

Программирование на C++ и Python

Лекция 1

Грибанов Сергей Сергеевич

14 сентября 2022 г.

ИЯФ СО РАН, НГУ

- Кратко изложить основные особенности языков программирования C++ и Python.
- Развить начальные навыки разработки на этих языках программирования.
- Сформировать начальное представление о базовых концепциях программирования:
 - алгоритмы и структуры данных,
 - парадигмы программирования,
- Познакомить с инструментами разработки:
 - система контроля версий (git),
 - компиляторы (gcc),
 - система сборки (cmake),
 - инструменты тестирования (pytest, GoogleTest)
 - ...
- Кратко познакомить с библиотеками научного программирования и анализа данных (numpy, matplotlib, SciPy и др.).

Учебная нагрузка

- 8 лекций,
- 16 практических занятий по 1.5 пары.

Зачет

1. **На тройку:** набрать по 5 баллов в каждом из 9 блоков заданий.
2. **На четверку:** выполнить условие 1 и набрать 60 или более баллов.
3. **На пятерку:** выполнить условие 1 и набрать 80 или более баллов.

C++

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

Python

1. Введение в Python;
2. Стандартная библиотека Python;
3. Вычисления с [numpy](#);
4. Построение диаграмм с [matplotlib](#).

- Выполнение проекта не является обязательным
- Желающие разбиваются на группы
- Между группами будет проводиться соревнование по написанию ИИ для игры в «Пятнашки»
- Победители определяются в двух номинациях:
 1. наиболее быстрый поиск решения
 2. наиболее оптимальный (по числу ходов) поиск решения
- Группа-победитель (в каждой номинации) поощряется: **+10 баллов** каждому участнику группы

13	10	11	6
5	7	4	8
1	12	14	9
3	15	2	

- Сайт курса: <https://cpp-python-nsu.inp.nsk.su>
 - Учебник: <https://cpp-python-nsu.inp.nsk.su/textbook>
 - Задания¹: <https://cpp-python-nsu.inp.nsk.su/assignments>
- Лекции:
 - <https://github.com/NSU-Programming/lectures2022>
 - <https://github.com/NSU-Programming/lectures2021>
 - <https://github.com/NSU-Programming/lectures2020>

- Telegram-группа: t.me/joinchat/dpx594KRwstiNDh



¹Задания сдаются через сервис github.com. Процедура описана на сайте курса.

- Эффективный и быстрый

- Развивается

Год	Стандарт
1998	C++98
2003	C++03
2011	C++11
2014	C++14
2017	C++17
2020	C++20

- Большое сообщество программистов \Rightarrow легко найти необходимую информацию
- Востребованный. Много научных библиотек / фреймворков написано на C++ или имеют интерфейсы для работы с использованием C++: Eigen 3, NLOpt, GSL, ROOT, Geant 4 и мн. другие.

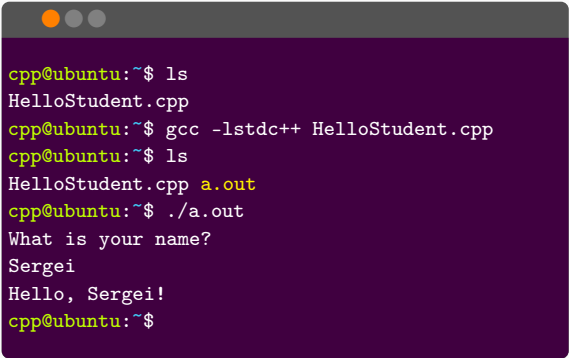
- Как высокоуровневый язык программирования
 - а не как развитие языка Си
- Продвинутое возможности языка (классы, шаблоны, динамическое выделение памяти) будем обсуждать в несколько этапов
 - Будем придерживаться принципа «от частного к общему»
 - Сначала научимся использовать стандартную библиотеку
 - Потом научимся использовать продвинутое возможности языка при написании собственных программ
- Предполагаем, что вы знакомы с языком Си

- cppreference.com и cplusplus.com — документация
- isocpp.org — Standar C++ Foundation
- hackingcpp.com — cheat sheets (таблицы с кратким описанием)
- поисковик + stackoverflow.com
- boost.org — расширенный набор библиотек
- Онлайн курсы по C++
 - tutorialspoint.com
 - ...
- Онлайн задачи: hackerrank.com
- Книги:
 1. Bjarne Stroustrup «The C++ programming language»
 2. Scott Meyers «Effective C++»
 3. Aditya Y. Bhargava «Grokking Algorithms»
 4. Luridas Panos «Algorithms for beginners»

