znacheniya elementov\n ";
  for (int i=0; i<n; i++)
     {cout<<"A["<<i<<"]=";
      cin>>A[i];
      cout<<endl;
     }
  for (int i=0; i<n; i++)
     cout<<A[i]<<"    ";
  cout<<endl;
 system("pause");
}
```

```
D:\ковчег\Алутина\Программирование\К занятиям\Задания
VVodite znacheniya elementov
 A[0]=5
A[1]=-2
A[2]=4
A[3]=-3
A[4]=7
A[5]=4
A[6]=11
A[7]=14
A[8]=32
A[9]=0
5    -2    4    -3    7    4    11    14    32    0
Для продолжения нажмите любую клавишу . . .
```

Массивы

```
#include <stdlib.h>
#include <iostream>
using namespace std;
const int n=5;
int main()
{   int k, array[n];
    for (k=0; k<n; k++)
      {
         cout<<"\nVvedite "<<k<<"  element ";
         cin>>array[k];
      }
    system ("Pause");
}
```

```
D:\ковчег\C++\Безымянный7.exe

Vvedite 0  element 3

Vvedite 1  element 1

Vvedite 2  element -2

Vvedite 3  element 7

Vvedite 4  element -5
Для продолжения нажмите любую клавишу . . .
```

Массивы

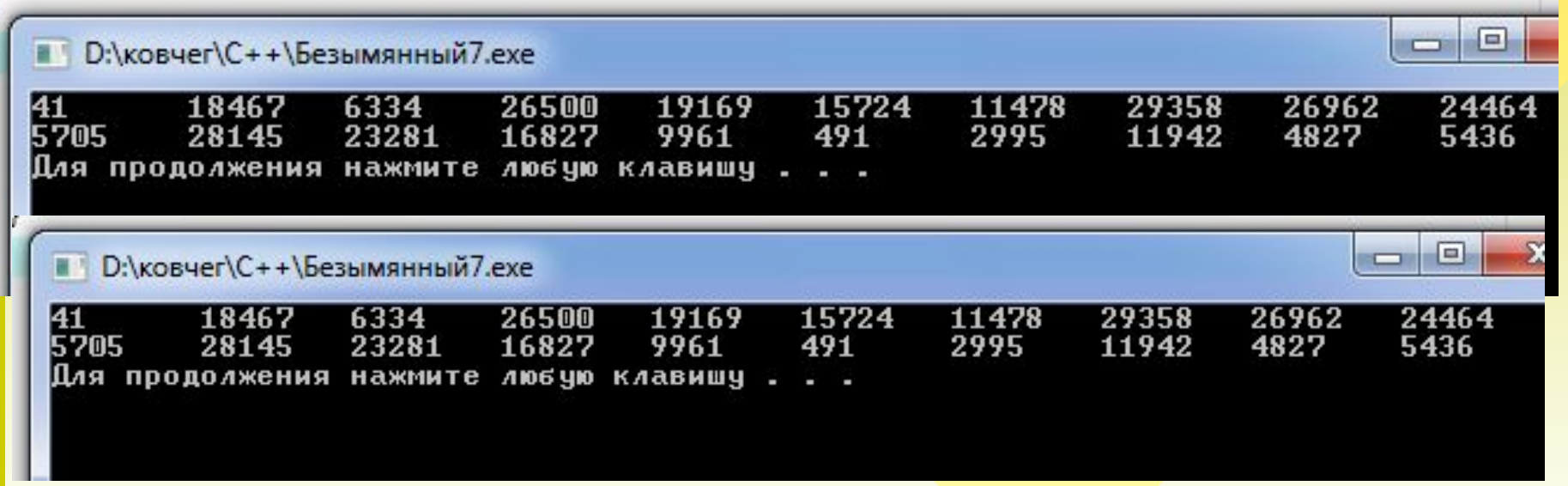
Значения элементов массива можно задавать с помощью функции, вырабатывающей «случайные числа».

Для получения случайных чисел служит функция ***rand***( ), которая возвращает случайное число из диапазона от 0 до значения константы *RAND_MAX* (как правило, эта константа равна 32767, но оно может быть и больше, в зависимости от компилятора 2 147 483 647).

Функция ***rand( )*** (как и константа *RAND_MAX*) описана в файле ***stdlib.h***:

Массивы

```
#include <stdlib.h>
#include <iostream>
using namespace std;
int main()
{   int k, array[20];
    for (k=0; k<20; k++)
      {
         array[k]=rand( );
         cout<<array[k]<<"\t";
      }
    system ("Pause");
}
```



Задание значений элементов

4. Можно задавать значения элементов, используя датчик случайных чисел:

Пример.

```
#include<iostream>
#include <stdlib.h>
using namespace std;
int main ()
{ const int n=10;
  int  A[n];
 cout<<"P
 for (int i=0; i<n; i
   { A[i]=rand()%100;
     cout<<A[i]<<"   ";
   }
 cout<<endl;
 system("pause");
}
```

```
#include<iostream.h>
#include <stdlib.h>
using namespace std;
int main ()
{ const int n=10;
  int  A[10];
  cout<<"Poluchennie elementi massiva \n ";
  for (int i=0; i<n; i++)
     { A[i]=rand()%100;
       cout<<A[i]<<"    ";
 system("pause");
```

Используя операцию % (остаток от деления) получим число из диапазона 0 .. 99

```
D:\ковчег\Алутина\Программирование\К занятиям\Задания к заня
Poluchennie elementi massiva
 41   67   34   0   69   24   78   58   62   64
Для продолжения нажмите любую клавишу . . .
```

Получили одинаковые последова-тельности «псевдослучай ных» чисел

```
Poluchennie elementi massiva
 41   67   34   0   69   24   78   58   62   64
Для продолжения нажмите любую клавишу . . .
```

Массивы

В примере функция *rand()* постоянно возвращает одну и ту же **последовательность псевдослучайных чисел**.

В реальных программах желательно получать разные последовательности случайных чисел. Необходимо использовать функцию ***srand()***, которая инициализирует последовательность случайных чисел для функции *rand()*. Функцию ***srand()*** достаточно вызвать только один раз в начале программы, для ее работы необходимо подключить заголовочный файл библиотеки ***time.h***:

