Презентации, используемые на лекциях будут доступны тут:

github.com/posgen/OmsuMaterials

GitHub, Inc. [US] | https://github.com/posgen/OmsuMaterials

README.md

Учебные материалы, ОмГУ

Для 2-го курса.

Потенциально полезные файлы находятся выше в директории "2course/Programming/".

В https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/examples/ - приводятся программы с подробными, по возможности, комментариями - что да как. Смотреть на пронумерованные директории.

В https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/examples/from_study_guide/ - все программы из методического пособия в рабочем виде.

В https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/docs/ - размещаются краткие описания по компиляции и запуску программ, а также справочные материалы по C/C++

Файлы будут периодически обновляться

Краткие или не очень руководства и справки по некоторым темам будут появляться тут:
https://github.com/posgen/OmsuMaterials/wiki

PDF'ки с лекций доступны здесь:
https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/docs/lectures

Для 4-го курса, предмет "Компьютерное моделирование фазовых переходов"

Базовые программы и некоторые справочные документы находятся в директории "4course/phaseTransitions/"

GitHub, Inc. [US] | https://github.com/posgen/OmsuMaterials

README.md

Учебные материалы, ОмГУ

Для 2-го курса.

Потенциально полезные файлы находятся выше в директории "2course/Programming/".

В https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/examples/ - приводятся программы с подробными, по возможности, комментариями - что да как. Смотреть на пронумерованные директории.

В https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/examples/from_study_guide/ - все программы из методического пособия в рабочем виде.

В https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/docs/ - размещаются краткие описания по компиляции и запуску программ, а также справочные материалы по C/C++

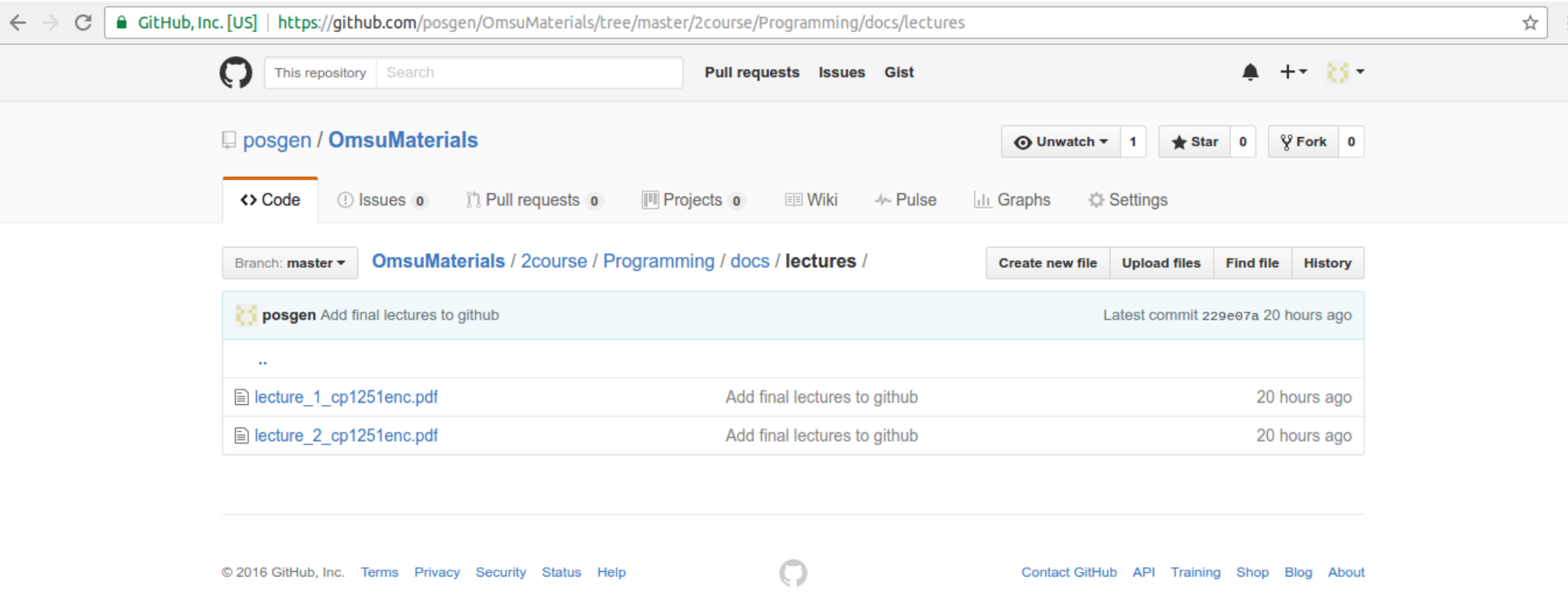
Файлы будут периодически обновляться

Краткие или не очень руководства и справки по некоторым темам будут появляться тут:
https://github.com/posgen/OmsuMaterials/wiki