Пример: Hello, student!

```
/* Файл HelloStudent.cpp */
// Подключаем заголовочные файлы
#include <iostream> // cin, cout, endl, ...
#include <string> // string

// Функция main - точка входа в программу
int main() {
    // Вывод текста в стандартный поток вывода
    std::cout << "What is your name?" << std::endl;
    // Создание строковой переменной
    std::string name;
    // Ввод из стандартного потока ввода
    std::cin >> name;
    std::cout << "Hello, " << name << "!" << std::endl;
    return 0;
}
```



```
cpp@ubuntu:~$ ls
HelloStudent.cpp
cpp@ubuntu:~$ gcc -lstdc++ HelloStudent.cpp
cpp@ubuntu:~$ ls
HelloStudent.cpp a.out
cpp@ubuntu:~$ ./a.out
What is your name?
Sergei
Hello, Sergei!
cpp@ubuntu:~$
```

- `std::cout` — объект для вывода в `stdout`
- `std::cin` — объект для ввода из `stdin`
- `std::endl` используется для перехода на новую строку
- `<<` — оператор вывода
- `>>` — оператор ввода

Объект для ввода-вывода	Его класс	Куда происходит вывод
<code>std::cout</code>	<code>std::ostream</code>	<code>stdout</code> ¹ (поток стандартного вывода)
<code>std::cin</code>	<code>std::istream</code>	<code>stdin</code> ² (поток стандартного ввода)
<code>std::cerr</code>	<code>std::ostream</code>	<code>stderr</code> ³ (поток вывода ошибок)
<code>std::clog</code>	<code>std::ostream</code>	<code>stderr</code>

¹`stdout` — файловый дескриптор стандартного вывода (в консоль). В буквальном смысле, это указатель на специальный «Си»-шный «файл», отвечающий стандартному выводу. Возможно из курса «Основы программирования» (язык C) вам известно, что вызов функции `fprintf(stdout, "hello")` выведет в консоль строку `"hello"`. Тут сказалась концепция UNIX-подобных операционных систем: «Любое устройство есть файл». Большинство операционных систем семейства UNIX (в частности Linux) написаны на языке C.

²`stdin` — файловый дескриптор стандартного ввода (с клавиатуры).

³`stderr` — файловый дескриптор стандартного вывода ошибок (в консоль).

```
[1]: // строка содержит 8-битные символы (тип каждого элемента char)
std::string s = "Hello, Student"; // присваиваем строке значение типа const char* (строковый литерал)
// к элементам строки можно обращаться по индексу, нумерация начинается с нуля
// чтобы получить размер строки, необходимо воспользоваться методом size()
std::cout << "s[0] = " << s[0] << ", s[1] = " << s[1] << ", s.size() = " << s.size();
```

s[0] = H, s[1] = e, s.size() = 14

```
[2]: // добавление символа (char) в конец
s.push_back('!');
std::string a(" I'm std::string!");
s.append(a); // добавление строки a в конец строки s
s.append(" I came from <string>."); // добавление строкового литерала в конец строки s
std::cout << s;
```

Hello, Student! I'm std::string! I came from <string>.

```
[3]: // вывод подстроки из 15 символов начиная с 16-той позиции
std::cout << s.substr(16, 15);
```

I'm std::string

```
[1]: std::string s = "Hello, Student!";
```

```
[2]: s.erase(0, 5); // стереть 5 символов, начиная с позиции 0  
s.insert(0, "Hi"); // вставить строчку, начиная с позиции 0  
std::cout << s;
```

Hi, Student!

```
[3]: // поиск позиции, в которой впервые появляется символ 't'  
std::cout << s.find('t') << std::endl;  
// если символ не найден, то выводит значение std::string::npos  
std::cout << s.find('z');
```

5
18446744073709551615

- имеется множество других методов
- поддерживаются операторы сложения, сравнения (лексикографический порядок) и операторы ввода-вывода
- подробное описание можно найти на cppreference.com
- по ссылке на QR-коде справа доступен **cheat sheet** по std::string