```
#include <time.h>
int main()
{  srand((unsigned)time(NULL));
   …
}
```

Массивы

```
#include <stdlib.h>
#include <iostream>
#include <time.h>
using namespace std;
int main()
{   srand((unsigned)time(NULL));
    int k, array[20];
    for (k=0; k<20; k++)
      {
         array[k]=rand( );
         cout<<array[k]<<"\t";
      }
    system ("Pause");
}
```

```
D:\ковчег\C++\Безымянный7.exe
18843   20051   30353   29978   1121    9174    21455   30282   8023    15270
1410    23554   19192   15016   14863   999     29671   2624    26454   13864
Для продолжения нажмите любую клавишу . . .
```

```
D:\ковчег\C++\Безымянный7.exe
19036   31615   2995    7922    19857   11475   8482    21412   31935   12982
25452   26152   14666   8209    13481   8138    6301    8368    3394    25563
Для продолжения нажмите любую клавишу . . .
```

Задание значений элементов

Необходимо инициализировать счетчик случайных чисел.

Пример.

```
#include<iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
int main ()
{ const int n=10;
  srand((unsigned) time(N
  int  A[n];
  cout<<"Poluchennie elem
  for (int i=0; i<n; i++)
     { A[i]=rand()%100;
       cout<<A[i]<<"   ";
     }
  cout<<endl;
 system("pause");
}
```

```
#include<iostream.h>
#include <stdlib.h>
#include <time.h>
using namespace std;
int main ()
{ const int n=10;
  srand((unsigned) time(NULL));
  int  A[10];
  cout<<"Poluchennie elementi massiva \n ";
  for (int i=0; i<n; i++)
     { A[i]=rand()%100;
       cout<<A[i]<<"   ";
     }
  cout<<endl;
 system("pause");
}
```

D:\ковчег\Алутина\Программирование\К занятиям\Задания к занятия

```
Poluchennie elementi massiva
 45   84   34   42   65   25   80   59   44   22
Для продолжения нажмите любую клавишу . . .
```

D:\ковчег\Алутина\Программирование\К занятиям\Задания к занятия

```
Poluchennie elementi massiva
 1   67   13   2   5   43   56   31   97   77
Для продолжения нажмите любую клавишу . . .
```

Основные типы задач при работе с массивами

1. Поиск элементов и/или их индексов, удовлетворяющих некоторому условию (например: максимума или минимума).
2. Нахождение суммы, произведения, среднего арифметического и т.п. элементов массива или его части.
3. Замена элементов удовлетворяющих заданному условию.
4. Упорядочивание элементов массива (сортировка).
5. Подсчет количества элементов, удовлетворяющих заданному условию.
6. Одновременная обработка нескольких массивов.

Основные этапы решения задач с массивами

1. Объявление массива.
2. Выбрать способ задания элементов массива и задать значения.
3. Обработка элементов массива по условию задачи.
4. Вывод результатов или вывод обновленного массива.

В некоторых случаях можно выполнять 1 этап одновременно со 2-м (инициализировать при объявлении). Объединить 2-й и 3-й этапы в одном цикле или 3-й с 4-м (при этом необходимо помнить о целесообразности такого объединения).

Пример: Найти количество отрицательных элементов

```
#include <stdlib.h>
#include <iostream>
using namespace std;
const int n=5;
int main()
{   int i, k=0,   array[n];
    for (i=0; i<n; i++)
      { cout<<"\nVvedite "<<i<<"  element ";
        cin>>array[i];
      }
    for (i=0; i<n; i++)
      if ( array[i]<0) k++;
    cout<<"\n Negative elementov "<<k<<endl;
    system ("Pause");
}
```

```
D:\ковчег\C++\Безымянный7.exe

Vvedite 0  element -2
Vvedite 1  element -1
Vvedite 2  element 0
Vvedite 3  element -5
Vvedite 4  element 9

 Negative elementov 3
Для продолжения нажмите любую клавишу . . .
```

Обработка массивов

```
#include<iostream.h>
#include <stdlib.h>
using namespace std;
 int main ()
{   int    A[] = {3, 5, 1, 6, 2, 4, 8, 3, 7, 2};
   cout<<"elementi massiva"<<endl;
   for (int i=0; i<size        //sizeof(int); i++)
      {
        cout<<A[i]<<"    ";
      }
   cout<<endl;
 system("pause");
}
```

Вычисление размера массива

```
D:\ковчег\Алутина\Программирование\К занятиям\Задания к за
elementi massiva
 3    5    1    6    2    4    8    3    7    2
Для продолжения нажмите любую клавишу . . .
```

Обработка массивов

