Novosibirsk State University
*THE REAL SCIENCE

Программирование на C++ и Python

Лекция 1

C++ быстрый старт

Воробьев Виталий Сергеевич (ИЯФ СО РАН, НГУ)

Новосибирск, 1 сентября 2021

Цели курса

1. Дать начальные навыки разработки на языках *C++* и Python
2. Познакомить с базовыми концепциями программирования
   - Структуры данных и алгоритмы
   - Парадигмы программирования
3. Показать средства для анализа данных с библиотеками SciPy
4. Познакомить с инструментами для совместной разработки программ

За один семестр невозможно стать профессиональным программистом (да и не надо!). После прохождения курса вам будет проще осваивать языки программирования самостоятельно.

Программа курса

Учебная нагрузка

- 8 лекций
- 16 практических занятий по 1.5 пары

Зачет

1. На тройку: набрать по 5 баллов из девяти блоков заданий
2. На четверку: выполнить условие на тройку и набрать 60 или более баллов
3. На пятерку: выполнить условие на тройку и набрать 80 или более баллов

Задания сдаются через сервис github.com.
Процедура описана на сайте курса.

C++

1. Потоки ввода-вывода, строки
2. Контейнеры STL
3. Алгоритмы STL
4. Классы
5. Шаблоны

Python

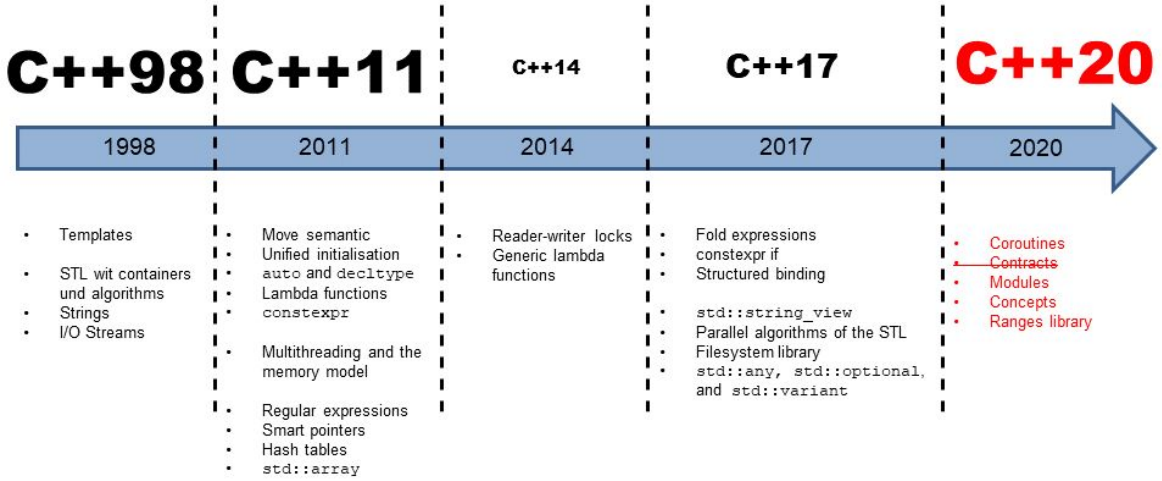
6. Введение в Python
7. Стандартная библиотека Python
8. Вычисления с numpy
9. Построение диаграмм с matplotlib

Ресурсы

1. Сайт курса cpp-python-nsu.inp.nsk.su
   - Учебник: cpp-python-nsu.inp.nsk.su/textbook
   - Задания: cpp-python-nsu.inp.nsk.su/assignments
2. Репозитории с лекциями:
   - github.com/NSU-Programming/lectures2020
   - github.com/NSU-Programming/lectures2021
3. Telegram-группа (t.me/joinchat/dpx594KRwstiNDhi, QR-код)