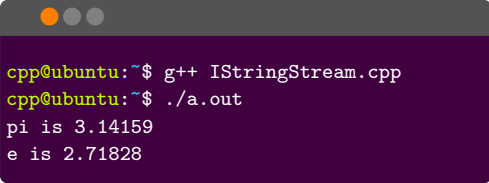
Пример: строковые потоки

```
/* Файл IStringStream.cpp */
#include <string>
#include <sstream> // строковые потоки
#include <iostream>

int main() {
    std::string s = "pi = 3.14159, e = 2.71828";
    std::istringstream stream(s); // строковый поток (input)
    std::string key;
    double value;
    // stream.eof() - true, если конец потока
    while (!stream.eof()) {
        stream >> key;
        stream.ignore(2); // ignore " = "
        stream >> value;
        stream.ignore(1); // ignore ','
        std::cout << key << " is " <<
            value << std::endl;
    }
    return 0;
}
```

Строковые потоки определены в заголовочном файле `<sstream>`.

- `std::istringstream` — ПОТОК ВВОДА
- `std::ostringstream` — ПОТОК ВЫВОДА
- `std::stringstream` — ПОТОК ВВОДА-ВЫВОДА

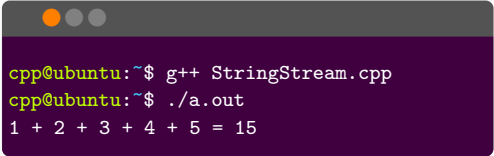


```
cpp@ubuntu:~$ g++ IStringStream.cpp
cpp@ubuntu:~$ ./a.out
pi is 3.14159
e is 2.71828
```

Пример: строковые потоки

```
/* Файл StringStream.cpp */
#include <sstream>
#include <iostream>
#include <string>

int main() {
    std::string x = "1 2 3 4 5";
    std::stringstream ss; // строковый поток (input/output)
    ss << x;
    int k, sum = 0;
    ss >> k;
    std::cout << k;
    for (sum = k; ss >> k; sum += k) std::cout << " + " << k;
    std::cout << " = " << sum << std::endl;
    return 0;
}
```



```
cpp@ubuntu:~$ g++ StringStream.cpp
cpp@ubuntu:~$ ./a.out
1 + 2 + 3 + 4 + 5 = 15
```

Пример: чтение из файла

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

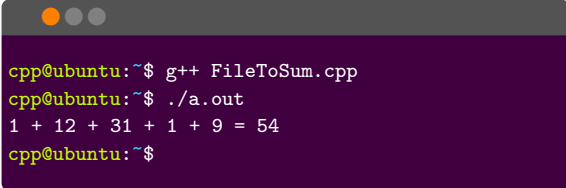
```
/* Файл FileToSum.cpp */
#include <fstream>
#include <iostream>

int main() {
    std::ifstream file("data.txt");
    if (!file.good()) {
        std::cerr << "Can't load file!\n";
        return 1;
    }
    int value, sum = 0;
    file >> value;
    std::cout << value;
    sum += value;
```

```
while (file >> value) {
    std::cout << " + " << value;
    sum += value;
}
file.close();
std::cout << " = " << sum << std::endl;
return 0;
}
```

Файл data.txt

1 12 31 1 9



```
cpp@ubuntu:~$ g++ FileToSum.cpp
cpp@ubuntu:~$ ./a.out
1 + 12 + 31 + 1 + 9 = 54
cpp@ubuntu:~$
```

Потоки

- `std::ifstream` — для чтения из файла
- `std::ofstream` — для записи в файл
- `std::fstream` — для совмещения чтения и записи

Примеры инициализации

```
std::fstream file("data.txt", std::ios::in);  
то же самое, что  
std::ifstream file("data.txt", std::ios::in);  
то же самое, что  
std::ifstream file("data.txt");
```

Режимы открытия файла

- `std::ios::in` — файл открывается для чтения
- `std::ios::out` — файл открывается для записи
- `std::ios::app` — файл открывается для дозаписи (строго в конец файла)
- `std::ios::ate` — после открытия файла указатель перемещается в конец, но с помощью метода `seek` можно переместить указатель в любое место файла
- `std::ios::binary` — файл открывается в бинарном режиме

Запись в текстовый файл

```
std::ofstream file("output.txt");  
for (int i = 0; i < 5; ++i)  
    file << i << " ";
```

Чтение по символу

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

Чтение по строке

```
std::string line;  
while (getline(file, line)) {  
    cout << line;  
}
```