PDF'ки с лекций доступны здесь:
https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/docs/lectures

Для 4-го курса, предмет "Компьютерное моделирование фазовых переходов"

Базовые программы и некоторые справочные документы находятся в директории "4course/phaseTransitions/"


GitHub, Inc. [US] | https://github.com/posgen/OmsuMaterials/tree/master/2course/Programming/docs/lectures
This repository
Search
Pull requests
Issues
Gist
posgen / OmsuMaterials
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Pulse
Graphs
Settings
Branch: master
OmsuMaterials / 2course / Programming / docs / lectures /
Create new file
Upload files
Find file
History
posgen Add final lectures to github
Latest commit 229e07a 20 hours ago
..
lecture_1_cp1251enc.pdf
Add final lectures to github
20 hours ago
lecture_2_cp1251enc.pdf
Add final lectures to github
20 hours ago
© 2016 GitHub, Inc.
Terms
Privacy
Security
Status
Help
Contact GitHub
API
Training
Shop
Blog
About

Лекция IV

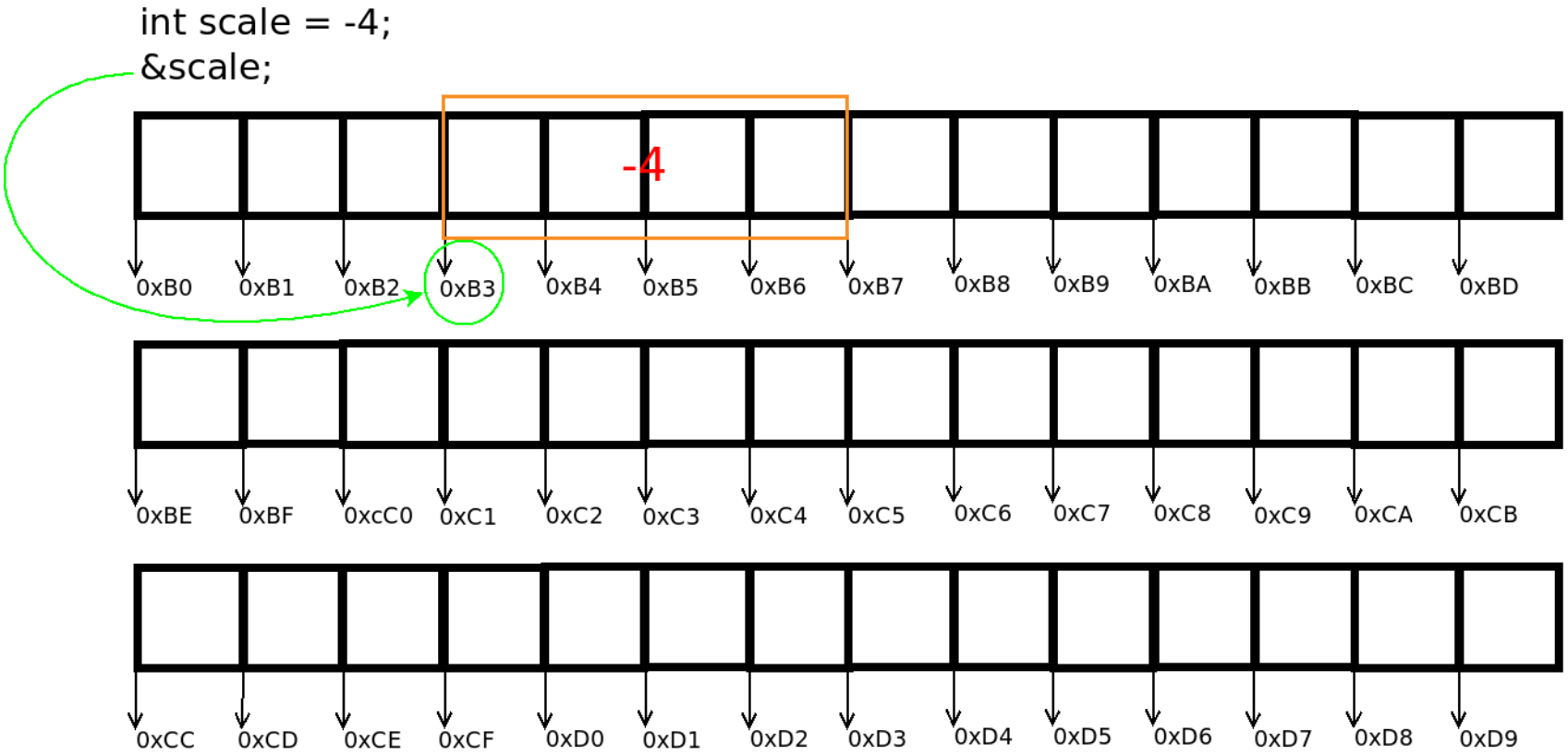
22 октября 2016

Адрес переменной

У переменной можно узнать адрес ячейки памяти, в которой она располагается с помощью оператора - **&**

```
1 int scale = 5;
2 double rate = 3.4;
3
4 std::cout << "Адрес rate: " << &rate;
5 std::cout << "\nАдрес scale: " << &scale;
```

Адрес переменной



Указатели

Указателем - называют тип данных, переменные которого предназначены для хранения адресов ячеек памяти. Указатели в C/C++ являются типизированными.

