Кирилл Волков @ MERA
github.com/vulko/Cpp_Basics_Lectures

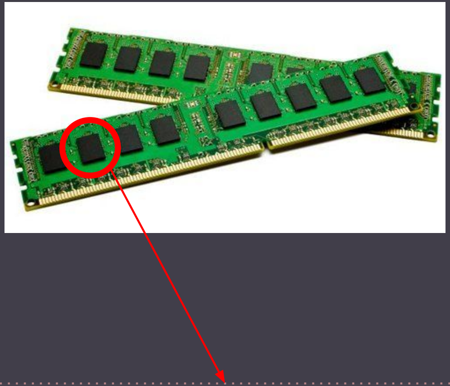
C++

Указатели и ссылки: выделение памяти, работа с указателями, работа со ссылками, разница между ссылками и указателями, указатели и ссылки как аргументы функций

Массивы и строки: одномерные и двумерные статические и динамические массивы.

Отличия массива и указателя. Строковые типы.

RAM: Random Access Memory



1 byte = 8 bit

0	1	1	0	1	1	0	0

Byte matrix
(physical level)

1	2	3	…
…	…	…	…
…	…	…	N

Abstraction of RAM byte

Address of cell

32 bit:

0x00000000

```
2^4 = 16
1 HEX digit = 4 bits

Hexadecimal: 3
Binary:      0 0 1 1

Hexadecimal: F
Binary:      1 1 1 1
```

Указатели и ссылки

Pointer type

```
int a = 10;
// allocate 4 bytes (32 bits) of memory

int* b = 0xFFFFFFFF;
// pointer to an integer value requires 4 bytes


char a = 10;
// allocate 1 byte (8 bits) of memory

char* b = 0xFFFFFFFE;
// pointer to a char value requires 4 bytes
```

Reference type

```
int a = 10;
int *b = &a;

a++;
// now a=11.
// Value of memory cell,
// that b points is also 11

*b = -10;
// dereference pointer (turn to value type)
// now a = -10 as well!
```

Указатели и ссылки

Pointer initialization

```
int value = 111;

int* somePointer;
// uninitialized

somePointer = &value;
// now it points to value

somePointer = NULL;
// now it NULL

somePointer = nullptr;
// g++ style
```

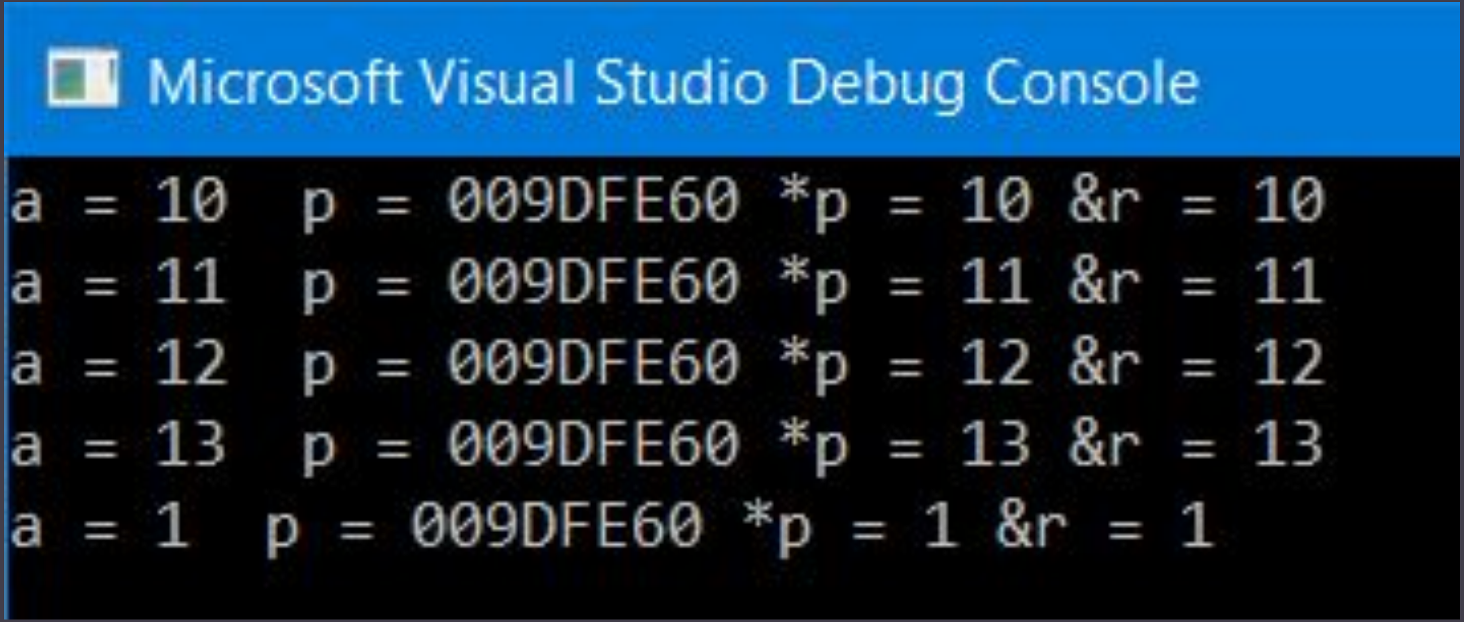
Memory allocation/deallocation

```
int* somePointer;
// uninitialized

somePointer = new int;
// allocated 4 bytes in RAM. Pointer stores address

somePointer +=1;
// now it points to next 4 bytes... could be unallocated!

somePointer -=1;
// points again to the allocated 4 bytes

...

delete somePointer;
// always release unused objects to avoid memory leaks
```