Запись в бинарный файл

```
std::ofstream file("output.bin", std::ios::binary);  
for (int i = 0; i < 5; ++i)  
    // reinterpret_cast - приведение типов  
    // в данном случае: int* --> char*  
    file.write(reinterpret_cast<char*>(&i), sizeof(int));
```

Чтение из бинарного файла

```
// открытие бинарного файла на чтение  
std::fstream file("output.bin", std::ios::in|std::ios::binary);  
int value;  
while (true) {  
    file.read(reinterpret_cast<char*>(&value), sizeof(int));  
    /* file.eof() возвращает true, если достигнут конец  
    файла; в противном случае возвращается false */  
    if (file.eof()) break;  
    std::cout << value << " ";  
}
```

При передаче аргументов в функцию создаются копии этих аргументов (внутри функции).

Передача без ссылки

```
void fcn1(std::string x) {  
    // x --- копия аргумента, переданного в  
    // функцию  
    x = "abc"; // изменится  
    ...  
}  
...  
std::string str = "hello";  
fcn1(str); // str не изменится!  
  
// void fcn1(std::string x) - плохо, т.к.  
// чем больше аргумент (строка) x занимает  
// места в памяти, тем больше будет  
// тратиться времени на копирование этого  
// аргумента при его передаче в функцию
```

Передача по ссылке

```
void fcn2(std::string& x) {  
    x = "qwe"; // x ссылается на str  
    ...  
}  
...  
std::string str = "hello";  
fcn2(str); // str изменится на "qwe"!  
  
// стоимость передачи строки str  
// внутри функции низкая и не зависит  
// от размера этой строки, т.к.  
// строка передается по ссылке
```

Как передать объект в функцию по ссылке, но при этом запретить изменение объекта внутри функции?

Передача по константной ссылке

```
const fcn3(const std::string& x) {  
    // x = "asd";  
    /* если убрать комментарий выше,  
     * то получим ошибку, так как x  
     * ссылается на объект типа  
     * const std::string, который  
     * является неизменяемым  
     * (ключевое слово const) */  
    ....  
}
```

Выводы

1. Передача аргумента по ссылке позволяет:
 - 1.1 передать в функцию сам объект, а не его копию
 - 1.2 избежать излишнего (возможно дорогого) копирования
2. Константная ссылка обеспечивает эффективную передачу и гарантирует, что объект не будет изменен
3. Нет смысла передавать базовые типы (char, int, float, double, long) по константной ссылке

Контейнер `std::tuple` предназначен для хранения фиксированного числа гетерогенных объектов. Для использования необходимо подключить заголовочный файл `<tuple>`.

Инициализация / присваивание

```
// [пример 1] - инициализация с помощью конструктора
std::tuple<std::string, std::string, double> w("hello", "world", 1.23);

// [пример 2] - инициализация / присваивание с помощью фигурных скобок
std::tuple<std::string, double, int> x = {"abc", 3.14, -1};

// [пример 3] - инициализация с помощью std::make_tuple
auto y = std::make_tuple("e", 2.718);
// Тип y: std::tuple<const char*, double>

// [пример 4] - инициализация / присваивание с помощью std::make_tuple
std::tuple<std::string, double> z = std::tuple("e", 2.718);
// Тип z: std::tuple<std::string, double>
```

Обращение к элементам

```
// [пример 5] - (first, second)
std::tuple<std::string, int, double> a;
// присваивание значения "abc" 0-му элементу
std::get<0>(a) = "abc";
// присваивание значения -3 1-му элементу
std::get<1>(a) = -3;
// присваивание значения -2.198 2-му элементу
std::get<2>(a) = -2.198;
std::cout << std::get<1>(a);
```

Распаковка

```
std::tuple<std::string, double, int> a;
a = {"abc", 2.2, 1};
// [пример 6] - "распаковка" std::tuple
auto [f1, s1, t1] = a;
// f1 (std::string), s1 (double),
// t1 (int) - копии элементов из a

// [пример 7] - "распаковка" std::tuple
// по ссылке
auto& [f2, s2, t2] = a;
// f2, s2, t2 - ссылки на элементы a;
// если изменить f2, s2 или t2,
f2 = "qwe";
// то изменится соответствующий элемент в a

const auto& [f3, s3, t3] = a;
```

Определим тип данных `GeoCoord`, который будем использовать для задания географических координат в следующих примерах.

```
// GeoCoord - псевдоним для  
// типа std::tuple<std::string, double, double>  
using GeoCoord = std::tuple<std::string, double, double>;
```