Зачем изучать *C++*

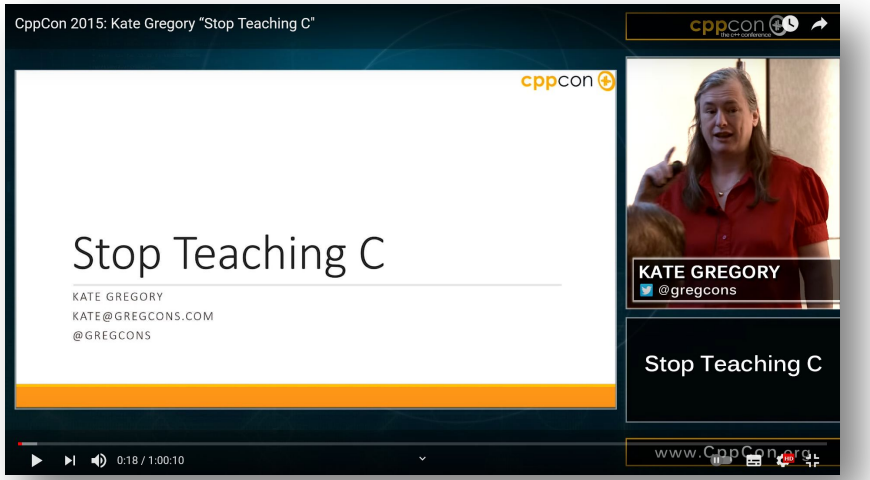
- *C++* быстрый и он развивается
  - **Быстрый**: это основной язык разработки в коммерческих и научных проектах, в которых важна *эффективность* (OS X, MS Windows, Firefox, Chromium, Adobe Photoshop, Tensorflow, и ещё **очень** много чего)
  - **Развивается**: на современном *C++* можно писать ясный и надёжный код

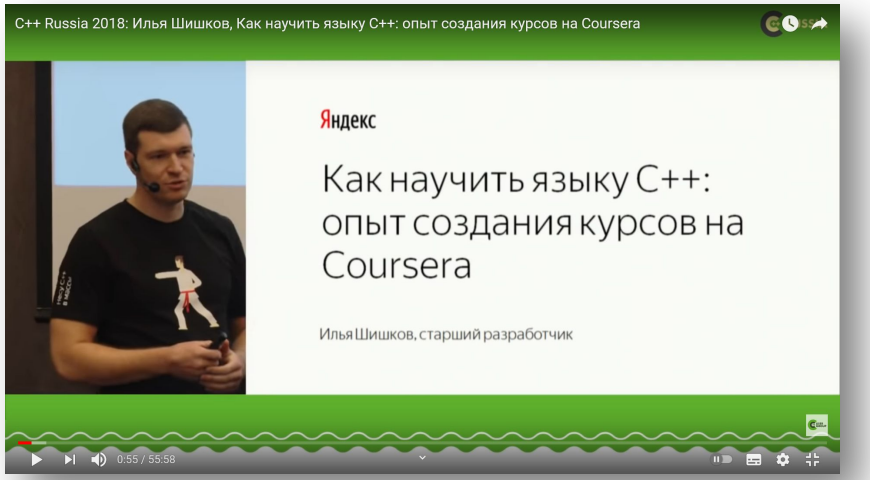


- Выпускнику ФФ полезно иметь представление о написании эффективного кода
- Философия *C++*: **zero cost abstractions**

Как мы будем изучать *C++*

- Как высокоуровневый язык программирования
  - А не как развитие языка Си
- Продвинутые возможности языка (классы и шаблоны, динамическое выделение памяти) будем обсуждать в несколько этапов
  - Сначала научимся использовать стандартную библиотеку
  - Потом научимся использовать продвинутые возможности языка при написании собственных программ
- Предполагаем, что вы знакомы с языком Си





Ресурсы по *C++*

- en.cppreference.com – документация
- isocpp.org – Standard *C++* Foundation
- [Поисковик] + stackoverflow.com
- Coursera
  - Искусство разработки на современном C++
  - …
- Книги
  - Bjarne Stroustrup «The C++ programming language»
  - Scott Meyers «Effective C++»
  - …
- boost.org – расширенный набор библиотек
- Задачи online: hackerrank.com и др.
- …

Bjarne Stroustrup

Scott Meyers

C++ быстрый старт

Hello, student!

