Лекция II

21 сентября 2018

Прежде, чем идти дальше

Любая переменная может быть задана **константной** (неизменяемой) с помощью ключевого слова **const**.

```
1 const double LIGHT_SPEED = 2.99792458e8;
2 // Из константной переменной можно прочитать ↩
     значение
3 print("Скорость света: ", LIGHT_SPEED, "\n");
4 // но нельзя поменять:
5 // LIGHT_SPEED = 3e8; // Ошибка компиляции
6
7 const size_t PARTS_COUNT = 10;
```

Совет

На название констант сам язык C++ не накладывает никаких ограничений, то есть может быть использован любой уникальный идентификатор, но по общим практикам рекомендуется использовать *верхний регистр* для их именования.

Функции в C++. Определение

Функция в C++ - обособленный набор инструкций, пригодный для повторного использования, который может принимать *произвольное количество* значений и возвращать строго *одно* значение.
Синтаксис определения функции:

```
[...] <тип_возвращаемого_значения>
      <имя_функции> ( <список_параметров> )
{
  [инструкции;]
  return <значение>;
}
```

, где **список_параметров** - это набор пар *тип данных* и *символьное обозначение* каждого параметра, перечисляемых через запятую. Троеточия - специальные указания компилятору, по необходимости будут рассмотрены на практике.

Функции в C++. Пример

$$n! = 1 * 2 * 3 * 4 ... * n$$

Функции в C++. Пример

$$n! = 1 * 2 * 3 * 4 ... * n$$

```
1  double factorial(unsigned n)
2  {
3    double result = 1.0;
4
5    for (unsigned i = 2; i <= n; ++i) {
6      result *= i;
7    }
8
9    return result;
10 }
11
12 ...
13 double result = factorial(8);
14 print("Факториал 8 равен: ", result);
```

Функции в C++. Множественный return

Число операторов возврата **return** внутри функции - неограничено.

```
1  double factorial(unsigned n)
2  {
3    if ( n < 2) {
4      return 1.0;
5    }
6
7    double result = 1.0;
8
9    for (unsigned i = 2; i <= n; ++i) {
10     result *= i;
11   }
12
13   return result;
14 }
```

Функции в C++. Пример

Возведение в степень:

$$\text{number}^n, n - \text{целое}$$

Функции в C++. Пример

Возведение в степень:

$$\text{number}^n, n - \text{целое}$$

```
1  double degree_nth(double num, int n)
2  {
3    double result = 1.0;
4    // Берём модуль от числа n
5    unsigned degree = std::abs(n);
6
7    while ( degree > 0 ) {
8      result *= num;
9      --degree;
10   }
11
12   return (n < 0) ? (1.0 / result) : result;
13 }
```

Функции в C++. Пример

```
1 double degree_nth(double num, int n)
2 { ... }
3
4 double rate  = 4.5;
5 int    scale = -4;
6
7 double rate_in_sc = degree_nth(rate, scale);
8 print("rate^scale = ", rate_in_sc, "\n");
9 print("8.85^-3 = ", degree_nth(8.85, -3), "\n");
```