Синтаксис объявления указателя

```
<тип_данных> *<имя_переменной>;
```

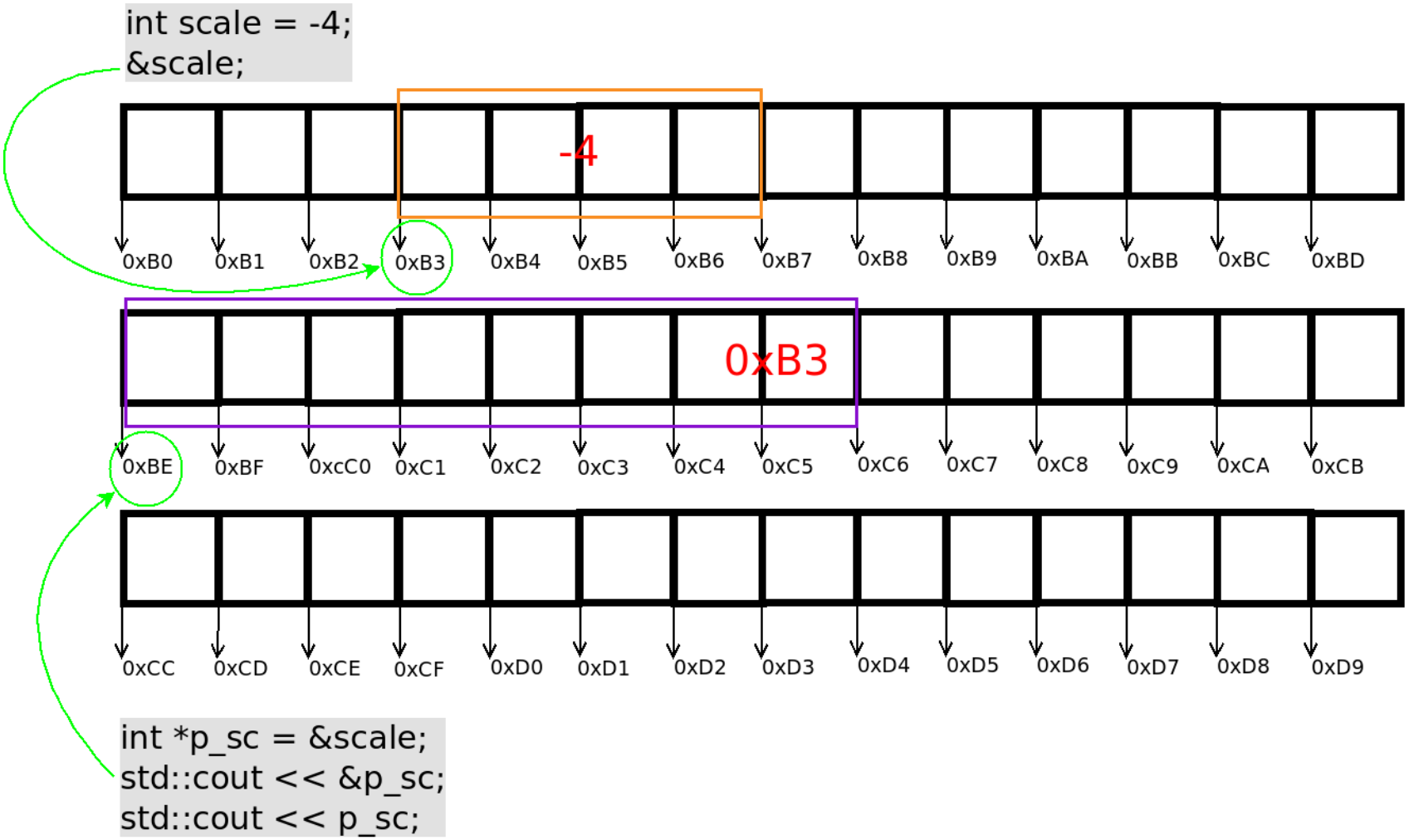
```
1 int    *p_int;    // указатель на int
2 char   *p_char;   // указатель на char
3 double *p_double; // указатель на double
```

Операции с указателями: **присвоение значения**

```
1 int scale = 5, *p_sc;
2
3 p_sc = &scale;
4 int *p2 = p_sc;
```

1-я строка: определяем переменную целого типа **scale** и указатель на целое **p_sc**.
3-я строка: присваиваем указателю **p_sc** значения, равное адресу ячейки памяти, в которой находится переменная **scale**.
4-я строка: присваиваем указателю **p2** значения, которое находится в указателе **p_sc**.

