3. Динамические массивы

Динамическое распределение памяти – способ выделения оперативной памяти для объектов, при котором выделение памяти осуществляется во время выполнения программы. Для создания динамических массивов в Си++ необходимо использование операции new. Для освобождения памяти – операцию delete.

3. Динамические массивы

Только первый (самый левый) размер массива может быть задан с помощью переменной.

3. Динамические массивы

Пример. Создать динамический массив, присвоив в его элементы значения от 1 до n. Вывести полученный массив на экран.

```
int n;
cout << "Enter n:";
cin >> n;
double *matr; // Указатель - имя массива
matr = new double [n]; // Массив с элементами типа double
for(int i = 0; i<n; i++)
  matr[i] = i + 1;
for(int i = 0; i<n; i++)
  cout << matr[i] << '\t';
delete [] matr;
return 0;
```

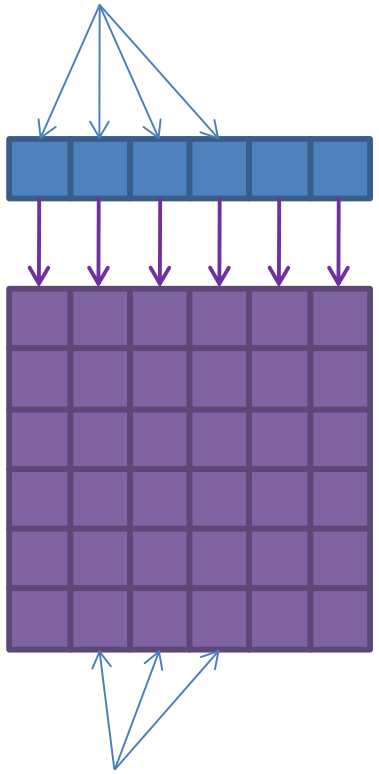
Массив matr

3. Динамические массивы

Пример. Единичная матрица с изменяемым порядком.

```
int n;
cin >> n;
double **matr; //Указатель для массива указателей
matr = new double *[n]; //Массив указателей double *
for(int i = 0; i<n; i++){
  matr[i] = new double[n];
  for(int j = 0; j<n; j++)
    matr[i][j] = (i != j ? 0 : 1);
}
cout << "Result:\n";
for(int i = 0; i<n; i++){
  for(int j = 0; j<n; j++)
    cout << '\t' << matr[i][j];
  cout << endl;
}
for(int i = 0; i < n; i++)
  delete [] matr[i];
delete [] matr;
return 0;
```

Элементы типа double *



Элементы типа double

3. Динамические массивы

Пример. Создать динамический массив размерности 3 и в каждый элемент этого массива присвоить сумму индексов этого элемента.

```
int n;
cin >> n;
double ***matr;
matr = new double ** [n];
for(int i = 0; i<n; i++){
  matr[i] = new double *[n];
  for(int j = 0; j<n; j++){
    matr[i][j] = new double [n];
    for(int k = 0; k<n; k++)
      matr[i][j][k] = i + j + k;
  }
}
for(int i = 0; i<n; i++)
  for(int j = 0; j<n; j++)
    for(int k = 0; k<n; k++)
      cout<<"matr["<<i<<"]["<<j<<"]["<<k<<"]="<<matr[i][j][k]<<'\n';
```

3. Динамические массивы (Си)

Функции библиотеки stdlib.h (cstdlib) для работы с динамической памятью:

- malloc(size) – выделение участка памяти размером size байт
- free(pointer) – освобождение памяти по указателю pointer
- calloc(n, size) – выделение участка памяти для n элементов размера size и инициализация всех битов нулями

3. Динамические массивы (Си)


```
#include <stdio.h>      /* printf, scanf, NULL */
#include <stdlib.h>     /* malloc, free, rand */
int main ()
{
  int i,n;
  char * buffer;
  printf ("How long do you want the string? ");
  scanf ("%d", &i);
  buffer = (char*) malloc (i+1);
  if (buffer==NULL) exit (1);

  for (n=0; n<i; n++)
    buffer[n]=rand()%26+'a';
  buffer[i]='\0';

  printf ("Random string: %s\n",buffer);
  free (buffer);
  return 0;
}
```

