Junior C++ developer

Лекция 2

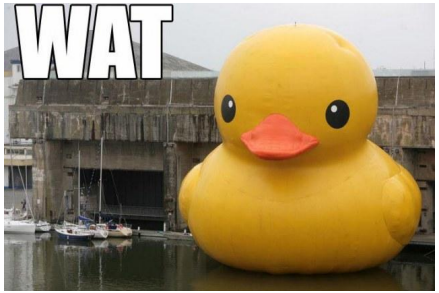
- Функции
- Контейнеры (vector, string)
- enum/enum class

Передача по значениею

При передачи параметров по значению передают **только значения** объектов, но не сами объекты!

```
void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}
int main() {
  int a = 3;
  int b = 7;
  cout << a << ", " << b;
  swap(a, b);
  cout << a << ", " << b;
}
```

Ничего не происходит



Передача по ссылке

При передаче по ссылке передается **сам объект**, просто с другим именем (псевдонимом).

```
void swap(int &x, int &y) {
  int temp = x;
  x = y;
  y = temp;
}

int main() {
  int a = 3;
  int b = 7;
  cout << a << ", " << b;
  swap(a, b);
  cout << a << ", " << b;
}
```

Передача аргументов

```
int DoubleIt(int x)
{
   x = x * 2;
   return x;
}
```

- Функция принимает аргумент *x* по значение
- *x* - внутри функции, это *копия* аргумента *x*
- переменная *x*, которая была передана в функцию не изменилась

```
int DoubleIt(int& x)
{
   x = x * 2;
   return x;
}
```

- Функция принимает аргумент по ссылке
- Функция не создает копий аргумента
- Функция меняет значение аргумента *x*

Практика

Реализовать функцию, которая:

- принимает один аргумент типа int по значению
- вычисляет является ли полученное число простым
- возвращает true в случае если число простое, false в противном случае

Практика

```
bool isPrime(int x)
{
    bool is_prime = true;
    for (int i = 2; i < x; i++)
    {
        int factor = x/i;
        if (factor * i == x)
        {
            cout << "Find factor " << factor << endl;
            is_prime = false;
            break;
        }
    }
    return is_prime;
}
```

Реализовать функцию, которая:

- принимает один аргумент типа int по значению
- вычисляет является ли полученное число простым
- возвращает true в случае если число простое, false в противном случае

Практика

```
#include <iostream>

using std::cout;
using std::endl;

void changeValue(int x)
{
   x = 15;
}

int main()
{
   int value = 8;
   cout << value << endl;
   changeValue(value);
   cout << value << endl;
   return 0;
}
```

Что будет выведено на экран?

Передача аргументов по ссылке

- Когда необходимо изменить передаваемое значение
- Когда передаваемое значение большого размера. Копирование больших объектов затратная операция, поэтому их лучше передавать по ссылке

Последовательный контейнеры (Sequential Containers)

- Предоставляют последовательный доступ к элементам
- Позволяют обращаться к элементам по индексу.
- Позволяют контролировать порядок элементов.

  Примеры:

  - *vector*
  - *string*
  - *array*
  - *deque*

std::string

Дано:

```
string str = "abcdefg";
```

Задача:

вывести все символы строки через пробел

```
int main()
{
    string str = "abcdefg";
    for (auto c : str)
    {
        cout << c << " ";
    }
    return 0;
}
```

std::vector

Дано:

```
vector<int> numbers = {1, 2, 3, 4, 5};
```

Задача:

вывести все элементы вектора через пробел

std::vector

Дано:

```
vector<string> numbers = {"1", "2", "3", "4", "5"};
```

Задача:

вывести все элементы вектора через пробел

std::vector

Дано:

```
vector<int> numbers = {1, 2, 3, 4, 5, 1, 3, 1};
```

Задача:

посчитать количество 1 в векторе

Наивная реализация:

```
int main()
{
   vector<int> numbers = {1, 2, 3, 4, 5, 1, 3, 1};
   int count = 0;
   for (auto x : numbers)
   {
     if (x == 1)
     {
        count += 1;
     }
   }
   cout << count;
   return 0;
}
```

Посчитать количество 1 в векторе

Реализация с помощью алгоритмов:

Посчитать количество 1 в векторе

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> numbers = {1, 2, 3, 4, 5, 1, 3, 1};
    int count_elements = count(numbers.begin(), numbers.end(), 1);
    cout << count_elements;
    return 0;
}
```

numbers.begin() - указывает на первый элемент вектора

numbers.end() - указывает на следующий за последним элементом вектора

std::vector

Дано:

```
vector<int> numbers = {1, 2, 3, 4, 5, 1, 3, 1};
```

Задача:

вывести отсортированный вектор

Вывести отсортированный вектор

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> numbers = {1, 2, 3, 4, 5, 1, 3, 1};
    sort(numbers.begin(), numbers.end());
    for (auto x : numbers)
    {
        cout << x << " ";
    }

    return 0;
}
```