Указатели


int scale = -4;
&scale;
-4
0xB0
0xB1
0xB2
0xB3
0xB4
0xB5
0xB6
0xB7
0xB8
0xB9
0xBA
0xBB
0xBC
0xBD
0xB3
0xBE
0xBF
0xcC0
0xC1
0xC2
0xC3
0xC4
0xC5
0xC6
0xC7
0xC8
0xC9
0xCA
0xCB
0xCC
0xCD
0xCE
0xCF
0xD0
0xD1
0xD2
0xD3
0xD4
0xD5
0xD6
0xD7
0xD8
0xD9
int *p_sc = &scale;
std::cout << &p_sc;
std::cout << p_sc;

Операции с указателями: **присвоение значения**: нулевой указатель

```
1 // C-версия
2 double *p1 = NULL;
3
4 // современный C++
5 int *p2 = nullptr;
6
7 if (p2 == nullptr) {
8   // что-нибудь делаем
9 }
```

Указатели

Операции с указателями: **разыменование** - получение значения, содержащегося в ячейке, адрес которой сохранён в указателе.

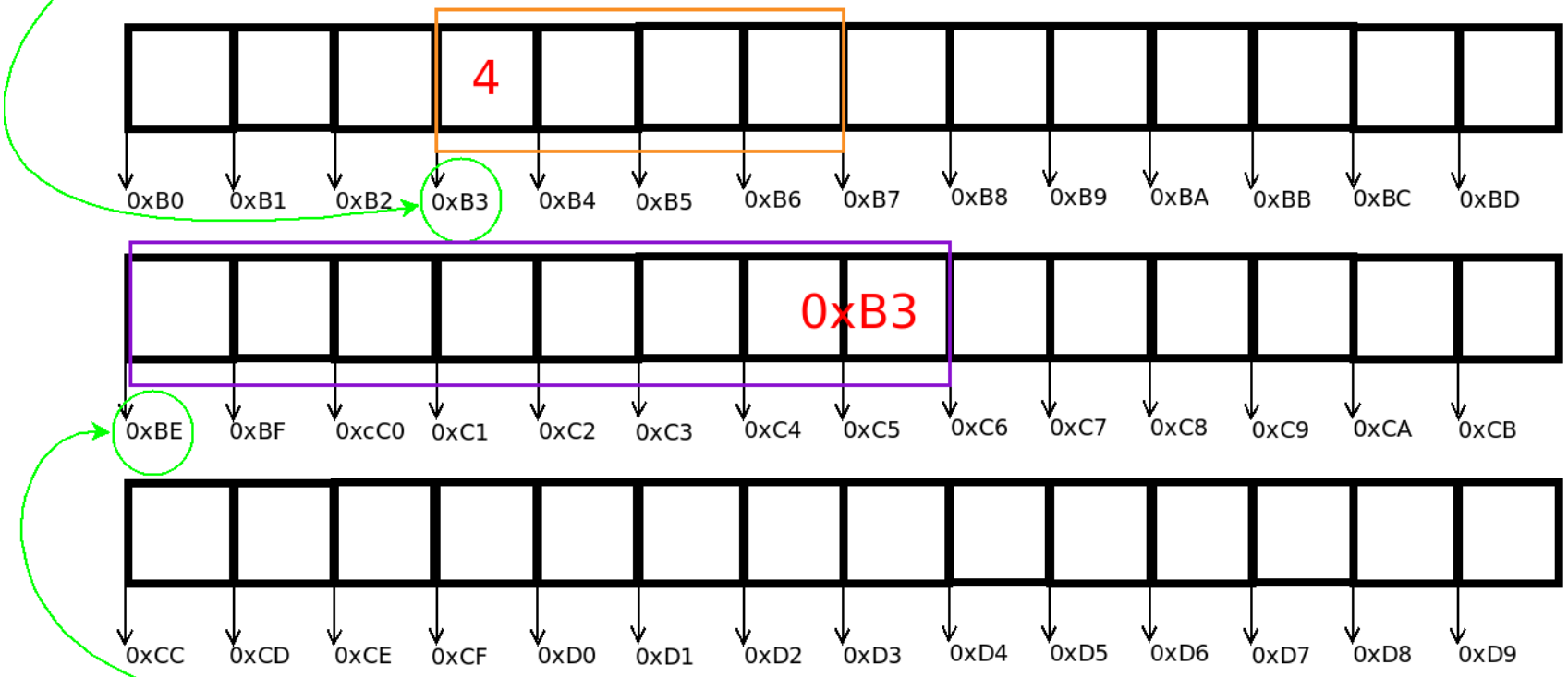
Синтаксис

```
*<имя_переменной>;
```

```
1 int scale = 4;
2 int *p_sc = &scale;
3
4 // Вывод 5 на экран
5 std::cout << *p_sc;
6
7 int rate = (*p_sc) + 15;
8 // изменяем значение scale
9 *p_sc = 15;
```