```
#include<iostream.h>
#include <stdlib.h>
using namespace std;
 int main ()
{  int   A[] = {3, 5, 1, 6, 2, 4, 8, 3, 7, 2};
  cout<<"elementi massiva \n ";
  for (int i=0; i<sizeof(A)/sizeof(int); i++)
     {
       cout<<A[i]<<"   ";
     }
  cout<<i<<endl;
 system("pause");
}
```

Компилятор | Ресурсы | Журнал компиляции | Отладка | Результаты поиска | Закрыть

Строка	Файл	Сообщение
1	D:\ковчег\Dev-Cpp\include\c++\...	...am.h:31, fr...
1	D:\ковчег\Алутина\Программиров...	from D:\ко... программирование\К занятиям\Задания к занятиям\...
32:2	D:\ковчег\Dev-Cpp\include\c++\3.4...	#warning This file in... least one deprecated or antiquated header. Please consider using o...
	D:\ковчег\Алутина\Программиров...	In function `int ma...():
11	D:\ковчег\Алутина\Программиров...	name lookup of `i' changed for new ISO `for' scoping
7	D:\ковчег\Алутина\Программиров...	using obsolete binding at `i'

Параметр цикла i не доступен

Размещение массива в памяти

Под хранение массива в памяти компилятором отводятся смежные ячейки памяти.

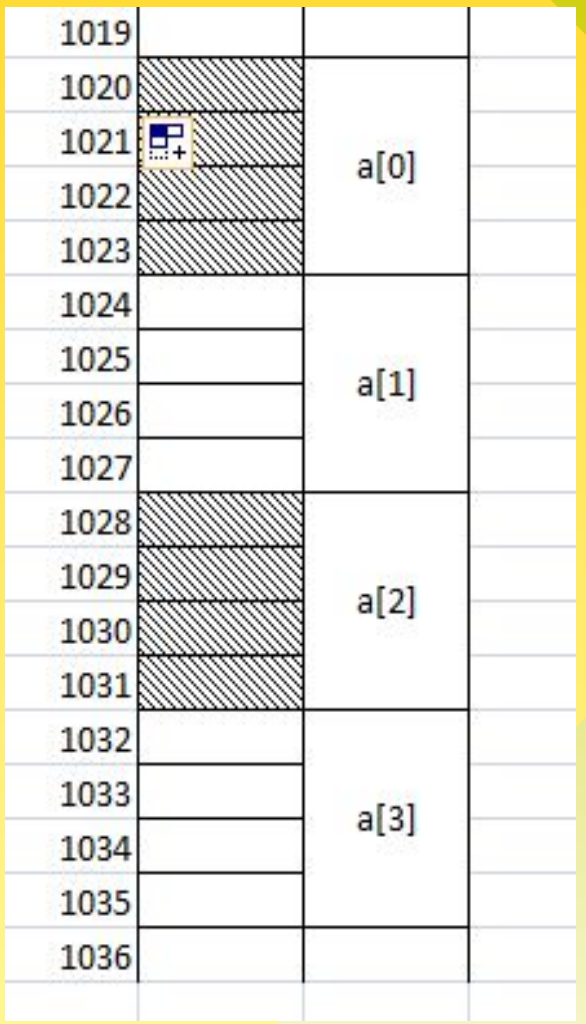
Пусть объявлен массив: ***int a***[4];

Под хранение этого массива будет зарезервировано 4 по 4 байта, т.е. 16 байт памяти.

При этом запоминается адрес нулевого элемента a[0] (пусть это ячейка 1020), адреса остальных элементов вычисляются по формуле:

адрес a[3] = адрес a[0] + 4*3 =

= 1020+4*3 = 1032



Размещение массива в памяти

int a[4]; // массив с элементами ***a***[0], ***a***[1], ***a***[2], ***a***[3]

По стандарту языка С++ при попытке обратиться к элементу a[4] – произойдет некорректное поведение программы, что возможно вызовет ее аварийное завершение.

Однако Dev-C++ позволяет обратиться к несуществующему элементу, но необходимо помнить, что

```
#include<iostream.h>
#include <stdlib.h>
using namespace std;
int main ()
{ const int n=10;
  int  A[] = {3, 5, 1, 6, 2, 4, 8, 3, 7, 2};
  cout<<"elementi massiva \n ";
  for (int i=0; i<=n; i++)
     {
       cout<<A[i]<<"   ";
     }
  cout<<endl;
 system("pause");
}
```

```
D:\ковчег\Алутина\Программирование\К занятиям\Задания к
Poluchennie elementi massiva
 3   5   1   6   2   4   8   3   7   2   2293616
Для продолжения нажмите любую клавишу . . .
```

При обращении к A[10] (несуществующему элементу) – выводит мусор

Массивы

Объявление многомерного массива:

тип имя_массива[размерностьN1]...[размерностьNM];

int a[3][5];

/**двумерный массив из 15 элементов целого типа, состоящий из трех строк по пять столбцов**/

Объявление многомерного массива с одновременной инициализацией:

тип имя_массива[размерностьN]...[размерностьM]={
{значение0, значение1, ..., значениеM-1},
...
{значениеN0, значениеN1, ..., значениеNM-1}};

int a[3][5]={ {1, 2, 3, 4, 5},
{3, 5, 2, 7, 1},
{-3, 7, 4, 1, 0}};

Двумерный массив

Пусть объявлен массив:

```
int a[3][5]={ {1, 2, 3, 4, 5},
              {3, 5, -2, 7, 1},
              {-3, 7, 4, 0, -2}};
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

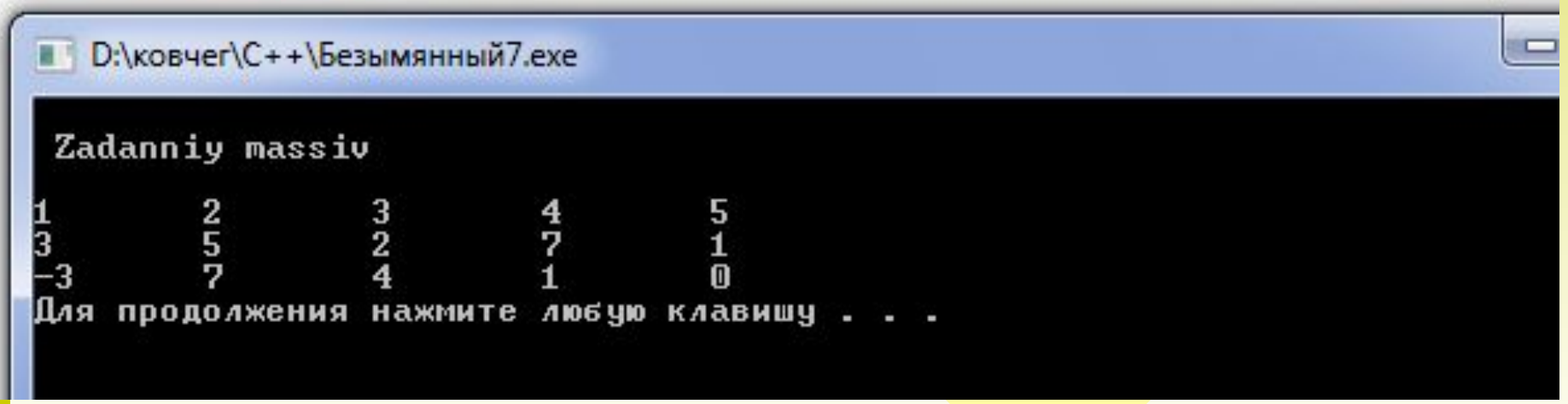
Определите значение элементов *a[2][0]* , *a[1][2]*, *a[2][3]*