Подключаем библиотеки

Поговорим об этом позже

Вывод текста в стандартный поток вывода

Строковая переменная

Ввод из стандартного потока ввода

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
  cout << "What is your name? ";
  string name;
  cin >> name;
  cout << "Hello, " << name << "!" << endl;
  return 0;
}
```

Потоки ввода-вывода

Ввод и вывод в *C++* реализован с помощью *потоков*

1. Стандартные потоки: stdin, stdout, stderr
2. Файловые потоки: fstream, [i/o]fstream
3. Строковые потоки: stringstream, [i/o]stringstream

- Стандартные объекты:
  1. cout – вывод в stdout
  2. cin – ввод из stdin
  3. cerr – вывод в поток ошибок stderr

- Операторы ввода-вывода
  1. `operator<<` – оператор вывода
  2. `operator>>` – оператор ввода
     - Нет необходимости указывать тип данных
     - Можно использовать *цепочки вызовов*

```
<iostream>
<fstream>
<sstream>
<iomanip>
```

Пример: чтение файла

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    const string fname("numbers.txt");
    ifstream ifile(fname, ios::in);
    if (!ifile.good()) {
        cout << "Can't load file "
             << fname << endl;
        return 0;
    }

    int value, sum = 0;
    while (ifile >> value) sum += value;

    cout << "Sum equals " << sum << endl;
    return 0;
}
```

В файл записаны целые числа, разделенные пробелом. Вывести сумму чисел в стандартный поток вывода.

numbers.txt

```
1 1 2 3 5 8 13 21 34 55
89 144 233 377 610 987
1597 2584 4181 6765
```

```
> Sum equals 24475
```

Чтение из файла

- Чтение по символу

```
char c;
while (ifile.get(c)) {
    cout << c;
}
```

- Чтение по строке

```
string line;
while (getline(ifile, line)) {
    cout << line << '\n';
}
```

- Чтение бинарных данных

```
ifstream ifile(fname, ios::binary);
char c;
while (ifile.get(c)) {
    cout << static_cast<int>(c) << ' ';
}
```

- Метод `fstream::eof()` позволяет узнать достигнут ли конец файла

```cpp
#include <iostream>
#include <sstream>
#include <utility>  // std::pair

pair<double, double>
parse(const string& data) {
    istringstream ss(data);
    double lat, lon;
    ss.ignore(1);  // skip '('
    ss >> lat;
    ss.ignore(2);  // skip ", "
    ss >> lon;
    return {lat, lon};
}

int main() {
    auto [lat1, lon1] = parse("(54.847830, 83.094392)");
    auto [lat2, lon2] = parse("(54.835815, 83.101360)");
    cout << dist(lat1, lon1, lat2, lon2)
         << " km" << endl;
}
```

Пример: строковый поток

Вэб-сервер вернул географические координаты двух точек в виде строк. Найти расстояние между точками

Input

```
"(54.847830, 83.094392)"
"(54.835815, 83.101360)"
```

```
> 1.40854 km
```

Отступление: pair и tuple

```
<utility>
<tuple>
```

- Объект `pair` может хранить пару объектов любых типов

```
auto point = make_pair(lat, lon);
cout << point.first << ", " << point.second;
```

- Объект `tuple` может хранить фиксированное количество гетерогенных объектов

```
const string name = "Vitaly";
int age = 32;
float score = 1.05;
auto record = tie(name, age, score);
cout << get<0>(record) << ", "
     << get<1>(record) << ", "
     << get<2>(record) << endl;
```

Отступление: `pair` и `tuple`

```
<utility>
<tuple>
```

```cpp
using Record = tuple<string, int, float>;

Record get_record() {
    const string name = "Vitaly";
    int age = 32;
    float score = 1.05;
    return {name, age, score};
}

int main() {
    auto [name, age, score] = get_record();
    cout << name << ", " << age << ", "
         << score << endl;
    return 0;
}
```

