Массивы

```
//объявление массива из 10 элементов типа int
int array [10];

//объявление массива длиной 6 и инициализация трех
//первых элементов
double vector [6] = {2, 3.6, 0};

//объявление и инициализация массива длиной 4
//размерность вычисляется компилятором
//похоже на строку, но еще не строка…
char string [] = {'w', 'o', 'r', 'd'};
```

Строки

В C++ нет встроенного типа «строка», НО строка – это массив символов

```
char correctString [] = “word”;                    //строка

char theSameString [] = {‘w’, ‘o’, ‘r’, ‘d’, 0};   //строка

char charArray [] = {‘w’, ‘o’, ‘r’, ‘d’};          //массив символов
```

Строка в C++ заканчивается символом с кодом 0 (‘\0’)

Обращение к элементам массива

```
int i, a [4] = {10, 20, 30 ,40};
for (i = 0; i < 4; ++i)
        printf (“a[%d] = %d\n”, i, a[i]);

int b = a[4]; /*выход за границу массива;
результат исполнения инструкции непредсказуем*/

char ch = “word”[3];        //ch == ‘d’
```

```
a[0] = 10
a[1] = 20
a[2] = 30
a[3] = 40
```

Обращение к элементам массива

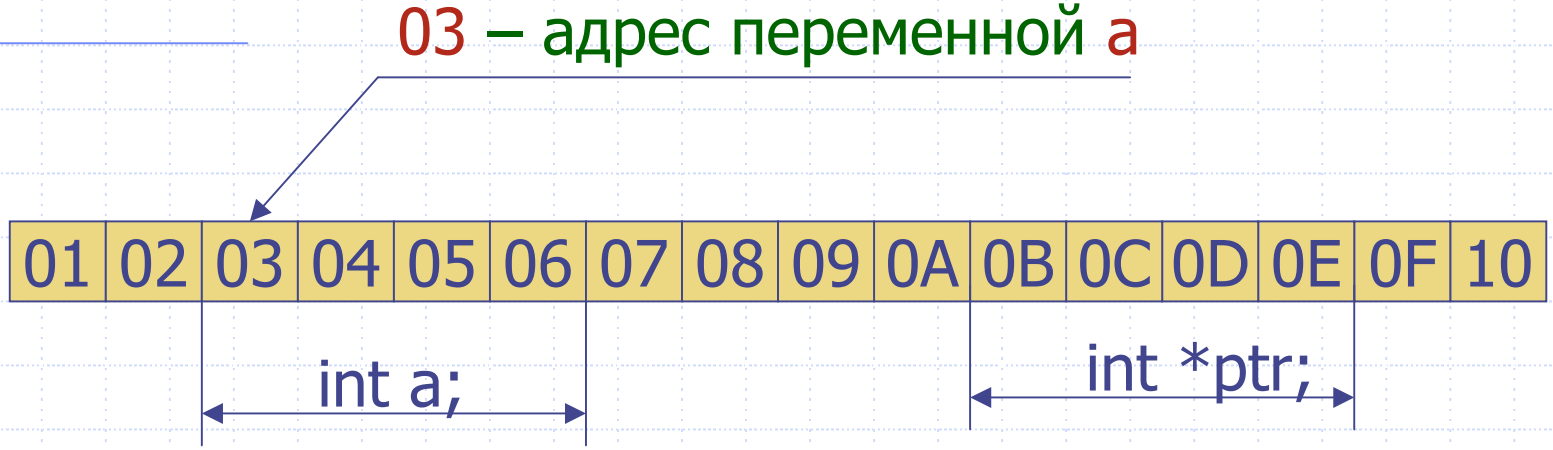
Замечания:

1. Если объявлен массив a[N], то он индексируется от 0 до N-1
2. В C++ НЕ контролируется выход за границу массива
3. Размер массива должен быть вычисляем на стадии компиляции (в старом стандарте), то есть не должен задаваться значениями переменных
   int a = 10, b [a]; //ошибка
4. Обращение к элементу массива – это lvalue

Рекомендации:

1. Если массив не является строкой, определяйте явно его размер
2. Следите за границами массива