Указатели

```
int scale = 4;
&scale;
```




```
int *p_sc = &scale;
std::cout << *p_sc;
*p_sc = 15;
```

Операции с указателями: **разыменование**

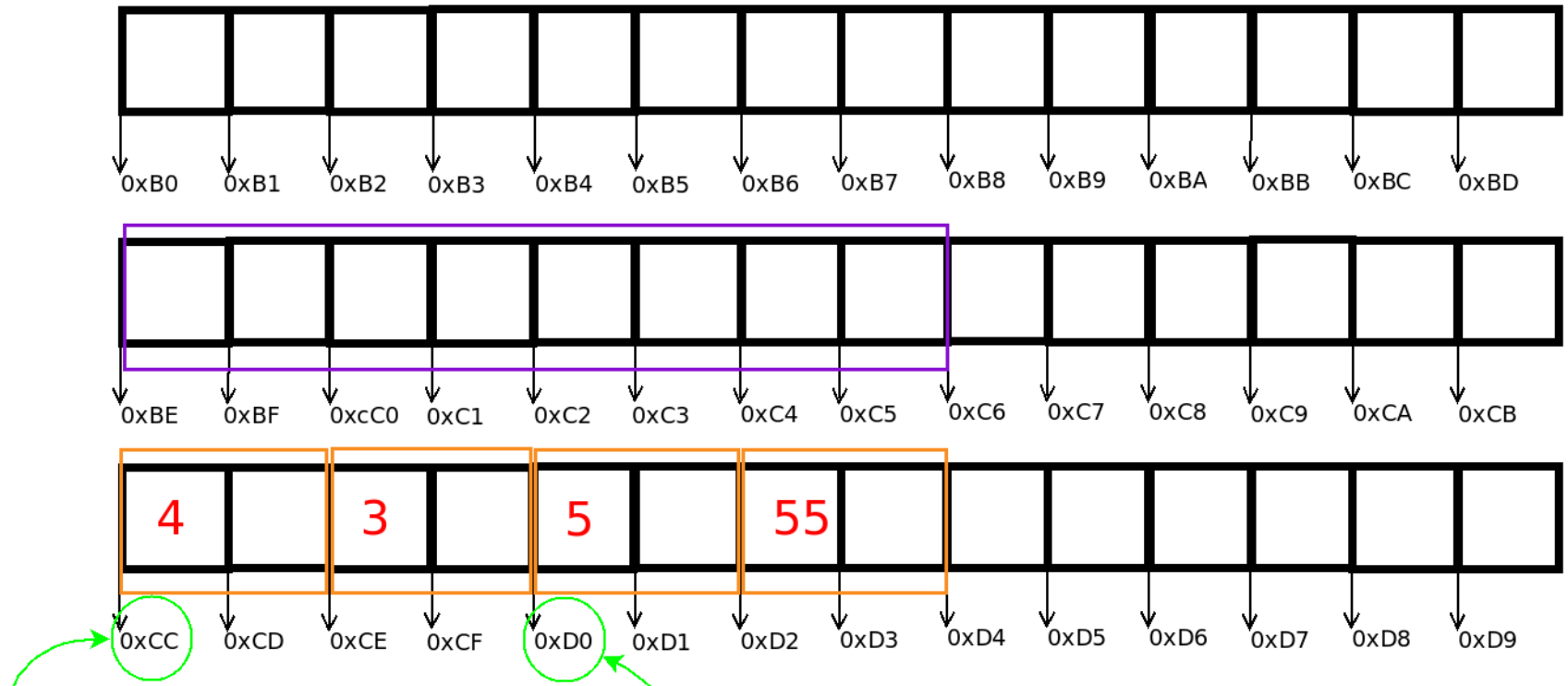
Так никогда не делать!

```
1 char *p_ch;
2 std::cout << *p_ch;
```

Операции с указателями: **разыменование**

```
1  // Как было в C
2  void c_swap(int *i1, int *i2)
3  {
4    int tmp = *i1;
5    *i1 = *i2;
6    *i2 = tmp;
7  }
8
9  // Альтернатива в C++
10 void cpp_swap(int& i1, int& i2)
11 {
12   int tmp = i1;
13   i1 = i2;
14   i2 = tmp;
15 }
```