Ссылка на описание и пример:
http://www.cplusplus.com/reference/cstdlib/malloc/

3. Динамические массивы (Си)


```
int i,n;
int * buffer;

printf ("How long do you want the array? ");
scanf ("%d", &i);

buffer = (int*) calloc (i, sizeof(int));
if (buffer==NULL) exit (1);

for (n=0; n<i; n++)
  buffer[n]=n;

for (n=0; n<i; n++)
  printf("\t%d",buffer[n]);

// не забудьте освободить память: free (buffer);
```

3. Динамические массивы (Си)

Функции библиотеки stdlib.h (cstdlib) для работы с динамической памятью:

- …
- realloc(ptr, size) – выделение участка памяти размера size, обычно используется для тех случаев, когда указатель ptr уже указывает на адрес динамической памяти, и когда нужно изменить размер этого участка.

Пример: http://www.cplusplus.com/reference/cstdlib/realloc/

3. Динамические массивы

Задания.

1. Освободить память, выделенную для динамического массива matr (на предыдущем слайде).
2. Создать двумерный динамический целочисленный массив размера N на M элементов. В элементы массива присвоить значения, равные произведению номера строки на номер столбца.

4. Адресная арифметика

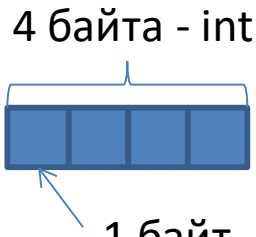
Значение выражения *указатель зависит не только от значения указателя, но и от типа.

Арифметические операции с адресами заключаются в том, что при увеличении адреса на 1 результатом является адрес соседнего «справа» блока памяти длины sizeof(тип данных), а при уменьшении на 1 – адрес соседнего «слева» блока памяти.