Первый элемент — название точки, второй — широта, третий — долгота.

```
std::istream& operator >>
(std::istream& is, GeoCoord& x) {
    char c = 0;
    while (c != '(' && !is.eof()) is >> c;
    if (is.eof()) return is;
    auto& [label, lat, lon] = x;
    label.clear(); // чистим строку
    c = 0;
    do {
        if (c) label.push_back(c);
        is.get(c);
    } while (c != ';');
    is >> lat; // read latitude
    is.ignore(1); // skip ','
    is >> lon; // read longitude
    is.ignore(1); // skip ')'
    return is;
}
```

Перегрузка оператора ввода. Предполагается, что формат данных на входе следующий: (name; latitude, longitude).

```
std::ostream&
operator << (std::ostream& os,
            const GeoCoord& x) {
    os << "(" <<
        std::get<0>(x) << "; " <<
        std::get<1>(x) << ", " <<
        std::get<2>(x) << ")";
    return os;
}
```

Перегрузка оператора вывода. Предполагается, что формат вывода данных такой же, как и при вводе: (name; latitude, longitude).

Пример: перегрузка операторов

```
#include "GeoCoord.hpp"
#include <fstream>
#include <sstream>
#include <vector>
#include <iostream>
#include <iomanip>

int main() {
    std::string str =
        "(dormitory no. 7; 54.848857, 83.092331)";
    // строковый поток (ввода)
    std::istringstream stream(str);
    GeoCoord start;
    stream >> start;
    std::ifstream fl("coords.txt");
    std::vector<GeoCoord> coords;
    GeoCoord temp;
    while (fl >> temp) coords.push_back(temp);
    // выводить расстояние с точностью до 1 метра
    std::cout << std::fixed << std::setprecision(3);
```

```
    for (const auto& el : coords) {
        std::cout << std::get<0>(el) << ": " <<
            dist(start, el) << " km" << std::endl;
    }
    return 0;
}
```

Вывести расстояние (в км) от общежития №7 до некоторых других объектов на территории Академгородка. Координаты общежития №7 необходимо прочитать из строкового потока, названия и координаты всех остальных объектов необходимо прочитать из файла coords.txt.

Пример: перегрузка операторов

```
#include "GeoCoord.hpp"
#include <fstream>
#include <sstream>
#include <vector>
#include <iostream>
#include <iomanip>

int main() {
    std::string str =
        "(dormitory no. 7; 54.848857, 83.092331)";
    // строковый поток (ввода)
    std::istringstream stream(str);
    GeoCoord start;
    stream >> start;
    std::ifstream fl("coords.txt");
    std::vector<GeoCoord> coords;
    GeoCoord temp;
    while (fl >> temp) coords.push_back(temp);
    // выводить расстояние с точностью до 1 метра
    std::cout << std::fixed << std::setprecision(3);
```

```
    for (const auto& el : coords) {
        std::cout << std::get<0>(el) << ": " <<
            dist(start, el) << " km" << std::endl;
    }
    return 0;
}
```

Файл coords.txt

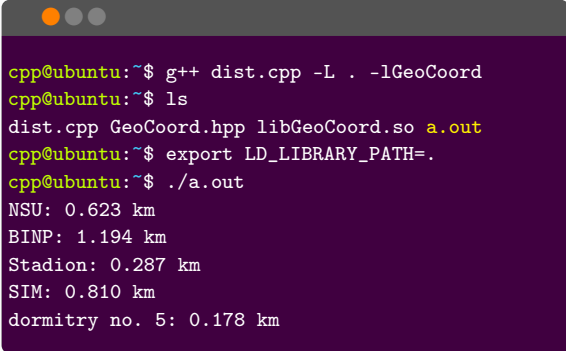
```
(NSU; 54.843269, 83.093104)
(BINP; 54.848810, 83.110989)
(Stadion; 54.847447, 83.096089)
(SIM; 54.846467, 83.104279)
(dormitry no. 5; 54.847260, 83.092349)
```