Указатели и массивы




```
short vec[4] = {4, 3, 5, 55};

&vec[0];

&vec[2];
```

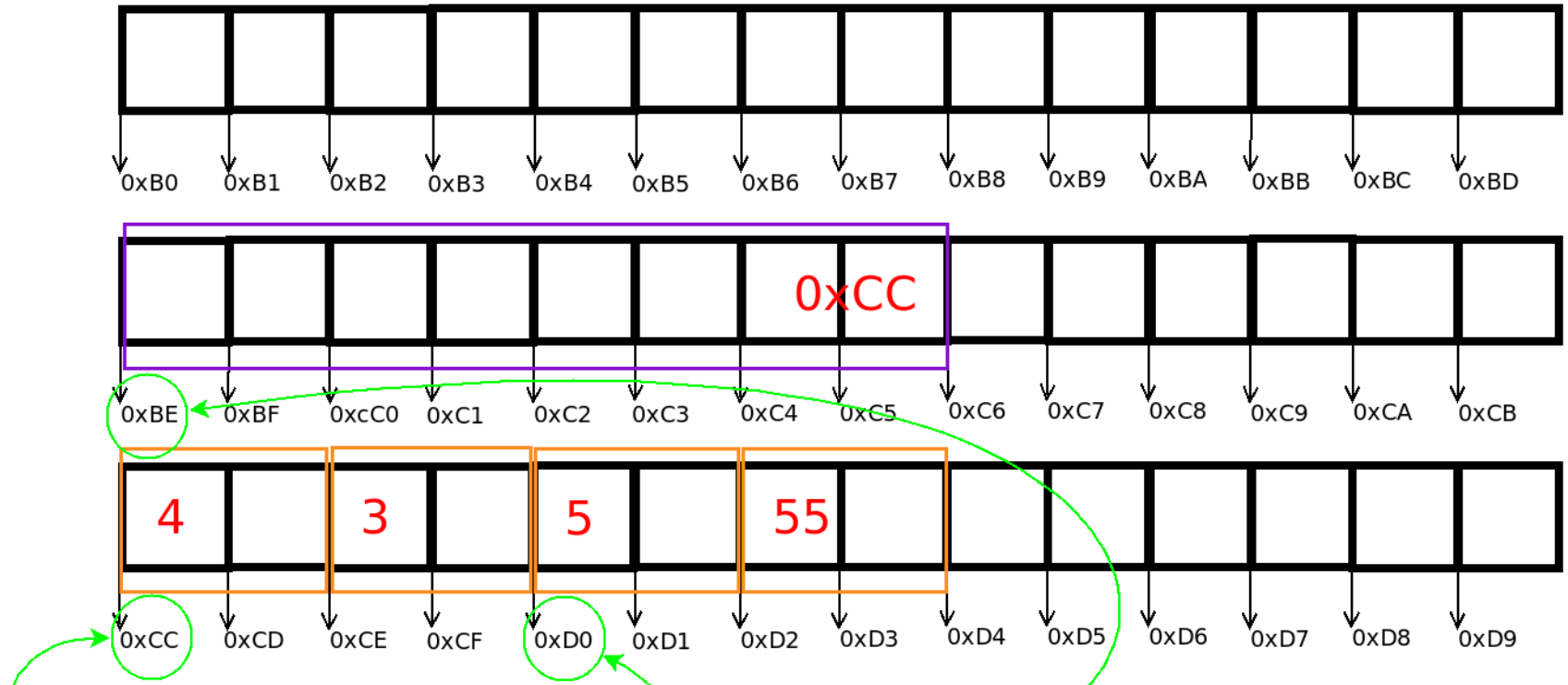
Указатели и массивы

Связь указателей и массивов

- Имя переменной-массива (выше - **vec**) является указателем на его первый элемент
- Массивы передаются в функцию как указатели
- Переменной массива **нельзя** присвоить никакой другой адрес

```
1 void print_array(short* arr, size_t count);
2 ...
3 short vec[4] = {4, 3, 5, 55};
4
5 print_array(vec, 4);
```