Прежде, чем идти дальше

Адрес переменной

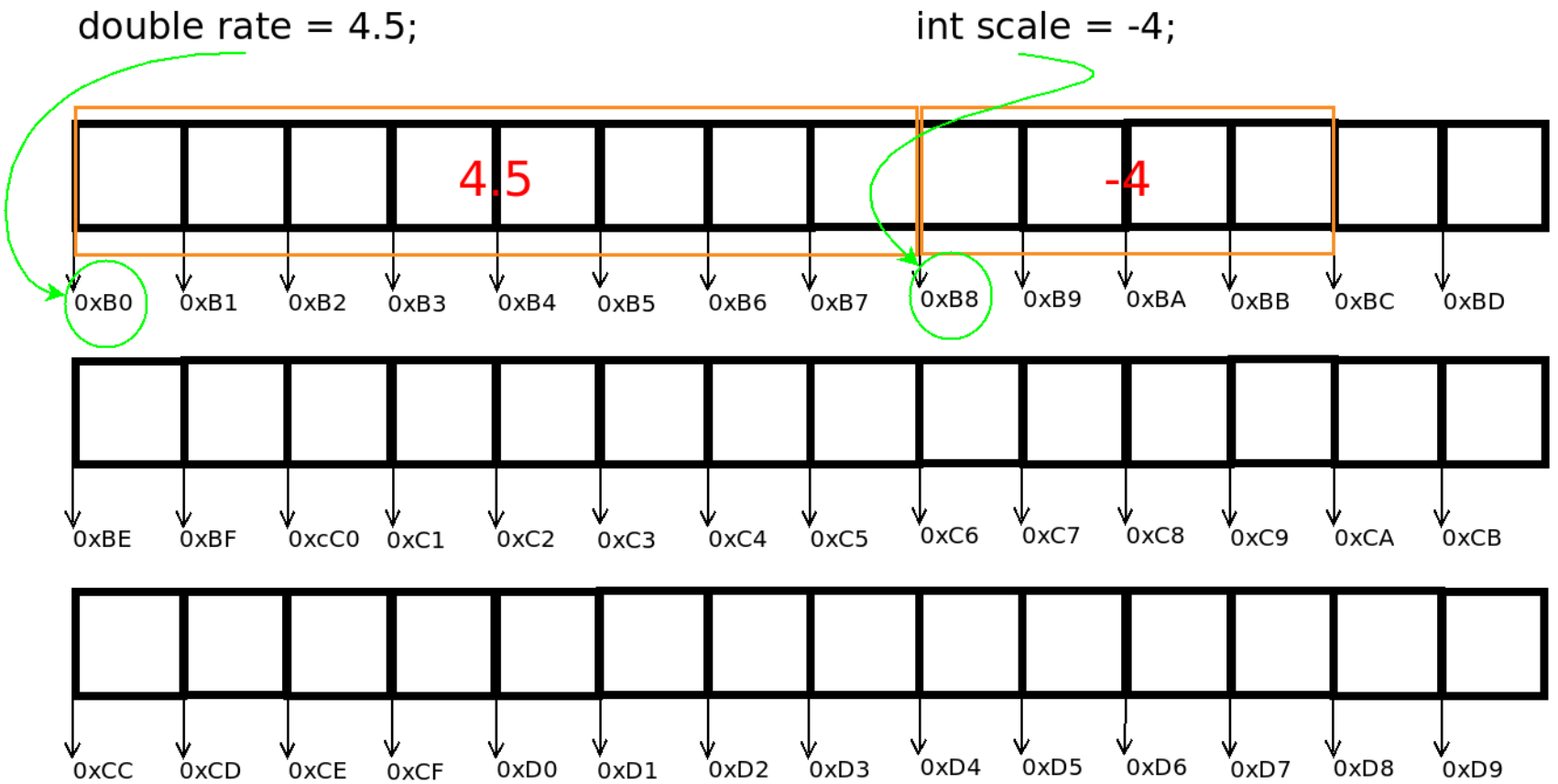
Каждая переменная любого типа связана с блоком в памяти, который состоит из какого-либо количества байт. Каждый байт в модели памяти языка C++ пронумерован - ему присвоен порядковый номер: 0, 1, 2, 3, ... Номер первого байта блока называется **адресом переменной**. Адреса обычно приводят в *шестнадцатиричном* виде.

Функции в C++. Передача аргументов по значению

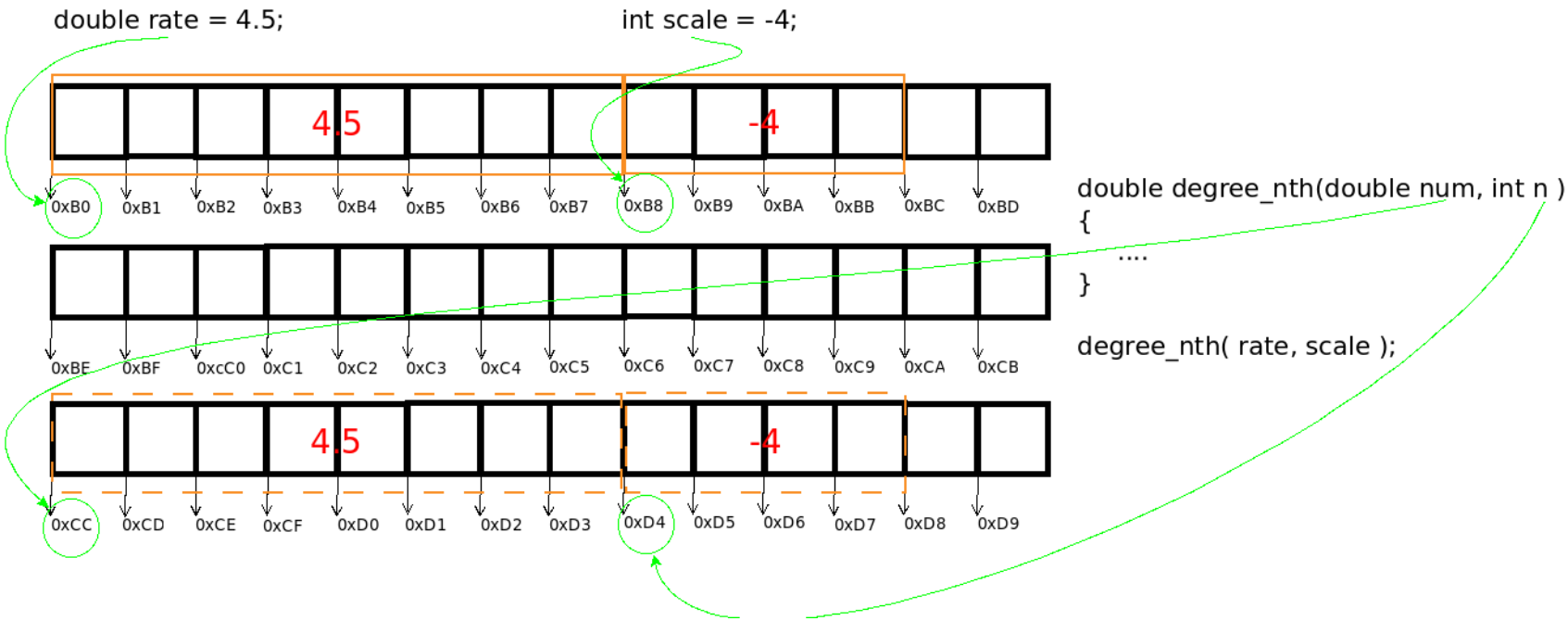
```
1 double degree_nth(double num, int n)
2 { ... }
3
4 double rate  = 4.5;
5 int    scale = -4;
6
7 double rate_in_sc = degree_nth(rate, scale);
```