Указатели и ссылки

Reference type

```
int a = 10;
int *p = &a;
int &r = *p;

std::cout << "a = " << a << "  p = " << p << " *p = " << *p << " &r = " << r << std::endl;

a++;
std::cout << "a = " << a << "  p = " << p << " *p = " << *p << " &r = " << r << std::endl;

(*p)++;
std::cout << "a = " << a << "  p = " << p << " *p = " << *p << " &r = " << r << std::endl;

r++;
std::cout << "a = " << a << "  p = " << p << " *p = " << *p << " &r = " << r << std::endl;

int b = 1;
r = b;
std::cout << "a = " << a << "  p = " << p << " *p = " << *p << " &r = " << r << std::endl;
```



```
a = 10  p = 009DFE60 *p = 10 &r = 10
a = 11  p = 009DFE60 *p = 11 &r = 11
a = 12  p = 009DFE60 *p = 12 &r = 12
a = 13  p = 009DFE60 *p = 13 &r = 13
a = 1  p = 009DFE60 *p = 1 &r = 1
```

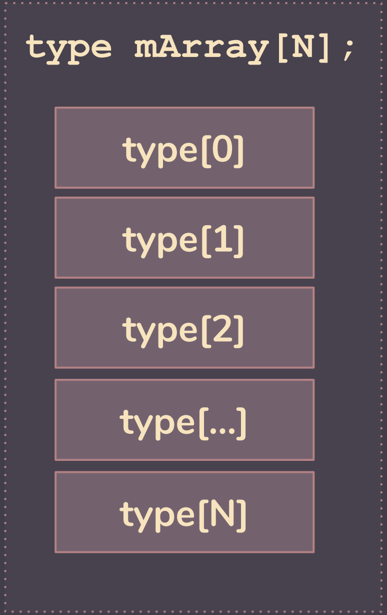
Pass by pointer

```
int main() {
    int a;
    for (a = 0; a < 20; increase(&a)) {
        cout << “this will be printed 20 times!”;
    }

    if (a != 19) {
        // this will never happen!
    }
}

void increase(int *val) {
    *val += 1;
}
```

Pass by reference

```
int main() {
    int a;
    for (a = 0; a < 20; increase(a)) {
        cout << “this will be printed 20 times!”;
    }

    if (a != 19) {
        // this will never happen!
    }
}

void increase(int &val) {
    val += 1;
}
```

Массивы



```
type mArray[N];
```

```
int array[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

char c_string[50] = “Hello World”;
```

```
int* array;
...
array = new int[100];
array[0] = 0;
...
delete[] array;
...
array = new int[200];
```

Массивы

```
int array[n][m] =
{
   {(0,0)}, {...}, {(0,m)},
   {...},     {...}, {...},
   {(n,0)}, {...}, {(n,m)}
};
```

```
int** array;

array = new int*[n];
for (int i = 0; i < n; ++i) {
    array[i] = new int[m];
}
```

```
for (int i = 0; i < m; ++m) {
   for (int j = 0; j < n; ++n) {
      array[j][i] = 123;
   }
}
```

```
type mArray[N][M];
```

type[0][0]	...	type[0][M]
type[1][0]	...	type[1][M]
type[2][0]	...	type[2][M]
type[...][...]	...	type[...][...]
type[N][0]	...	type[N][M]

Массивы и указатели

```
int numbers[] = {11, 22, 33};
int* ptr = numbers;
cout << ptr << endl;                  // 0x22cd30
cout << ptr + 1 << endl;              // 0x22cd34 (increase by 4 - sizeof int)
cout << *ptr << endl;                 // 11
cout << *(ptr + 1) << endl;           // 22
cout << *ptr + 1 << endl;             // 12
```

```
int numbers[100];

cout << sizeof(numbers) << endl;
// Size of entire array in bytes (400)

cout << sizeof(numbers[0]) << endl;
// Size of first element of the array in bytes (4)

cout << "Array size is " << sizeof(numbers) / sizeof(numbers[0]) << endl;
// Array size is 100
```

C-Style string

```
int main() {
    char str1[] = "Hello";
    char *str2 = "Hello";         // warning: deprecated conversion from string constant to 'char*'

    cout << strlen(str1) << endl;                // 5
    cout << strlen(str2) << endl;
    cout << strlen("Hello") << endl;

    int size = sizeof(str1) / sizeof(char);
    cout << size << endl;                        // 6 - including the terminating '\0'
    for (int i = 0; str1[i] != '\0'; ++i) {
        cout << str1[i];
    }
    cout << endl;

    for (char *p = str1; *p != '\0'; ++p) {      // *p != '\0' is the same as *p != 0, is the same as *p
        cout << *p;
    }
}
```

STL string

```
std::string mSomeString = “This is simple string”;
cout << mSomeString;                                    // This is simple string

mSomeString.push_back(‘!’);
cout << mSomeString;                                    // This is simple string!

mSomeString.pop_back();
cout << mSomeString;                                    // This is simple string
```

```
// mem usage

capacity()
resize()
shrink_to_fit()
```

```
// iterators

begin()
end()
rbegin()
rend()
```

```
// copy and swap

copy(“char array”, len, pos)
swap()
c_str()
```