Указатели и массивы



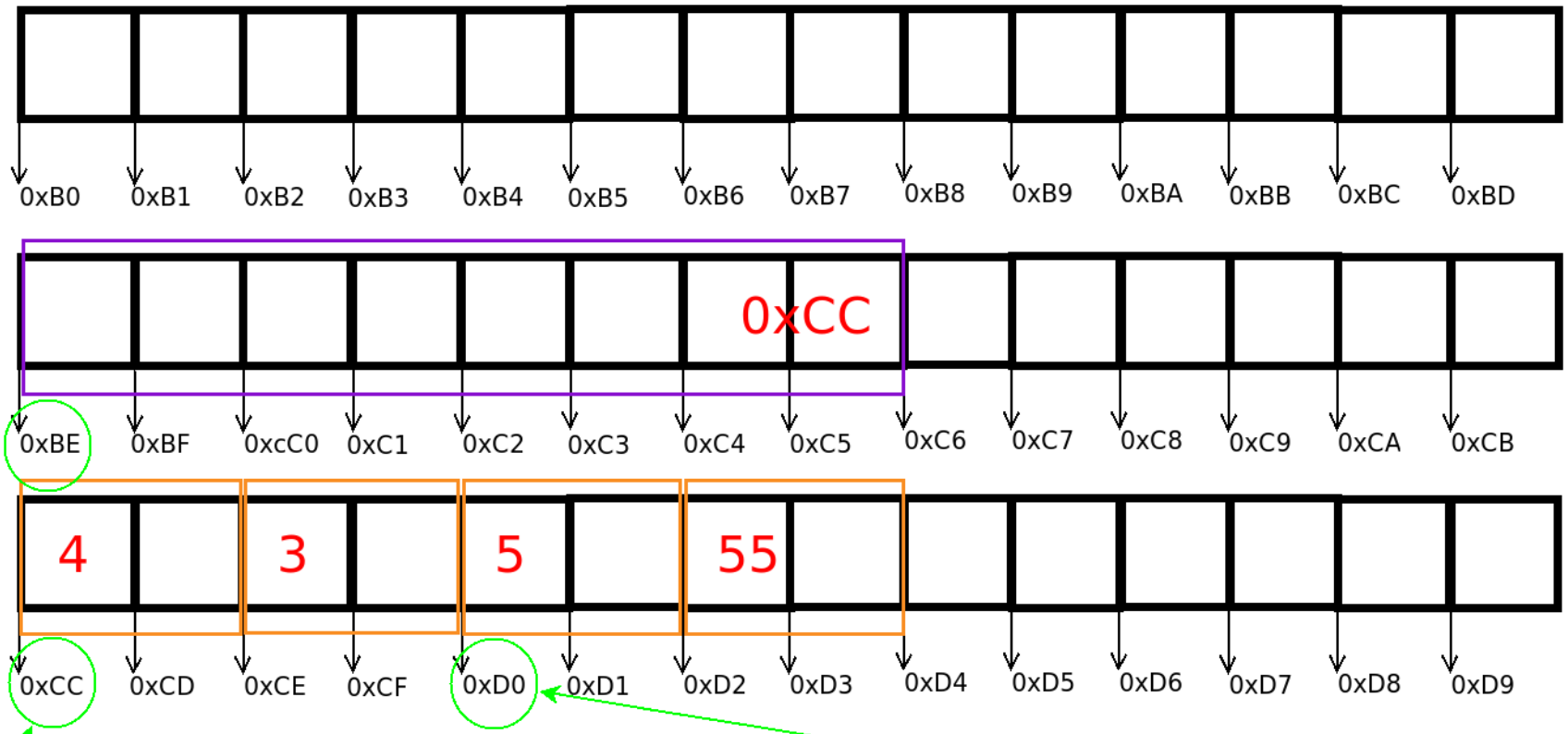

```
short vec[4] = {4, 3, 5, 55};

&vec[0];

&vec[2];
```

```
short *p_vec = vec;
std::cout << *p_vec;
```

Операции с указателями: **сложение с целым числом**: результатом операции прибавления целого числа **n** к указателю является новый указатель, значение которого смещено на $n * \text{sizeof}(< \text{type} >)$ байт.

```
1 short vec[4] = {4, 3, 5, 55};
2 short p_vec = vec;
3
4 // Печатаем 5
5 std::cout << *(p_vec + 2);
```

Указатели и массивы




```
short vec[4] = {4, 3, 5, 55};

short *p_vec = vec;
p_vec;
p_vec + 2;
```

Операции с указателями: **сложение с целым числом**.

```
1 short vec[4] = {4, 3, 5, 55};
2 short *p_vec = vec;
3
4 p_vec++;
5 p_vec += 1;
6 p_vec -= 2;
```

Операции с указателями: **индексация**

```
1 short vec[4] = {4, 3, 5, 55};
2 short *p_vec = vec;
3
4 if (p_vec[2] == *(p_vec + 2)) {
5   std::cout << "Значения равны\n";
6 }
7
8 std::cout << *(vec + 3) << '\n';
```

Указатели и массивы

Операции с указателями: **вычитание однотипных указателей**: результатом операции является целое число (как положительное, так и отрицательное), показывающее количество блоков памяти между двумя указателями. Под блоком памяти понимается размер типа данных указателя.

```
1 short vec[4] = {4, 3, 5, 55};
2 short *p1 = vec, *p2 = &vec[3];
3
4 int diff = p2 - p1;
5 std::cout << diff << '\n'; // Печатает 3
6
7 diff = p1 - p2;
8 std::cout << diff << '\n'; // Печатает -3
```

Указатели

Указатель на тип **void**.