4. Адресная арифметика

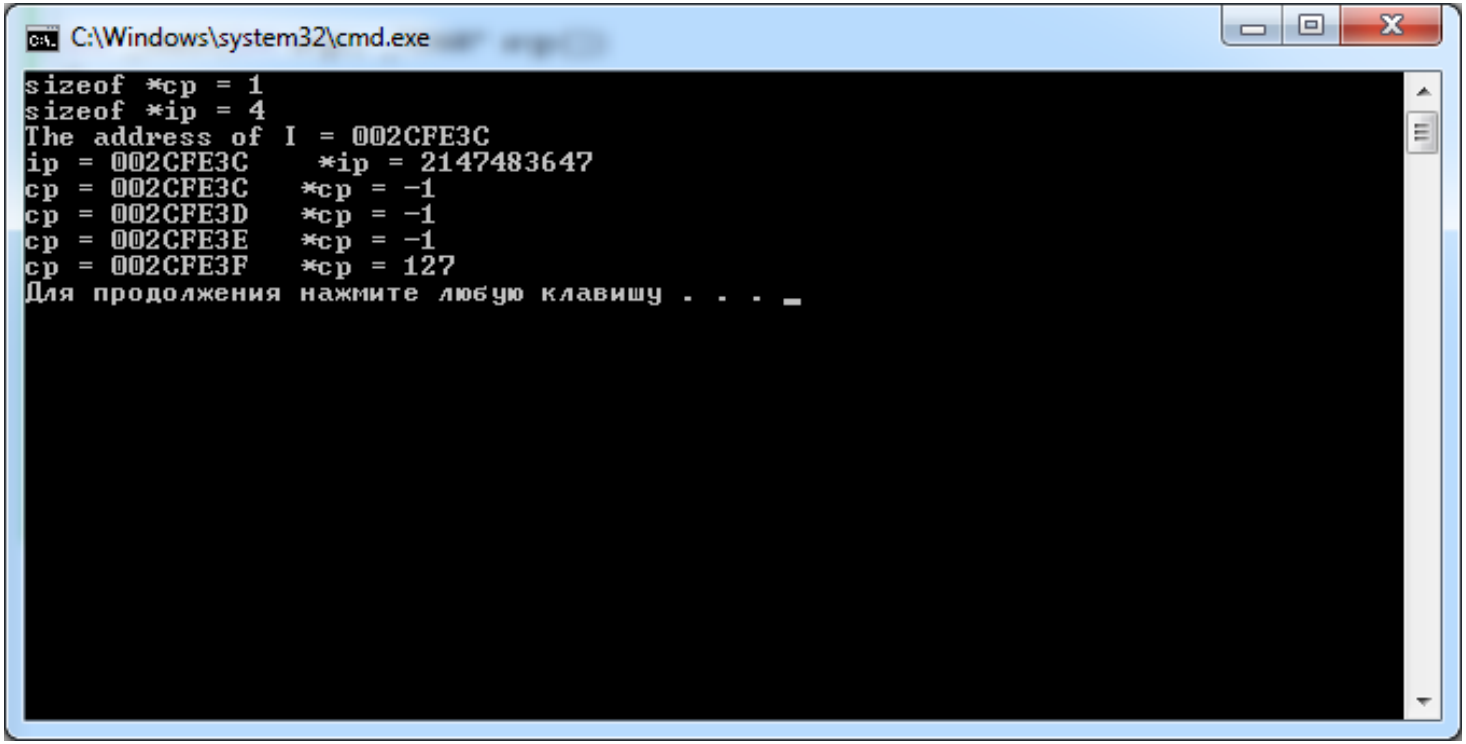
Пример. Целое число: что внутри?

```
int I = INT_MAX;
char *cp = (char*)&I;
int *ip = &I;
cout << "sizeof *cp = " << sizeof * cp << endl;
cout << "sizeof *ip = " << sizeof * ip << endl;
cout << "The address of I = " << &I << endl;
cout << "ip = " << (void*) ip << "\t *ip = " << *ip << endl;
for(; cp < (char *)ip + 4; cp++)
   cout << "cp = " << (void *)cp << "\t*cp = " << (int)*cp << endl;
return 0;
```



4. Адресная арифметика

Пример. Целое число: что внутри?



4. Адресная арифметика

В языке Си++ истинно:

имя_массива
== &имя_массива
== &имя_массива[0]

Имя массива – константный указатель того типа, к которому отнесены элементы массива.

4. Адресная арифметика

Пример. Адресная арифметика и массивы. Вывод элементов символьного массива на экран с помощью операции разыменования (*).

```
char x[] = "QWERTY";
int i = 0;
while(*(x+i)!='\0')
        cout << *(x + i++) << endl;
return 0;
```

4. Адресная арифметика

Пример. Замена операции [] на сочетание адресной арифметики и разыменования

```
int main() {
    int myArray[] = {3, 76, 5, 43};
    for(int i = 0; i < 4; i++){
        cout << "Element #" << i << " = " << *(myArray+i) << endl;
        cout << "Element #" << i << " = " << myArray[i] << endl;
    }
    return 0;
}
```

4. Адресная арифметика

Задания.

1. Ввести и вывести элементы одномерного целочисленного массива размера N без операции [] (применять адресную арифметику и операцию разыменования).
2. Вывести все отрицательные элементы одномерного целочисленного массива размера N без операции [].
3. Вывести те элементы одномерного целочисленного массива размера N на экран, которые имеют чётный индекс.

5. Сортировка пузырьком (bubble sort)


```
#define N 10
int a[N] = {6, 3, 9, 10, 0, 12, 5, -1, 4, 9};
int vspom;
for(int i = 0; i < N - 1; i++)
  for(int j = 0; j < N - i - 1; j++)
    if(a[j] > a[j + 1]){
      vspom = a[j];
      a[j] = a[j + 1];
      a[j + 1] = vspom;
    }
for(int i = 0; i < N; i++)
  cout << a[i] << '\t';
return 0;
```

5. Сортировка пузырьком (bubble sort)

С чего всё начиналось?

```
int a[N] = {6, 3, 9, 10, 0, 12, 5, -1, 4, 9};
```

Что происходит на каждой итерации внешнего цикла?