Многомерные массивы

При размещении трехмерного массива int A[3][2][5] память под элементы этого массива будет выделяться последовательно в соответствии со следующими значениями индексов:

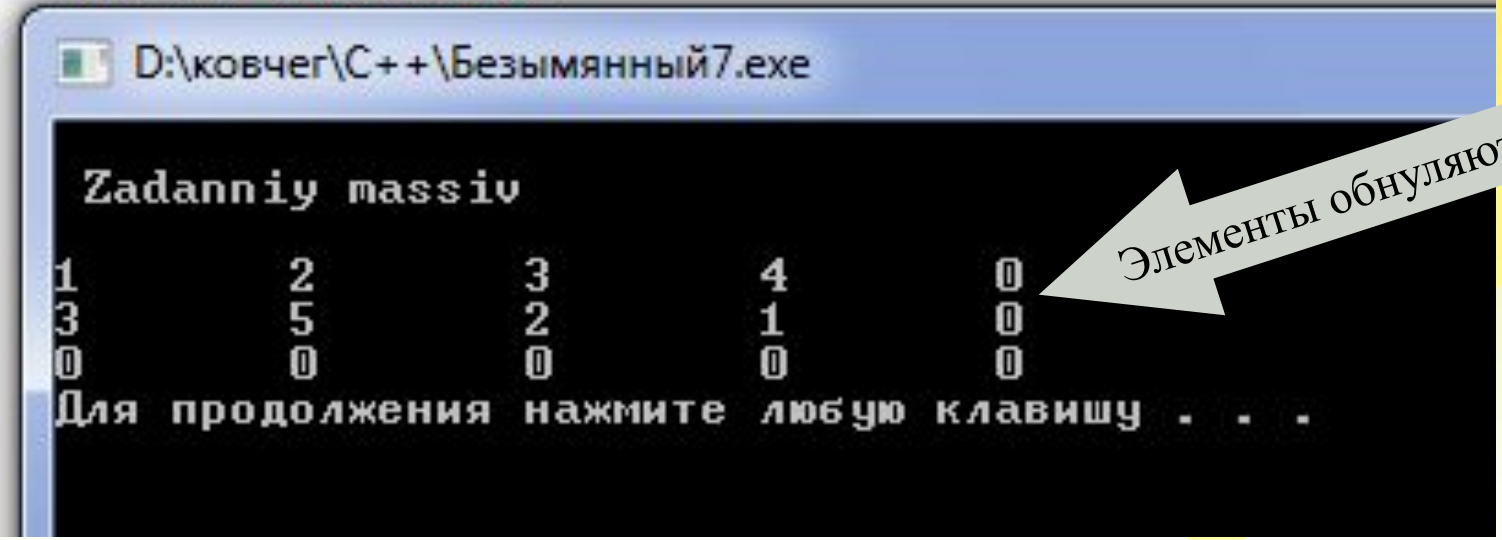
000	001	002	003	004					
010	011	012	013	014	103	104			
		110	111	112	113	114			
					200	201	202	203	204
					210	211	212	213	214

```
#include <stdlib.h>
#include <iostream>
using namespace std;
int main()
{   int i, j;
    int array[3][5]={ {1, 2, 3, 4, 5} , {3, 5, 2, 7, 1}, {-3, 7, 4, 1, 0}};
    cout<<"\n Zadanniy massiv \n";
    for (i=0; i<3; i++)
    { cout<<endl;
      for (j=0; j<5; j++)
         cout<<array[i][j]<<"\t";
     }
    cout<<endl;
    system ("Pause");
}
```



```
#include <stdlib.h>
#include <iostream>
using namespace std;
int main()
{   int i, j;
    int array[3][5]={ {1, 2, 3, 4} , {3, 5, 2, 1}};
    cout<<"\n Zadanniy massiv \n";
    for (i=0; i<3; i++)
    { cout<<endl;
      for (j=0; j<5; j++)
          cout<<array[i][j]<<"\t";
     }
    cout<<endl;
    system ("Pause");
}
```