Практика

Что будет выведено на экран:

```
string word = "babacaf";
sort(begin(word), end(word));
cout << word << " " << count(begin(word), end(word), 'a');
```

Почему передавать большие объекты в функцию по значению плохая идея

Время работы функции

Для вычисления работы функции можно воспользоваться std::chrono, для этого необходимо:

- подключить заголовок `#include <chrono>`
- объявить использование пространства имен `using namespace std::chrono;`
- зафиксировать момент времени перед вызовом функции и сразу после

```
auto start = steady_clock::now();
fooo();
auto finish = steady_clock::now();
```

- Подсчитать промежуток времени между start и finish

```
auto ms = duration_cast<milliseconds>(finish - start).count();
```

```cpp
#include <iostream>
#include <vector>
#include <limits>
using namespace std;
struct Person
{
    string f_name;
    string l_name;
    int age;
};
vector<Person> createPersonList(int list_size)
{
    vector<Person> list;
    for (int i = 0; i < list_size; i++)
    {
        Person person;
        person.age = i;
        list.push_back(person);
    }

    return list;
}
```

```cpp
int main()
{
    auto start = steady_clock::now();
    vector<Person> my_list = createPersonList(40'000'000);
    auto finish = steady_clock::now();
    auto ms = duration_cast<milliseconds>(finish - start).count();
    cout << ms << " ms" << endl;
    return 0;
}
```

А теперь передадим по ссылке

Реализуем функцию, которая выводит размер списка:

```
void printPersonListSize(vector<Person> &p_list)
{
   cout << "Size " << p_list.size() << endl;
}
```

Запустить и проверить время выполнения

```
int main()
{
    auto start = steady clock::now();
    vector<Person> my list = createPersonList(40'000'000);
    auto finish = steady clock::now();
    auto ms = duration_cast<milliseconds>(finish - start).count();
    cout << ms << " ms" << endl;

    start = steady clock::now();
    printPersonListSize(my list);
    finish = steady clock::now();
    ms = duration cast<milliseconds>(finish - start).count();
    cout << ms << " ms" << endl;

    return 0;
}
```

Что будет выведено на экран?

```
int main()
{
    printPersonListSize(createPersonList(40'000'000));
    return 0;
}
```

```
/cygdrive/f/jundev_winter19/L2 Functions/main.cpp:36:41: error: cannot bind non-const lvalue reference of type 'std::vector<Person>&' to an rvalue of type 'std::vector<Person>'
    printPersonListSize(createPersonList(40'000'000));
                        ~~~~~~~~~~~~~~~~^~~~~~~~~~~~
/cygdrive/f/jundev_winter19/L2 Functions/main.cpp:28:6: note:   initializing argument 1 of 'void printPersonListSize(std::vector<Person>&)'
 void printPersonListSize(vector<Person> &p_list)
      ^~~~~~~~~~~~~~~~~~
```

Передача аргументов по константной ссылке

```
void printPersonListSize(const vector<Person> &p_list)
{
   cout << "Size " << p_list.size() << endl;
}
```

Передача аргументов по константной ссылке

- Защищает от случайных изменений аргумента внутри функции
- Позволяет передавать в функцию результат выполнения функции

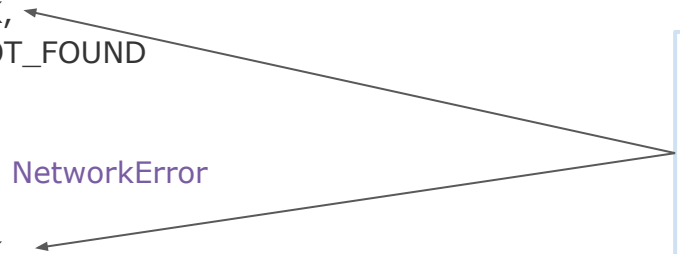
Перечисления - enum

Именованный набор констант

Имена констант должны быть уникальными в пределах пространства имен

```
enum FileError
{
   OK,
   NOT_FOUND
};

enum NetworkError
{
   OK,
   DISCONNECTED
};
```



Этот код не скомпилируется, так как есть константы с одинаковыми именами

enum class

Тип данных отличный от ***int***

Имена не обязаны быть уникальными

Для доступа к полям используется полное имя

Этот код успешно скомпилируется

```
enum class FileError
{
   OK,
   NOT_FOUND
};

enum class NetworkError
{
   OK,
   DISCONNECTED
};
```

Пример использования:

```
int main()
{
    FileError fe = FileError::OK;
    NetworkError ne = NetworkError::OK;

    return 0;
}
```