3 6 9 0 10 5 -1 4 9 12
3 6 0 9 5 -1 4 9 10 12
3 0 6 5 -1 4 9 9 10 12
0 3 5 -1 4 6 9 9 10 12
0 3 -1 4 5 6 9 9 10 12
0 -1 3 4 5 6 9 9 10 12
-1 0 3 4 5 6 9 9 10 12
-1 0 3 4 5 6 9 9 10 12
-1 0 3 4 5 6 9 9 10 12

5. Сортировка пузырьком (bubble sort)

int a[N] = {6, 3, 9, 10, 0, 12, 5, -1, 4, 9};

Что происходит на каждой итерации внутреннего цикла (для первой итерации внешнего цикла)?

3 6 9 10 0 12 5 -1 4 9
3 6 9 10 0 12 5 -1 4 9
3 6 9 10 0 12 5 -1 4 9
3 6 9 0 10 12 5 -1 4 9
3 6 9 0 10 12 5 -1 4 9
3 6 9 0 10 5 12 -1 4 9
3 6 9 0 10 5 -1 12 4 9
3 6 9 0 10 5 -1 4 12 9
3 6 9 0 10 5 -1 4 9 12

5. Сортировка вставками (insertion sort)


```
for(int i=1; i<n; i++)
      for(int j=i; j>0 && x[j-1]>x[j]; j--){
            int tmp=x[j-1];
            x[j-1]=x[j];
            x[j]=tmp;
            }
```

Главная идея алгоритма: есть уже упорядоченная часть массива, цель очередной итерации внешнего цикла – расширить упорядоченную часть вставкой первого элемента из неупорядоченной части в упорядоченную

5. Сортировка вставками (insertion sort)

С чего всё начиналось?

```
int x[N] = {6, 3, 9, 10, 0, 12, 5, -1, 4, 9};
```

Что происходит на каждой итерации внешнего цикла?

3 6 9 10 0 12 5 -1 4 9
3 6 9 10 0 12 5 -1 4 9
3 6 9 10 0 12 5 -1 4 9
0 3 6 9 10 12 5 -1 4 9
0 3 6 9 10 12 5 -1 4 9
0 3 5 6 9 10 12 -1 4 9
-1 0 3 5 6 9 10 12 4 9
-1 0 3 4 5 6 9 10 12 9
-1 0 3 4 5 6 9 9 10 12

Подчёркнута упорядоченная часть массива
Красным выделен первый неупорядоченный элемент

5. Сортировка выбором (selection sort)


```
for (int i = 0; i < size - 1; i++) {
    /* устанавливаем начальное значение минимального индекса */
    int min_i = i;

    /* находим индекс минимального элемента */
    for (int j = i + 1; j < size; j++) {
        if (array[j] < array[min_i])
            { min_i = j; }
    }

    /* меняем значения местами */
    int temp = array[i];
    array[i] = array[min_i];
    array[min_i] = temp;
}
```

5. Сортировка выбором (selection sort)

С чего всё начиналось?

```
int array[] = {4, -5, 6, 2, 66, 44, 0, -1};
```

Что происходит на каждой итерации внешнего цикла?

-5 4 6 2 66 44 0 -1 // поменялись местами первый и минимальный элементы
-5 -1 6 2 66 44 0 4 // второй и минимальный после второго
-5 -1 0 2 66 44 6 4 // третий и минимальный после третьего
-5 -1 0 2 66 44 6 4 // и т. д.
-5 -1 0 2 4 44 6 66
-5 -1 0 2 4 6 44 66
-5 -1 0 2 4 6 44 66