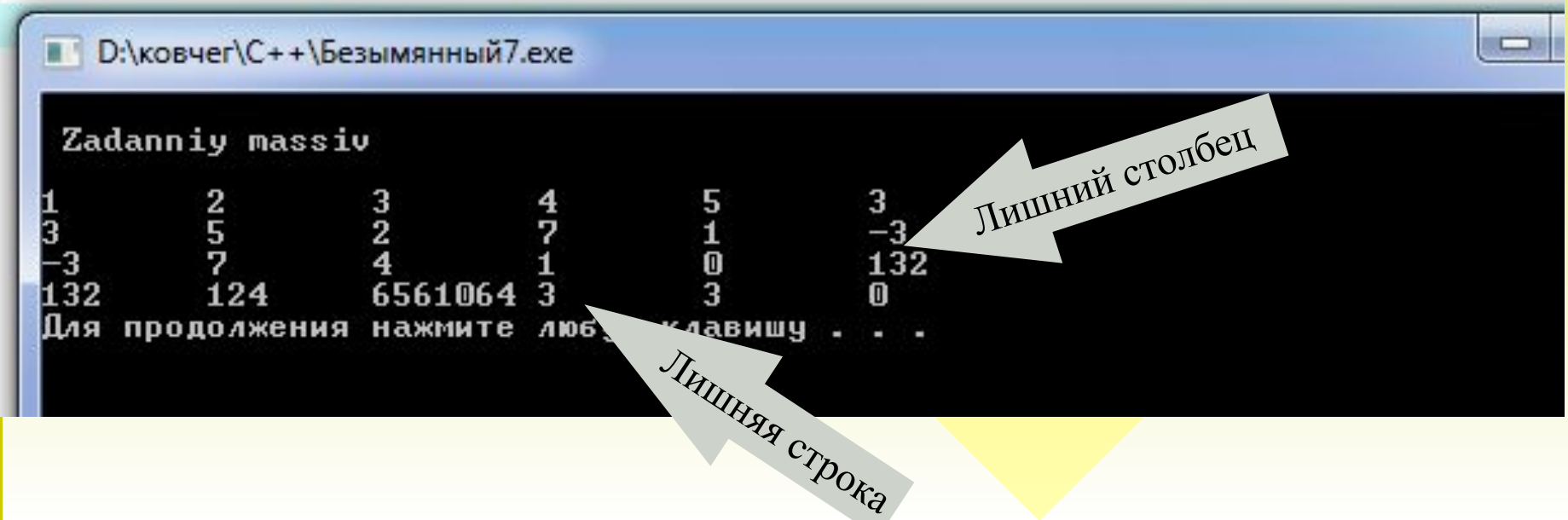
В списке инициализации задано меньше значений.



Элементы обнуляются

```
#include <stdlib.h>
#include <iostream>
using namespace std;
int main()
{   int i, j;
    int array[3][5]={ {1, 2, 3, 4, 5} , {3, 5, 2, 7, 1}, {-3, 7, 4, 1, 0}};
    cout<<"\n Zadanniy massiv \n";
    for (i=0; i<=3; i++)
    { cout<<endl;
      for (j=0; j<=5; j++)
          cout<<array[i][j]<<"\t";
     }
    cout<<endl;
    system ("Pause");
}
```

В DevCpp при попытке обратиться к несуществующим элементам массива программа работает, но выводит «мусор».



Массивы

В некоторых случаях при выходе за диапазон значений массива, компилятор аварийно завершает программу.

Отсутствие контроля индексов налагает на программиста большую ответственность. Программист обязан самостоятельно следить за границами размерностей массива, не допускать выход за пределы границ массива, помнить, что индексация начинается с 0 и на 1 меньше указанной при объявлении размерности.

Сортировка отбором

Сортируем массив по убыванию.

Пусть имеем массив: $a_0, a_1, \ldots a_{n-1}$

Фиксируем ***i***-ый элемент (сначала это ***i=0***), сравниваем его с остальными элементами (хвостом), если любой другой элемент больше a_i, то меняем их местами. Затем уже с новым значением a_i продолжаем сравнивать с остальными элементами.

После полного прохода по массиву на ***i***-ом месте (сначала на нулевом) будет находится наибольший из проверенных.

Затем сравниваем следующий (***i+1***)–***й*** с оставшимися, процедура повторяется.

Сортировка отбором

0-й проход: **7** 5 2 9 3

1-й проход: 9 **5** 2 7 3

2-й проход: 9 7 **2** 5 3

3-й проход: 9 7 5 **2** 3

Получим 9 7 5 3 2

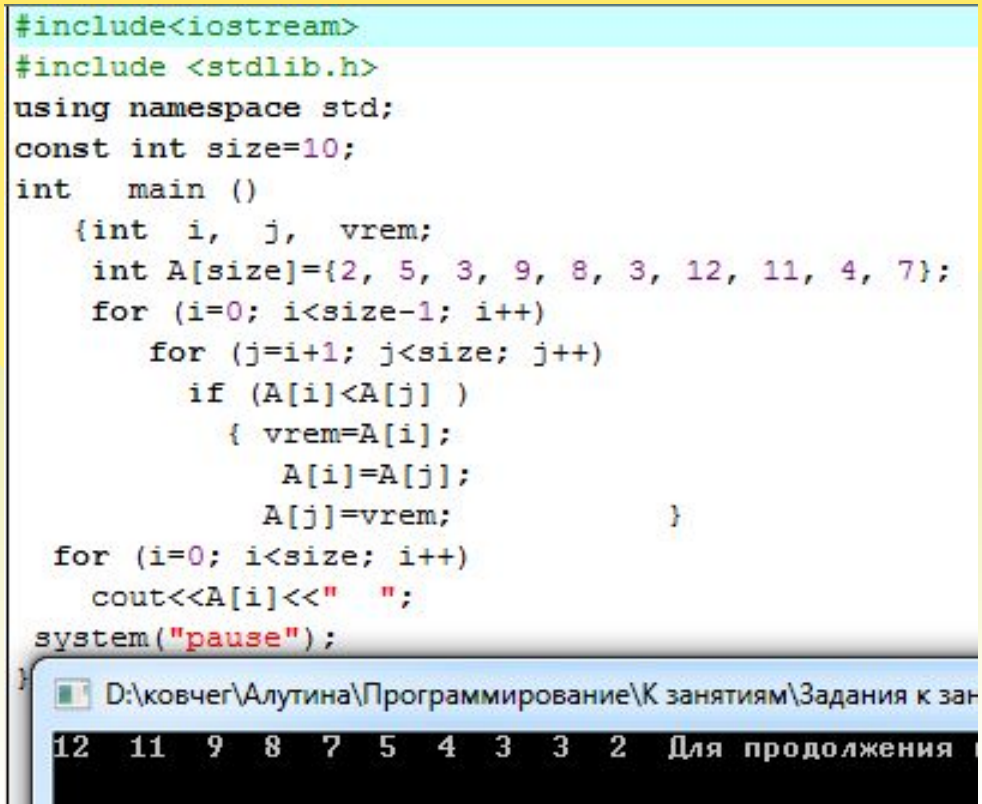
Два вложенных цикла.