- `pair` и `tuple` удобно использовать для возвращения из функции несколько объектов
- Использование *псевдонимов* позволяет сделать код проще для чтения

Передача аргументов I

```cpp
#include <string>
#include <iostream>

using namespace std;

void add_exclamation(string& str) {
    str.push_back('!');
}

int main() {
    string line("Hello, world");
    add_exclamation(line);
    cout << line << endl;
    return 0;
}
```

- В функцию по умолчанию передаются копии параметров
- А давайте вернем новую строку!
  - Для добавления одного символа мы скопировали строку два раза…
- Передача строки *по ссылке* – то что нужно в этой ситуации

```
> Hello, world!
```

Передача аргументов II

- Передача аргумента по ссылке позволяет решить две задачи:
  1. Передать объект в функцию (а не его копию)
  2. Избежать лишнего (иногда очень дорогого!) копирования
- *Константная ссылка* обеспечивает эффективную передачу аргумента и гарантирует, что объект не будет изменен
  - Нет смысла передавать базовые типы (`int`, `float`, `double`) по константной ссылке

```
void greetings(const string& name) {
    cout << "Good morning, " << name << "!\n";
}
```

std::string

<string>

```
#include <string>
#include <iostream>
using namespace std;

int main() {
    string a("Hello");
    string b("world");
    string c = a + ", " + b + "!";  // Hello, world!
    cout << c.size() << endl;  // 13
    string d = c.substr(7, 5);  // world
    size_t world_index = c.find("world"); // 7
    c.replace(world_index, 5, "Mike");  // Hello, Mike!
    int n = stoi("456");
    double x = stod("5.654");
    string pistr = "pi equals " + to_string(3.1415);
}
```

- Класс `string` работает с 8-битовыми символьными строками
- Поддерживает множество операций (смотрите документацию)
- Поддерживают операторы сравнения (лексикографический порядок) и операторы ввода-вывода
- Имеет инструменты для поиска внутри строки

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> vec = {1, 3, 5, 7, 9};
    for (int item : vec) cout << item << ' ';
    cout << '\n';
    cout << vec[3] << endl;  // 7
    cout << vec.size() << endl;  // 5
    vec[4] = 10;  // {1, 3, 5, 7, 10}
    vec.push_back(12);  // {1, 3, 5, 7, 10, 12}
    cout << vec.front() << ' '
         << vec.back() << endl;  // 1 12
    vec.pop_back();  // {1, 3, 5, 7, 10}

    vector<int> vec2(10);  // size = 10
    vec2.resize(20);  // size = 20
    vector<int> vec3(2, 5);  // {2, 2, 2, 2, 2}
    vector<vector<int>> vec2d;
    vec2d.push_back(vec3);
    return 0;
}
```

std::vector

`<vector>`

- Класс `vector` реализует тип данных *динамический массив*.
  - Позволяет хранить *гомогенные объекты* любого типа
  - Поддерживает эффективное *итерирование*
  - Поддерживает эффективную вставку и удаление *в конец*
  - Поддерживает множество других методов (смотрите документацию!)

Итерирование

```cpp
int main() {
    vector<int> vec = {1, 2, 3, 4, 5};

    for (int item : vec) cout << item << ' ';
    cout << endl;

    for (int& item : vec) ++item;

    for (size_t i = 0; i < vec.size(); ++i)
        cout << vec[i] << ' ';
    cout << endl;

    string s = "Novosibirsk";
    for (char ch : s) cout << ch << ' ';
    cout << endl;
}
```

```
> 1 2 3 4 5
> 2 3 4 5 6
> N o v o s i b i r s k
```

Заключительный пример

- Дано: текстовый файл со списком городов, их географическим расположением (широта и долгота) и населением
- Найти: пару городов с населением в каждом из которых не меньше некоторого порога и расстояние между которыми минимально.
- Определим удобные псевдонимы:

Ru.txt

```
"Moscow" (55.7558, 37.6178) 17125000
"Saint Petersburg" (59.9500, 30.3167) 5351935
"Novosibirsk" (55.0333, 82.9167) 1602915
"Yekaterinburg" (56.8356, 60.6128) 1468833
...
```