Значения переменных **rate** и **scale** при вызове функции **degree_nth** называют её **аргументами**. В примере эти аргументы передаются в тело функции **по значению**.

Функции в C++. Передача аргументов по значению



Функции в C++. Передача аргументов по значению

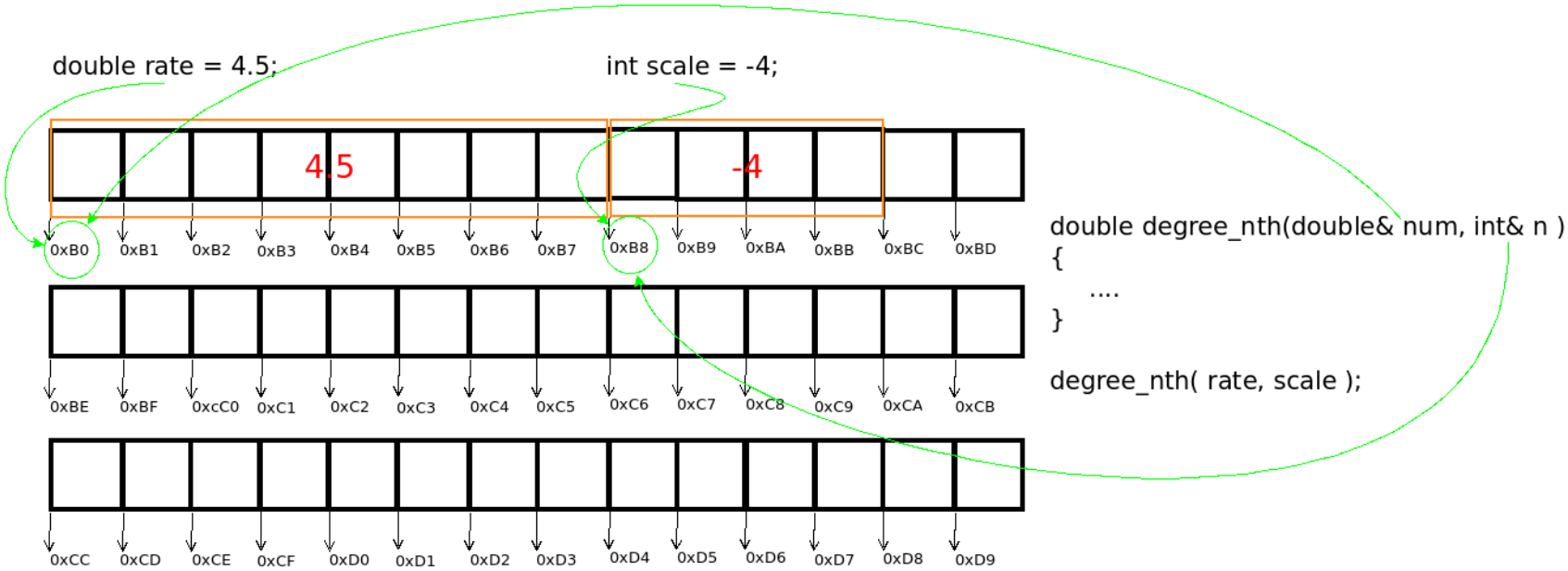


Функции в C++. Передача аргументов по ссылке

Есть второй способ передачи аргументов - **по ссылке**. Для этого в списке параметров *после* типа указывается знак «амперсанда» (**&**)

```
1 double degree_nth(double& num, int& n)
2 { ... }
3 ...
4
5 double rate = 4.5;
6 int scale = -4;
7
8 double rate_in_sc = degree_nth(rate, scale);
```