Внешний цикл: от 0 до n-2

Внутренний цикл начинается со следующего за ***i*** до последнего: от (***i***+***1***) до n-1

Сортировка отбором

```
#include<iostream>
#include <stdlib.h>
using namespace std;
const int size=10;
int   main ()
   {int  i,  j,  vrem;
    int A[size]={2, 5, 3, 9, 8, 3, 12, 11, 4, 7};
    for (i=0; i<size-1; i++)
       for (j=i+1; j<size; j++)
         if (A[i]<A[j] )
           { vrem=A[i];
             A[i]=A[j];
             A[j]=vrem;
           }
  for (i=0; i<size; i++)
    cout<<A[i]<<"  ";
  system("pause");
}
```



При обмене можно обойтись без временной переменной:

a = a + b ;

b = a - b;

a = a - b;

Пусть a = 5 и b = 7;

a = 5 + 7; ⇒ a = 12;

b = 12 – 7 ⇒ b = 5;

a = 12 – 5 ⇒ a = 7

В нашем примере:

A[i]=A[i] + A[j];

A[j]=A[i] - A[j];

A[i]=A[i] - A[j];

Пузырьковая сортировка (обменом)

Сортируем по не возрастанию (убыванию).

Метод «пузырька» заключается в том, что более «легкие» элементы массива постепенно «всплывают» к концу массива. Сравнение производится парами соседних элементов и при необходимости выполняется обмен значениями.

1-й проход: 7 5 2 9 3

7 5 2 9 3

7 5 ~~2~~9 ~~9~~2 3

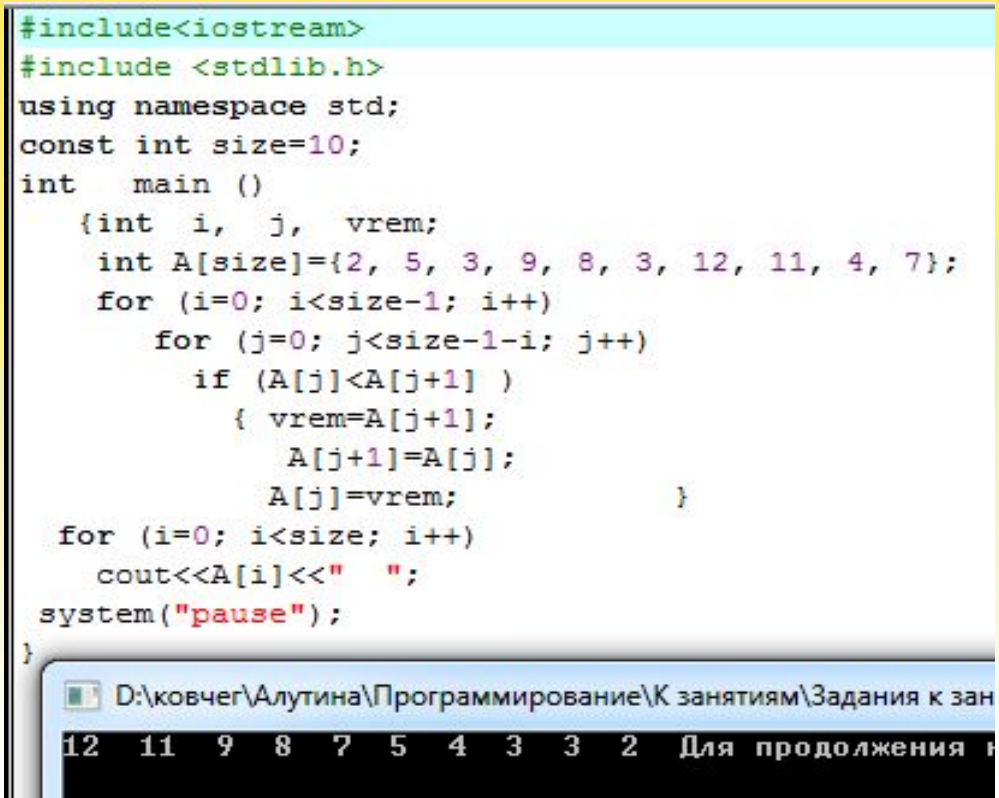
7 5 9 ~~2~~3 ~~3~~2

Пузырьковая сортировка (обменом)

```
#include<iostream>
#include <stdlib.h>
using namespace std;
const int size=10;
int   main ()
   {int  i,  j,  vrem;
    int A[size]={2, 5, 3, 9, 8, 3, 12, 11, 4, 7};
    for (i=0; i<size-1; i++)
       for (j=0; j<size-1-i; j++)
         if (A[j]<A[j+1] )
           { vrem=A[j+1];
             A[j+1]=A[j];
             A[j]=vrem;          }
  for (i=0; i<size; i++)
    cout<<A[i]<<"  ";
 system("pause");
}
```

При обмене можно обойтись без временной переменной:

A[j]=A[j]+A[j+1];
A[j+1]=A[j]-A[j+1];
A[j]=A[j]-A[j+1];



Сортировка массива методом выбора

```
const int n=10;
int   main ()
   {int i, j, k, m,  A[n]={4, 2, 8, 4, 6, 1, 3,9, 5, 11};
    for (i=1; i<n; i++)
   { m=A[i-1];
     k=i-1;
     for (j=i; j<n; j++)
        {  if (m>A[j] )
             {  m=A[j];
                k=j;            }
        }
     A[k]=A[i-1];
     A[i-1]=m;
   }
  for (i=0; i<n; i++)
    cout<<A[i]<<" ";
 system("pause");
}
```

Отличается от сортировки отбором (1-го примера) тем, что запоминаем в ***m*** A[i-1] элемент и его номер в ***k***, просматриваем массив, сохраняем в ***m*** наименьший из сравниваемых и запоминаем номер в ***k***. Обмен производим только после полного прохода по массиву (хвоста).