Адресация




```
int a = 5;
int *ptr = &a;
printf ("address of a = %p\na = %d\n",
        ptr, *ptr);
*ptr = 10;    //a = 10
printf ("address of a = %p\na = %d\n",
        ptr, a);
```

```
address of a = 03
a = 5
address of a = 03
a = 10
```

Указатели (pointers)

Указатель – переменная, содержащая адрес объекта заданного типа. Под объектами в частности понимаются переменные базовых типов.

```
type *имя_указателя;      //указатель на объект типа type
```

```
int *intPtr;                        //указатель на int
double a;
double *dblPtr = &a;          /*указатель на double
        инициализируется адресом переменной a*/
double **dbl2Ptr = &dblPtr;        /*указатель на указатель
        на double (двойной указатель)*/
long *lPtr = 0;                     //указатель в «никуда»
```

Операции с указателями

```
&lvalue        //получение адреса объекта

*expr          //обращение по адресу (разыменование)
```

Замечания:

1. Указатель может быть инициализирован адресом объекта или 0
2. Гарантируется отсутствие объекта по адресу 0; это единственный адрес, который можно написать явно
3. Результат разыменование неинициализированного указателя (или нулевого) непредсказуем
4. Разыменованный указатель является еще одним именем объекта, на который указывает

Арифметические операции с указателями

```
int array [10];

int *ptr = &array[0]; //указатель на нулевой элемент

*ptr = 15;                //array[0] == 15

int ptr1 = ptr + 1;   //указатель на 1ый элемент

++ptr1;                   //теперь указывает на 2ой

*(ptr + 10) = 1;      //выход за границу массива!!!

int offset = ptr1 - ptr; //offset == 2
```

Арифметические операции с указателями

```
ptr + (int)N    //:ptr, смещение на N элементов

ptr - (int)N    //:ptr, обратное смещение на N элементов

ptr - ptr1      //:int, смещение между указателями
                //если указатели в разных массивах –
                //результат непредсказуем

++, --          //префиксный/постфиксный инкремент и
                //декремент

+=, -=          //присваивания
```

Указатели и массивы

```
int array[10];
int *ptr = array;      //ptr указывает на нулевой элемент
ptr [5] = 10;          //array[5] == 10
*array = 2;            //array[0] == 2
*(array + 3) = 7;      //array[3] == 7
```

Выводы:

1. Имя массива является указателем за исключением следующего:
   - Невозможно присвоить значение имени массива
   - sizeof от имени массива дает размер массива, а от указателя – размер, занятый указателем (4 или 8 байт)
2. Операция *(ptr + N) эквивалентна ptr[N] (этим и объясняется индексация массивов с 0).

Динамические массивы

Размер массива задается статически и должен быть известен на момент компиляции программы (старый стандарт).
В некоторых случаях размер требуемой памяти вычисляется во время выполнения программы.
Как быть в этом случае?

1. Создать массив, которого хватит при любых исходных данных (FORTRAN).
2. Каким-то образом выделить память во время выполнения программы (динамически)

Динамическое управление памятью

```
new type                          //создать один элемент
delete (type*)expr                //удалить элемент
new type [(unsigned)expr]         //создать несколько элементов
delete [] (type*)expr             //удалить несколько элементов
```

Динамическое управление памятью

```
int *iptr;
iptr = new int;          //создание объекта в памяти типа int

/*iptr = new char;*///2 ошибки: создан объект не того
                       //типа, не удален старый объект
delete iptr;           //освобождение памяти

int size = 150;        //размер будущего массива
iptr = new int[size]; //создание массива типа int

/*delete [] iptr+1*/ //delete применяется к указателю,
                       //возвращаемому new
delete [] iptr;        //освобождение памяти
```

Инструкция по “уходу” за динамической памятью

1. Храните размеры динамически выделяемых массивов. Менеджер памяти его знает (delete [] применяется без указания размера), но никому не рассказывает!
2. Всегда удаляйте память, которую выделили динамически, иначе возникнут утечки памяти (memory leaks).
3. Если память выделена с помощью new[], удаляйте ее с помощью delete[].
4. Применяйте delete и delete[] только к тем указателям, которые были возвращены new и new[]. Идентификатором выделенной памяти является указатель на ее первый элемент (байт).