Функции в C++. Передача аргументов по ссылке



Важный факт. Если функция принимает аргументы только по ссылке, невозможно в неё передать временные значения. То есть, невозможен вызов вида:

```
9 degree_nth( 3.5, 5 );
```

Функции в C++. Константные параметры

Параметры функции можно сделать **неизменяемыми (константными)**. В случае ссылочных параметров это позволяет передавать в функцию временные значения.

```
1  double degree_nth(const double& num,
2                    const int& n)
3  { ... }
4
5  ...
6
7  double rate = 4.5;
8  int scale = -4;
9
10 double rate_in_scale = degree_nth( rate, scale );
11 // Теперь можно и так:
12 print("3.3^2 = ", degree_nth(3.3, 2), "\n");
```

Функции в C++. Пример

Пример: запросить у пользователя n целых чисел, найти максимальное и каким по счёту оно было введено

```
1  int find_max_with_pos(unsigned count, unsigned& place)
2  {
3      int current, max_num;
4
5      for (unsigned i = 1; i <= count; ++i) {
6          print("Введите ", i, " число: ";
7          get_value(current);
8
9          if ( (i > 1) and (current > max_num) ) {
10           max_num = current; place = i;
11         } else {
12           max_num = current; place = 1;
13         }
14     }
15     return max_num;
16 }
17 ...
18 unsigned position;
19 int max_elem = find_max_with_pos(5, position);
20 print("Максимальное число ", max_elem, " найдено на ",
21       position, " месте\n");
```

Функции в C++. Параметры по умолчанию

```
1  double degree_nth(double num, int n = 3)
2  { ... }
3
4  ...
5
6  double rate = 4.5;
7
8  // Внутри функции *degree_nth* n равен 2
9  print(degree_nth( rate, 2 ), "\n");
10 // Выведет на экран: 20.25
11
12 // Внутри функции n равен 3
13 print(degree_nth( rate ), "\n");
14 // Выведет на экран: 91.125
```

Функции в C++. Параметры по умолчанию

1. Параметры со значением по умолчанию **должны** идти в конце списка.

```
1 double some_fun1(double r1, double r2,
2                  double r3, int n1 = 2,
3                  int n2 = 3)
4 { ... }
```

2. Такие аргументы не могут быть **ссылочными изменяемыми** (константными ссылочными - могут).

```
1 // Ошибка компиляции
2 double some_fun2(int& n1 = 2)
3 { ... }
4
5 // Всё хорошо, компилятор доволен
6 double some_fun3(const int& n1 = 2)
7 { ... }
```

Функции в C++. Не хочу ничего возвращать

Возможно определять функции не требующие возврата значений из них. Для такого случая предусмотрен специальный тип данных - **void**

```
1  void print_only_even_number(const int num)
2  {
3    if ( (num % 2) != 0 ) {
4      return;
5    }
6
7    print(num);
8  }
9  // Переменных типа void создавать нельзя
10 // void var_of_void;
```

Функции в C++. Перегрузка

- Каждая функция должна быть уникальна: определена в программе единожды
- Уникальность функции в языках программирования определяется её **сигнатурой** - набором отличительных признаков
- В C++ сигнатура функции определяется:
  1. Именем (идентификатором)
  2. Количеством аргументов (термин *арность*)
  3. Типами аргументов и их порядком

Функции в C++. Перегрузка

```
1  int max(const int first, const int second)
2  {
3    print("Работает max(int, int)\n";
4    return (first > second) ? first : second;
5  }
6
7  double max(const double left, const double right)
8  {
9    print("Работает max(double, double)\n";
10   const double EPS = 5e-5; // 5 * 10^(-5)
11
12   if ( std::abs(left - right) > EPS ) {
13     return left;
14   } else {
15     return right
16   }
17 }
18
19 ...
20 print(max(56, 78), "\n");
21 print(max(3.4, 3.400007), "\n");
```