Сортировка массива методом выбора

```
const int n=10;
int   main ()
   {int i, j, k, m,  A[n]={4, 2, 8, 4, 6, 1, 3,9, 5, 11};
    for (i=1; i<n; i++)
   { m=A[i-1];
     k=i-1;
     for (j=i; j<n; j++)
        {  if (m>A[j-1] )
             {  m=A[j-1];
                k=j-1;        }
        }
     A[k]=A[i-1];
     A[i-1]=m;
   }
  for (i=0; i<n; i++)
    cout<<A[i]<<" ";
 system("pause");
}
```

4	2	8	4	6	1	3	9	5	11
0	1	2	3	4	5	6	7	8	9

i	m	k	j	m>A[j-1]	A[k-1]=A[i-1]	A[i-1]=m
1	4	0	1	4>4 -		
			2	4>2 +		
	2	1	3	2>8 -		
			4	2>4 -		
			5	2>6 -		
			6	2>1 +		
	1	5	7	1>3 -		
			8, 9	-	A[5]=A[0]=4	A[0]=1

Сортировка массива методом выбора

```
const int n=10;
int   main ()
   {int i, j, k, m,  A[n]={4, 2, 8, 4, 6, 1, 3,9, 5, 11};
    for (i=1; i<n; i++)
   { m=A[i-1];
      k=i-1;
     for (j=i; j<n; j++)
        {  if (m>A[j-1] )
             {  m=A[j-1];
                k=j-1;        }
        }
     A[k]=A[i-1];
     A[i-1]=m;
   }
  for (i=0; i<n; i++)
    cout<<A[i]<<" ";
 system("pause");
}
```

1	2	8	4	6	4	3	9	5	11
0	1	2	3	4	5	6	7	8	9

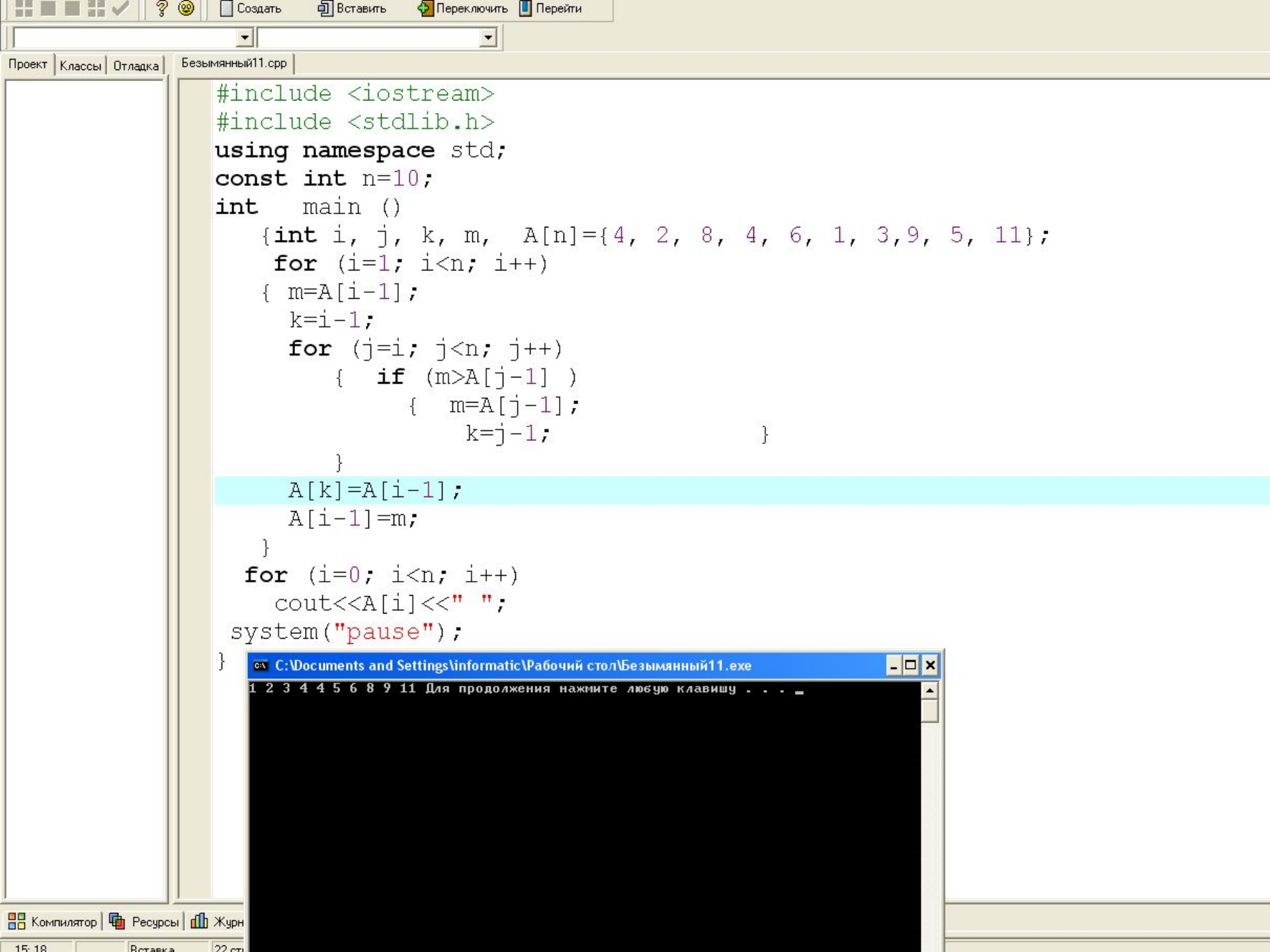
i	m=A[i-1]	k	j	m>A[j-1]	A[k-1]=A[i-1]	A[i-1]=m
2	2	1	2	2>2 -		
			3	2>8 -		
			от 4 до 9	-	A[1]=A[1]=2	A[1]=2