1. Указателю на **void** может быть присвоено значение любого другого указателя
2. Указатель на **void** может быть присвоен любой другой указатель **только с использованием явного приведения типа**.
3. Для указателей на **void** запрещены операции **разыменования**, **индексации** и **вычитания указателей**.

```
1 short vec[4] = {4, 3, 5, 55};
2 short *p1 = vec, *p2;
3
4 void *pv1 = p1;
5 p2 = (short *) pv1;
```

Указатели и динамическая память

Зачем нужно?

Указатели и динамическая память

Оператор **new** - запрос блока динамической памяти у ОС.

Выделение памяти под одно значение

```
(1) <тип_данных>* new <тип_данных>;
(2) <тип_данных>* new (nothrow) <тип_данных>;
```

```
1 int *p1;
2 p1 = new (nothrow) int;
3
4 if (p1 != nullptr) {
5   *p1 = 89;
6 }
```

Оператор **new[]**

Выделение памяти под массив

```
<тип_данных>* new <тип_данных>[<размер>];
<тип_данных>* new (nothrow) <тип_данных>[<размер>];
```

```
1 int *p1, *p2;
2 p1 = new (nothrow) int[10];
3
4 if (p1 != nullptr) {
5   p1[0] = 19;
6 }
7
8 int count = 6;
9 p2 = new (nothrow) int[count];
```

Оператор **new[]**

Инициализация массива нулями

```
<тип_данных>* new <тип_данных>[<размер>]();
<тип_данных>* new (nothrow) <тип_данных>[<размер>]();
```

```
1  int *p1;
2  p1 = new (nothrow) int[10]();
3
4  if (p1 != nullptr) {
5    bool is_zero = p1[0] == 0;
6    // is_zero здесь равен true
7  }
8
9  int count = 6;
10 double *p2;
11 p2 = new (nothrow) double[count]();
```

Указатели и динамическая память

Оператор **delete** - возвращение блока динамической памяти обратно ОС.

Возращение памяти, веделенной под одно значение

```
delete <указатель>;
```

```
1 int *p1;
2 p1 = new (nothrow) int;
3
4 if (p1 != nullptr) {
5   *p1 = 89;
6   // ...
7   delete p1;
8 }
```

Оператор **delete[]**

Возращение памяти, веделенной под массив

```
delete[] <указатель>;
```

```
1 int *p1;
2 p1 = new (nothrow) int[10];
3
4 if (p1 != nullptr) {
5   p1[2] = 89;
6   // ...
7   delete[] p1;
8 }
```

Пример - заполнение массива квадратом индекса

```
1  int *dyn_arr;
2  size_t count;
3  std::cout << "Введите размер: ";
4  std::cin >> count
5
6  dyn_arr = new (nothrow) int[count];
7  if (p1 != nullptr) {
8    for (size_t i = 0; i < count; ++i) {
9      dyn_arr[i] = i * i;
10   }
11
12   delete[] dyn_arr;
13 }
```

Пример - как создать утечку памяти

```
1  double get_complex_average(size_t count)
2  {
3    if (count == 0) {
4      return 0.0;
5    }
6
7    double *arr = new double[count], average;
8    // Сложная обработка массива
9
10   return average;
11 }
12
13 get_complex_average(5);
14 get_complex_average(10);
15 get_complex_average(1500);
```

Пример - как устранить утечку памяти

```
1  double get_complex_average(size_t count)
2  {
3    if (count == 0) {
4      return 0.0;
5    }
6
7    double *arr = new double[count], average;
8    // Сложная обработка массива
9
10   delete[] arr;
11   return average;
12 }
13
14 get_complex_average(10000);
15 get_complex_average(1500);
```

Пример - многомерные указатели

```
1  int **matrix;
2  size_t rows, cols;
3
4  std::cin >> rows >> cols;
5
6  matrix = new int*[rows];
7  for (size_t i = 0; i < rows; ++i) {
8    matrix[i] = new int[cols];
9  }
10
11 //... см. следующий слайд
```

Пример - многомерные указатели

```
1  //... см. предыдущий слайд
2  for (size_ti = 0; i < rows; ++i) {
3    for (size_t j = 0; j < cols; ++j) {
4      matrix[i][j] = i + j;
5    }
6  }
7
8  for (size_t i = 0; i < rows; ++i) {
9    delete[] matrix[i];
10 }
11 delete[] matrix;
```