Пример из стандартной библиотеки C++

```
1  /*
2    Получение модуля целого или
3    действительного числа
4  */
5  #include <cmath>
6  #include <cstdlib>
7
8  ...
9  std::abs( 56 );
10 std::abs( -8.888 );
```

Функции в C++. Объявление и определение

Аналогично переменным, для функций существуют понятия **объявления** и **определения**. Определение - это все примеры выше, а под **объявлением** понимается указание *типа возвращаемого значения, названия функции и перечисление типов всех аргументов*. Причём, давать имена аргументам - не обязательно

```
1  #include "ffhelpers.h"
2
3  double my_rand(unsigned); // Объявление!
4
5  int main()
6  {
7    print("Случайное число: ", my_rand(5), "\n");
8  }
9
10 double my_rand(unsigned seed) // Определение!
11 { return 1234.5688 / seed; }
```

Резюме по функциям

- Количество параметров выбираем произвольно, но вернуть можно только одно конкретное значение какого-нибудь типа
- Два способа передачи аргументов - **по значению** и **по ссылке**. В обоих случаях параметры можно сделать *константными*
- Части параметров можно присваивать значения по умолчанию
- Специальный тип **void** - когда из функции не нужно возращать никакого значения
- Только название функции не уникально - помним о перегрузке
- Определение функции не может быть помещено в тело другой функции

Прежде, чем идти дальше

Область видимости идентификатора

Место в файле с исходным кодом, где идентификатор доступен для операций (использование значения в случае переменной, вызов - в случае функции). В языке C++ различают две области - **глобальная** и **локальная**.

- Все функции имеют **глобальную** область видимости
- Переменная имеет **локальную** область видимости, если её объявление или определение находится внутри пары фигурных скобок {}
- Иначе переменная получает **глобальную** область видимости
- Локальная переменная видна во всех вложенных областях
- Идентификатор локальной переменной скрывает свои предыдущие определения

Прежде, чем идти дальше

Область видимости идентификатора

```
1  // Переменная с глобальной областью видимости
2  const double PI = 3.14159265358;
3
4  void do_something(double x)
5  {
6    // Локальная область видимости
7    double rate = 0.5;
8
9    if (x > 10.0) {
10     // Ещё одна. PI и rate тут доступны
11     double num = PI * x * rate;
12     print("Результат: ", num, "\n");
13   }
14
15   // Доступа к num уже нет:
16   // num += 3.05;
17   rate *= 9.5;
18 }
```

Прежде, чем идти дальше

Время жизни переменной

- Переменные с локальной областью видимости автоматически удаляются по достижении конца области видимости
- Глобальные переменные существуют до тех пор, пока выполняется программа

Составные типы данных

Материальная точка: три координаты да масса.
Найдём расстояние между двумя точками.

```
1  #include <cmath>
2
3  double get_distance(int x1, int y1, int z1,
4                      int x2, int y2, int z2)
5  {
6    double dx2 = std::pow(x2 - x1, 2),
7           dy2 = std::pow(y2 - y1, 2),
8           dz2 = std::pow(z2 - z1, 2);
9
10   return std::sqrt(dx2 + dy2 + dz2);
11 }
```

Структуры

Структура - составной пользовательский тип данных, состоящий из элементов других типов данных. Каждый элемент называется **полем** структуры.

```
1  struct MaterialPoint
2  {
3    int x, y, z;
4    double mass;
5  };
6
7  ...
8
9  MaterialPoint p1 = {1, 4, 5, 4.55}, p2;
10 print("Масса точки: ", p1.mass, "\n");
11
12 p2.x = p2.y = p2.z = 5;
13 int some_val = p1.x * 3 - p2.z;
```

```
1  double get_distance(MaterialPoint p1,
2                      MaterialPoint p2)
3  {
4    double dx2 = std::pow( p2.x - p1.x, 2),
5           dy2 = std::pow( p2.y - p1.y, 2),
6           dz2 = std::pow( p2.z - p1.z, 2);
7
8    return std::sqrt( dx2 + dy2 + dz2 );
9  }
10
11 ...
12 MaterialPoint p1 = {1, 2, 3, 5.5},
13                 p2 = {5, -5, -3, 3.2};
14 print("Расстояние: ", get_distance(p1, p2));
```


Why
Mr. Anderson,why?
quickmeme.com

Структуры

```
1  struct MaterialPoint
2  {
3    int x, y, z;
4    double mass;
5
6    double dist_from_base()
7    { return std::sqrt(x*x + y*y + z*z); }
8  };
9
10 ...
11
12 MaterialPoint p1 = {6, 4, 5, 8.55};
13 print("Масса точки: ", p1.mass, "\n)";
14 print("Длина радиус-вектора: ",
15       p1.dist_from_base() );
```