Сортировка массива методом выбора

```
const int n=10;
int   main ()
   {int i, j, k, m,  A[n]={4, 2, 8, 4, 6, 1, 3,9, 5, 11};
    for (i=1; i<n; i++)
   { m=A[i-1];
     k=i-1;
     for (j=i; j<n; j++)
        {  if (m>A[j-1] )
             {  m=A[j-1];
                k=j-1;        }
        }
     A[k]=A[i-1];
     A[i-1]=m;
   }
  for (i=0; i<n; i++)
    cout<<A[i]<<" ";
 system("pause");
}
```

1	2	8	4	6	4	3	9	5	11
0	1	2	3	4	5	6	7	8	9

i	m=A[i-1]	k	j	m>A[j-1]	A[k-1]=A[i-1]	A[i-1]=m
3	8	2	3	8>8 -		
			4	8>4 +		
	4	3	5	4>6 -		
			6	4>4 -		
			7	4>3 +		
	3	6	8,9	3>9 -		
				-	A[6]=A[2]=8	A[2]=3


Создать
Вставить
Переключить
Перейти
Проект
Классы
Отладка
Безымянный11.cpp
#include <iostream>
#include <stdlib.h>
using namespace std;
const int n=10;
int main ()
{int i, j, k, m, A[n]={4, 2, 8, 4, 6, 1, 3,9, 5, 11};
for (i=1; i<n; i++)
{ m=A[i-1];
k=i-1;
for (j=i; j<n; j++)
{ if (m>A[j-1])
{ m=A[j-1];
k=j-1; }
}
A[k]=A[i-1];
A[i-1]=m;
}
for (i=0; i<n; i++)
cout<<A[i]<<" ";
system("pause");
}
C:\Documents and Settings\informatic\Рабочий стол\Безымянный11.exe
1 2 3 4 4 5 6 8 9 11 Для продолжения нажмите любую клавишу . . .
Компилятор
Ресурсы
Журн
15: 18
Вставка

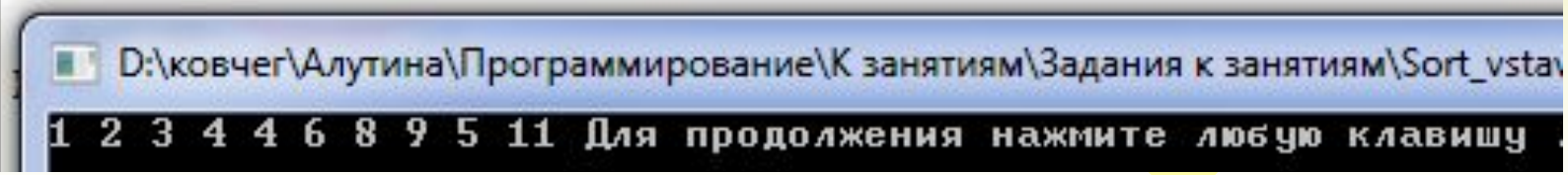
Сортировка массива простыми вставками

```
#include<iostream.h>
using namespace std;
const int m=10;
int   main ()
{ int i,j,k, l, Tmp;
  int A[m]={4, 2, 8, 4, 6, 1, 3,9, 5, 11};
i=1;
  do { j=0;
      do if (A[i]<=A[j])
         { k=i;
          Tmp=A[i];
          do
            { A[k]=A[k-1];
              k--;          }
          while (k>j);
          A[j]=Tmp;
          j=i;  }
        else (j++);
      while (j<i);
      i++;  }
  while (i<m);
   for (i=0; i<m; i++)
     cout<<A[i]<<" ";
  system("pause");              }
```

Последовательно просматриваем a_1, ..., a_{n-1} и каждый новый элемент a_i вставляем на подходящее место в уже упорядоченную совокупность a_{i-1}, ..., a_i. Это место определяется последовательным сравнением a_i с упорядоченными элементами a_0, ..., a_{i-1}.

Сортировка массива простыми вставками

```
const int m=10;
int   main ()
{ int i,j,k, n, l, Tmp;
  int A[m]={4, 2, 8, 4, 6, 1, 3,9, 5, 11};
  n=m-1;    n--;     i=1;
  do { j=0;
       do if (A[i]<=A[j])
          { k=i;
            Tmp=A[i];
            do
              { A[k]=A[k-1];
                k--;                }
            while (k>j);
            A[j]=Tmp;
            j=i;  }
          else (j++);
       while (j<i);
       i++;  }
  while (i<n);
   for (i=0; i<m; i++)
      cout<<A[i]<<" ";
```



Сортировка массива вставками

Сортировка вставками упорядочивает подсписки A[0]...A[i], 1 <= i <= n-1. Для каждого i A[i] вставляется в подходящую позицию A[j]

i определяет подсписок A[0]...A[i]

индекс j пробегает вниз по списку от A[i] в процессе поиска правильной позиции вставляемого значения

обнаружим подходящую позицию для вставки, сканируя подсписок, пока temp<A[j-1] или пока не встретится начало списка

сдвигаем элементы вправо, чтобы освободить место для вставки temp;

Сортировка массива вставками

```
#include<iostream.h>
#include <stdlib.h>
using namespace std;
int main ()
{int n=10,  A[] = {3, 5, 1, 6, 2, 4, 8, 3, 7, 2}, temp, i, j;
  for (i=1; i<n; i++)    // i определяет подсписок A[0]...A[i]
  {  // индекс j пробегает вниз по списку от A[i] в процессе
    // поиска правильной позиции вставляемого значения
    j = i;
    temp = A[i];//обнаружить подходящую позицию для вставки, сканируя подписок
    // пока temp < A[j-1] или пока не встретится начало списка
    while (j > 0 && temp < A[j-1])
    { // сдвинуть элементы вправо, чтобы освободить место для вставки
      A[j] = A[j-1];
      j--;
    }     // точка вставки найдена; вставить temp
    A[j] = temp;
  }
  for (i=0; i<n; i++)
     cout<<A[i]<<"  ";
  cout<<endl;
```

```
D:\ковчег\Алутина\Программирование\К занятиям\Задания к занятиям\Sort_Vstavkami.exe
1  2  2  3  3  4  5  6  7  8
Для продолжения нажмите любую клавишу . . .
```