Пример: перегрузка операторов

```
#include "GeoCoord.hpp"
#include <fstream>
#include <sstream>
#include <vector>
#include <iostream>
#include <iomanip>

int main() {
    std::string str =
        "(dormitory no. 7; 54.848857, 83.092331)";
    // строковый поток (ввода)
    std::istringstream stream(str);
    GeoCoord start;
    stream >> start;
    std::ifstream fl("coords.txt");
    std::vector<GeoCoord> coords;
    GeoCoord temp;
    while (fl >> temp) coords.push_back(temp);
    // выводить расстояние с точностью до 1 метра
    std::cout << std::fixed << std::setprecision(3);
```

```
    for (const auto& el : coords) {
        std::cout << std::get<0>(el) << ": " <<
            dist(start, el) << " km" << std::endl;
    }
    return 0;
}
```



```
cpp@ubuntu:~$ g++ dist.cpp -L . -lGeoCoord
cpp@ubuntu:~$ ls
dist.cpp GeoCoord.hpp libGeoCoord.so a.out
cpp@ubuntu:~$ export LD_LIBRARY_PATH=.
cpp@ubuntu:~$ ./a.out
NSU: 0.623 km
BINP: 1.194 km
Stadion: 0.287 km
SIM: 0.810 km
dormitry no. 5: 0.178 km
```

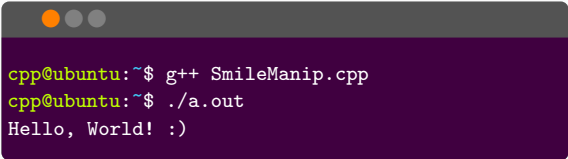
`std::fixed` и `std::setprecision`, используемые в предыдущем примере, — это примеры манипуляторов, т.е. специального вида функций, используемых для изменения формата потока. Последовательное использование манипуляторов `std::fixed` и `std::setprecision`, позволяет задать точность вывода чисел с плавающей точкой (в количестве знаков после запятой). Эти и другие стандартные манипуляторы становятся доступными после подключения заголовочного файла `<iomanip>`. Пользователь может создавать собственные манипуляторы. На следующем слайде приведен пример простейшего манипулятора.

Пример: простейший манипулятор

```
/* Файл SmileManip.cpp */
#include <iostream>

std::ostream& smile(std::ostream& os) {
    os << ":)";
    return os;
}

int main() {
    std::cout << "Hello, World! " << smile << std::endl;
    return 0;
}
```



```
cpp@ubuntu:~$ g++ SmileManip.cpp
cpp@ubuntu:~$ ./a.out
Hello, World! :)
```

`std::pair` — контейнер стандартной библиотеки, представляющий собой пару из двух гетерогенных объектов. Для того, чтобы получить доступ к `std::pair`, необходимо загрузить заголовочный файл `<utility>`. В использовании контейнер `std::pair` похож на `std::tuple`.

Инициализация / присваивание

```
/* [пример 1] - инициализация с помощью конструктора  
* первый элемент пары строка "pi" (std::string),  
* второй элемент пары число 3.14 (double) */
```

```
std::pair<std::string, double> w("pi", 3.14);
```

```
// [пример 2] - инициализация / присваивание с помощью фигурных скобок
```

```
std::pair<double, int> x = {1.23, -1};
```

Инициализация / присваивание

```
// [пример 3] - инициализация с помощью std::make_pair  
auto y = std::make_pair("e", 2.718);  
// Тут y: std::pair<const char*, double>  
  
// [пример 4] - инициализация / присваивание с помощью std::make_pair  
std::pair<std::string, double> z = std::make_pair("e", 2.718);  
// Тут z: std::pair<std::string, double>
```

Обращение к элементам

```
// [пример 5] - (first, second)  
std::pair<std::string, int> a;  
a.first = "abc"; // присваивание значения "abc" первому элементу пары  
a.second = -3; // присваивание значения -3 второму элементу пары  
// вывод первого (first) и второго (second) элементов пары:  
std::cout << a.first << ", " << a.second << std::endl;  
// вывод: abc, -3
```

Распаковка

```
// [пример 6] - "распаковка" пары  
auto [f1, s1] = a;  
std::cout << f1 << ", " << s1 << std::endl;  
// вывод: abc, -3  
// или так:  
auto& [f2, s2] = a;  
// или так:  
const auto& [f3, s3] = a;
```