```
// City record: [City name, latitude, longitude, population]
using Record = tuple<string, double, double, int>;
using RecordVec = vector<Record>;
```

Считываем данные

```
Record parse_record(const string& line) {
    double lat, lon;
    int population;
    istringstream ss(line);
    ss.ignore(1);
    string name;
    while (ss.peek() != '"')
        name.push_back(ss.get());
    ss.ignore(3);
    ss >> lat;
    ss.ignore(2);
    ss >> lon;
    ss.ignore(2);
    ss >> population;
    return {name, lat, lon, population};
}
```

```
RecordVec read_records(
  const string& fname, int min_pop) {
    ifstream ifile(fname, ios::in);
    if (!ifile.good()) {}

    RecordVec records;
    string line;
    while (getline(ifile, line)) {
        auto record = parse_record(line);
        if (get<3>(record) > min_pop)
            records.push_back(record);
    }
    return records;
}
```

Перегрузка операторов

- Было бы удобно сравнивать записи оператором `!=` и выводить в их поток с помощью оператора `<<`. Это возможно, но для этого необходимо *перегрузить операторы*:

```
bool operator!=(const Record& lhs, const Record& rhs) {
    const auto& [city1, lat1, lon1, pop1] = lhs;
    const auto& [city2, lat2, lon2, pop2] = rhs;
    return city1 != city2 || lat1 != lat2 || lon1 != lon2 || pop1 != pop2;
}

ostream& operator<<(ostream& os, const Record& record) {
    const auto& [city, lat, lon, pop] = record;
    return os << city << " (" << lat << ", " << lon << ") " << pop;
}
```

Основная логика программы

```
pair<Record, Record> find_closest(const RecordVec& cities) {
    double min_dist = 1.e9;
    Record r1, r2;
    for (const auto& ri : cities) {
        for (const auto& rj : cities) if (ri != rj) {
            double cur_dist = dist(ri, rj);
            if (cur_dist < min_dist) {
                r1 = ri;
                r2 = rj;
                min_dist = cur_dist;
            }
        }
    }
    return {r1, r2};
}
```

Можно ли оптимизировать эту процедуру?

main

```cpp
int main(int argc, char* argv[]) {
    if (argc < 3) {
        cout << "Format: ./a.exe [ifname] [minpop]" << endl;
        return 0;
    }
    const string ifname = argv[1];
    const int min_pop = stoi(argv[2]);

    auto cities = read_records(ifname, min_pop);
    auto [record1, record2] = find_closest(cities);
    cout << "Minimal distance " << dist(record1, record2)
         << " km is between:\n"
         << " - " << record1 << " and\n"
         << " - " << record2 << endl;
    return 0;
}
```

Заключение

- Ввод и вывод осуществляется через потоки
  - `cout`, `cin`, `fstream`, `stringstream`
- Типы `pair` и `tuple` позволяют группировать объекты
- Передавать аргументы в функцию можно
  - По значению (или по константному значению)
  - По ссылке
  - По константной ссылке
- Строки представлены типом `string`
- Динамические массивы представлены типом `vector`

Backup

Стандартная библиотека

isocpp.org

- Стандартная библиотека *C++* содержит множество полезных инструментов

Строки
- string
- regex

Библиотеки C
[21 файл]

Потоки ввода-вывода
- ios
- iostream
- fstream
- iomanip
- sstream
- iosfwd
- ostream
- streambuf

Контейнеры STL
- set
- list
- array
- map
- bitset
- vector
- stack
- queue
- forward_list
- unordered_set
- unordered_map

Численные библиотеки
- complex
- random
- valarray
- numeric

Вспомогательные
- exception
- limits
- new
- typeinfo

Общие
- algorithm
- chrono
- memory
- functional
- iterator
- stdexcept
- tuple
- utility

Многопоточность
- thread
- mutex
- future
- condition_variable

Локализация
- locale
- codesvt

(Перечислены не все заголовочные файлы, но большая их часть)