Многомерные массивы

```
int array2d [10][15];
array2d [2][3] = 10;
```

Если объявлен массив array2d[M][N], то обращение array2d[x][y] эквивалентно *(*(array2d+x)+y) или *(*array2d+N*x+y)

Имя двумерного массива напоминает указатель на указатель (двойной указатель), однако оно не ссылается на реальный указатель в памяти, это «подделка».

Недостатки многомерных массивов

1. Размеры массивов статические.

2. Двойной массив нельзя передать в качестве параметра функции

Альтернативы многомерным массивам

```
int *array2d[10];     //массив указателей
int i;
for (i = 0; i < 10; ++i)
        array2d[i] = new int [15];
array2d [2][3] = 10;
…
for (i = 0; i < 10; ++i)
        delete [] array2d[i];
```

Альтернативы многомерным массивам

```
int **array2d;          //двойной указатель
array2d = new int* [10];
int i;
for (i = 0; i < 10; ++i)
        array2d[i] = new int [15];
array2d [2][3] = 10;
...
for (i = 0; i < 10; ++i)
        delete [] array2d[i];
delete [] array2d;
```

Параметры main

```
//hello.cpp
#include <stdio.h>

int main (int argc, char *argv[])
{
        printf (“params = %d\n”, argc);
        if (argc > 1)  printf (“Hello, %s!\n”, argv[1]);
        return 0;
}
```

```
> hello.exe people
params = 2
Hello, people!
```

argc – количество аргументов
argv – массив аргументов (строк)

Обработка строк и массивов.

```
//стандартная библиотека работы
//со строками и массивами
#include <string.h>


//длина строки без \0
size_t  strlen (const char * str);
```

```
char str [] = “hello”;
size_t size = strlen (str);    //size == 5
```

Обработка строк и массивов.

//копирование строки

```
char*  strcpy (char * dest, const char * src);
```

```
char srcStr [] = "Hello";
char *destStr = new char [strlen(srcStr) + 1];
strcpy (destStr, srcStr);
printf ("%s\n", destStr);     //Hello
```

//копирование памяти

```
void* memcpy (void * dest, const void * src,  size_t size);
```

```
int src [4] = {1, 2, 3, 4};
int * dest = new int [4];
memcpy (dest, src, 4*sizeof(int));
```

Обработка строк и массивов.

```
//лексикографическое сравнение строк
//str1 == str2 ? 0: str1 < str2 ? –1 : 1
int  strcmp (const char * str1, const char * str2);

int  strncmp (const char * str1, const char * str2, size_t size)


//лексикографическое (побайтовое) сравнение массивов
int  memcmp (const void *arr1, const void *arr2, size_t size)
```

Обработка строк и массивов.

```
//перевод в нижний регистр
char *  strlwr (char * str);

//перевод в верхний регистр
char *  strupr (char * str);


//конкатенация строк
char * strcat (char *dest, const char *src);

char * strncat (char *dest, const char *src, size_t size);
```

Обработка строк и массивов.

```
//инициализация памяти
void *  memset (void *dest, int c, size_t n);
```

```
int * array = new int [1000];
int i;

//инициализация массива нулями
for (i = 0; i < 1000; ++i) array [i] = 0;

//то же, но гораздо быстрее
memset ((void*) array, 0, 1000 * sizeof (int));
```

Разборка (parsing) строк

```
//ищем максимум двух чисел, введенных в
//коммандной строке
#include <stdio.h>
int main (int argc, char *argv[])
{
        if (argc < 3) return 1;
        double a, b;
        sscanf (argv[1], "%lg", &a);
        sscanf (argv[2], "%lg", &b);
        printf ("max = %lg\n", a > b ? a:b);
        return 0;
}
```

```
> max.exe 12.8 2e3
max = 2000
```

Интерактивный ввод

```
//ищем максимум двух чисел, введенных в
//интерактивном режиме
#include <stdio.h>
int main (void)
{
        double a, b;
        printf ("Enter numbers >");
        scanf ("%lg,%lg", &a, &b);
        printf ("max = %lg\n", a > b ? a:b);
        return 0;
}
```

```
> max.exe
Enter numbers >12.8,2e3
max = 2000
```