Вектор

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

int main() {
    std::vector<int> a = {1, 2, 3, 4, 5};
    a.push_back(-1); // a = {1, 2, 3, 4, 5, -1}
    std::cout << a.front() << " "
              << a.back() << std::endl; // 1 -1
    std::cout << a[1] << std::endl; // 2
    std::cout << a.size() << std::endl; // 6
    std::cout << a.capacity() << std::endl; // 10
    a.reserve(100);
    std::cout << a.capacity() << std::endl; // 100
    a.resize(8, -3); // a = {1, 2, 3, 4, 5, -1, -3, -3}
    a.pop_back(); // a = {1, 2, 3, 4, 5, -1, -3}
    a.clear(); // a = {}
    return 0;
}
```

- Контейнер `std::vector` реализует тип данных *динамический массив*
- `std::vector` доступен после подключения заголовочного файла `<vector>`
- Позволяет хранить гомогенные объекты любого типа
- Поддерживает эффективную вставку (и удаление) в конец
 - Однако в худшем случае для вставки может понадобиться время $O(n)$
- Быстрое чтение элемента — $O(1)$.
- Поддерживает множество других методов.

Спасибо за внимание!

1. С появлением первых языков программирования высокого уровня, появилось множество программ.
2. У программистов возникла потребность не писать код «с нуля», а использовать чужие наработки.
3. Ага, в C++ у нас имеется препроцессорная директива `#include`. Будем использовать ее!
4. А как использовать?
5. А давайте **будем «includ-ить» сторонний код целиком**, т.е. не только определение (описание) функций, классов и др., но и их имплементацию.
6. Оказывается, **так делать плохо! Почему?**
7. Далее, для краткости, подход, изложенный в пункте 5, будем обозначать как **TOTAL INCLUDE**.

Недостатки TOTAL INCLUDE:

- трата ресурсов на компиляцию стороннего кода (при сборке собственной программы),
- увеличение размера готовых пользовательских программ (за счет стороннего кода),
- нет разделения между интерфейсом и конкретной реализацией.

Что тогда делать?

Заголовочные файлы и библиотеки

Исходные данные:

- Имеется сторонний код с набором полезных функций, классов и др.
- Этот код не содержит точку входа в программу, т.е. функцию `int main()`.
- Сторонний код делится на заголовочные файлы `.hpp` (`.h`) и файлы с имплементацией `.cpp`.
- Заголовочные файлы содержат только описание. Например, описание функций:

```
void fcn1(int);  
double fcn2(double, double);
```

Пусть имплементацию (тела функций) заголовочные файлы не содержат.

- `.cpp` файлы, напротив, содержат имплементацию (функций и др. из заголовочных файлов).

Этот код можно скомпилировать с помощью компилятора и получить файл, содержащий имплементацию функций и др. из стороннего кода уже в бинарном виде. Такой файл называется библиотекой.

Программисту для разработки с использованием сторонней библиотеки необходимы:

- сама библиотека (бинарный файл),
- интерфейс в виде соответствующих заголовочных файлов на языке C/C++.

Для запуска программы, зависящей от сторонней библиотеки необходимы:

- только сама библиотека (бинарный файл).

- **Статические библиотеки.** Все функции статической библиотеки, которые использует ваша программа, становятся частью исполняемого файла. В Windows: `.lib`, в Linux: `.a` (archive).
- **Динамические библиотеки.** При компиляции вашего исполняемого файла динамическая библиотека не становится частью исполняемого файла. Она подгружается программой во время запуска / работы программы. Т.е. в одну и ту же динамическую библиотеку могут одновременно использовать сразу несколько программ. В Windows: `.dll` (dynamic-link library), в Linux: `.so` (shared object).

Процесс компиляции состоит из нескольких этапов:

1. **Препроцессинг** — обработка файлов исходного кода в соответствии с языком препроцессора C/C++.
2. **Ассемблирование** — превращение кода на языке C/C++ в код на языке ассемблера.
3. **Компилирование** — процесс превращения кода на языке Ассемблер в объектные файлы.
4. **Компановка** (линковка) — процесс объединения файлов проекта и используемых библиотек в единую сущность.

Чуть подробнее прочитать про процесс компиляции можно, например, тут <https://habr.com/ru/post/478124>.

Вернемся к GeoCoord

```
/* Файл GeoCoord.hpp */
#pragma once
#include <string>
#include <tuple>
#include <iostream>

// GeoCoord - псевдоним для
// типа std::tuple<std::string, double, double>
using GeoCoord = std::tuple<std::string, double, double>;
// Описание оператора >> для типа GeoCoord
// Имплементация содержится в GeoCoord.cpp и
// будет приведена ниже
std::istream& operator >> (std::istream&, GeoCoord&);
// Объявление оператора << для типа GeoCoord
// Имплементация содержится в GeoCoord.cpp и
// будет приведена ниже
std::ostream& operator << (std::ostream&, const GeoCoord&);
// Описание функции dist
// Имплементация содержится в GeoCoord.cpp
double dist(const GeoCoord&, const GeoCoord&);
```

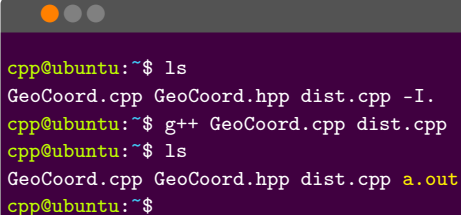
На рисунке слева приведен заголовочный GeoCoord.hpp. В файле GeoCoord.cpp имплементированы функции, объявленные в заголовочном файле. В файл GeoCoord.cpp подгружается файл GeoCoord.hpp с помощью `#include "GeoCoord.hpp"`.

Компиляция проекта из нескольких файлов

```
/* Файл GeoCoord.hpp */
#pragma once
#include <string>
#include <tuple>
#include <iostream>

// GeoCoord - псевдоним для
// типа std::tuple<std::string, double, double>
using GeoCoord = std::tuple<std::string, double, double>;
// Описание оператора >> для типа GeoCoord
// Имплементация содержится в GeoCoord.cpp и
// будет приведена ниже
std::istream& operator >> (std::istream&, GeoCoord&);
// Объявление оператора << для типа GeoCoord
// Имплементация содержится в GeoCoord.cpp и
// будет приведена ниже
std::ostream& operator << (std::ostream&, const GeoCoord&);
// Описание функции dist
// Имплементация содержится в GeoCoord.cpp
double dist(const GeoCoord&, const GeoCoord&);
```

Если тип `GeoCoord` и соответствующие ему функции планируется использовать в одном единственном проекте, то в создании динамической библиотеки нет необходимости. Пусть функция `int main()` имплементирована в файле `dist.cpp`. В файл `dist.cpp` подгружается файл `GeoCoord.hpp` с помощью `#include "GeoCoord.hpp"`.



```
cpp@ubuntu:~$ ls
GeoCoord.cpp GeoCoord.hpp dist.cpp -I.
cpp@ubuntu:~$ g++ GeoCoord.cpp dist.cpp
cpp@ubuntu:~$ ls
GeoCoord.cpp GeoCoord.hpp dist.cpp a.out
cpp@ubuntu:~$
```

Использование динамической библиотеки в своем проекте

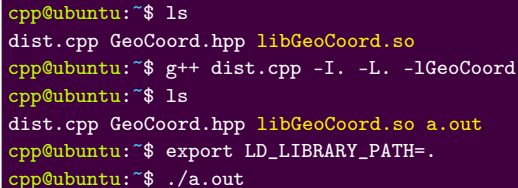
Допустим, что мы хотим использовать GeoCoord в нескольких проектах. Тогда нам необходимо скомпилировать динамическую библиотеку.



```
cpp@ubuntu:~$ ls
GeoCoord.cpp GeoCoord.hpp
cpp@ubuntu:~$ g++ -fPIC GeoCoord.cpp -I. --shared -o libGeoCoord.so
cpp@ubuntu:~$ ls
GeoCoord.cpp GeoCoord.hpp libGeoCoord.so
cpp@ubuntu:~$
```

Использование динамической библиотеки в своем проекте

Допустим, что у нас имеется динамическая библиотека `libGeoCoord.so` и соответствующий ей заголовочный файл `GeoCoord.hpp`. Мы хотим воспользоваться этой библиотекой в проекте `dist.cpp` (в данном примере всего один файл, но это не обязательно так). В `dist.cpp` объявлена функция `int main()`, и туда подгружается заголовочный файл `GeoCoord.hpp`: `#include <GeoCoord.hpp>`.



```
cpp@ubuntu:~$ ls
dist.cpp GeoCoord.hpp libGeoCoord.so
cpp@ubuntu:~$ g++ dist.cpp -I. -L. -lGeoCoord
cpp@ubuntu:~$ ls
dist.cpp GeoCoord.hpp libGeoCoord.so a.out
cpp@ubuntu:~$ export LD_LIBRARY_PATH=.
cpp@ubuntu:~$ ./a.out
```

На Windows вместо переменной `LD_LIBRARY_PATH`, нужно добавить путь к переменной `PATH`.