

# Программирование на языке C++

## Лекция 4

C++

# Структуры

## Постановка задачи

- Хранить в программе описание характеристик некоторого объекта

## Решение I

```
int aliceBirthYear;  
int aliceBirthMonth;  
int aliceBirthDay;  
double aliceHeight;  
double aliceWeight;
```

```
int bobBirthYear;  
int bobBirthMonth;  
int bobBirthDay;  
double bobHeight;  
double bobWeight;
```

## Решение I - Проблемы

- Для каждого человека нужно создавать по пять отдельных переменных – **долго, могут быть опечатки**
- Чтобы передать в функцию, нужно перечислить все аргументы – **можно перепутать порядок**

```
print(aliceBirthYear, aliceBirthMonth,  
      aliceBirthDay, aliceHeight, aliceWeight);
```

- Как вернуть из функции?

## Решение II - Структуры

```
struct human {      // Свой тип данных
    int BirthYear;
    int BirthMonth;
    int BirthDay;
    double Height;
    double Weight;
};    // Точка с запятой обязательно

human alice, bob;    // Создаём переменные
```

## Решение II - Структуры

```
struct human {  
    int BirthYear;  
    int BirthMonth;  
    int BirthDay;  
    double Height;  
    double Weight;  
} alice, bob;
```

## Решение II - Структуры

```
struct {  
    int BirthYear;  
    int BirthMonth;  
    int BirthDay;  
    double Height;  
    double Weight;  
} alice, bob;
```



## Где можно объявлять структуры?

- Внутри функций

```
void func(){
    struct num{int i;} var;
};
```
- Вне функций

```
struct num{int i;} var;
void func(){
};
```
- Внутри других структур

```
struct num{
    int i;
    struct {int k;} j;
} var;
```

## Что может быть членом структуры?

Если можно создать переменную этого типа, то это может быть членом структуры

Например:

- Примитивные типы: `int`, `double`, `char` ...
- Другие структуры;
- Массивы;
- Строки;
- ...

# Как работать со структурой

```
struct Data{  
    int Year;  
    int Month;  
    int Day;  
};
```

```
Data now;  
now.Year = 2018;  
now.Day = 9;  
now.Month = 11;
```

## Как работать со структурой

```
now.Year = now.Year + 1; // 2019
```

```
cout << now.Day; // 9
```

```
now.Month = now.Day + now.Year; // 2028
```

```
int *p = &now.Month;
```

# Инициализация структуры I

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```

```
// joe.id = 1, joe.age = 32, joe.wage = 60000.0
```

```
Employee joe = { 1, 32, 60000.0 };
```

```
// frank.id = 2, frank.age = 28, frank.wage = 0.0
```

```
Employee frank = { 2, 28 };
```

```
Employee frank { 2, 28 };    // C++11
```

## Инициализация структуры II C++11/C++14

```
struct Rectangle {  
    double length = 1.0;  
    double width = 1.0;  
};
```

```
int main() {  
    Rectangle x; // length = 1.0, width = 1.0  
    x.length = 2.0; // Меняем значение  
    return 0;  
}
```

## Инициализация структуры III C++11/C++14

```
struct Rectangle {  
    double length = 1.0;  
    double width = 1.0;  
};  
  
int main() {  
    // C++11 – Ошибка; C++14 – Разрешено  
    Rectangle x = {1.0, 1.0};  
  
    return 0;  
}
```

## Присваивание значений структурам I

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```

```
Employee joe;  
joe.id = 1;  
joe.age = 32;  
joe.wage = 60000.0;
```



## Присваивание значений структурам II

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```

```
Employee joe = {1, 20, 3.0}, mike;
```

```
mike = joe; // Копирование значений joe в mike
```

```
// Присваивание полям joe новых значений C++14
```

```
joe = {2, 22, 6.3};
```

## Передача структуры как параметр в функцию

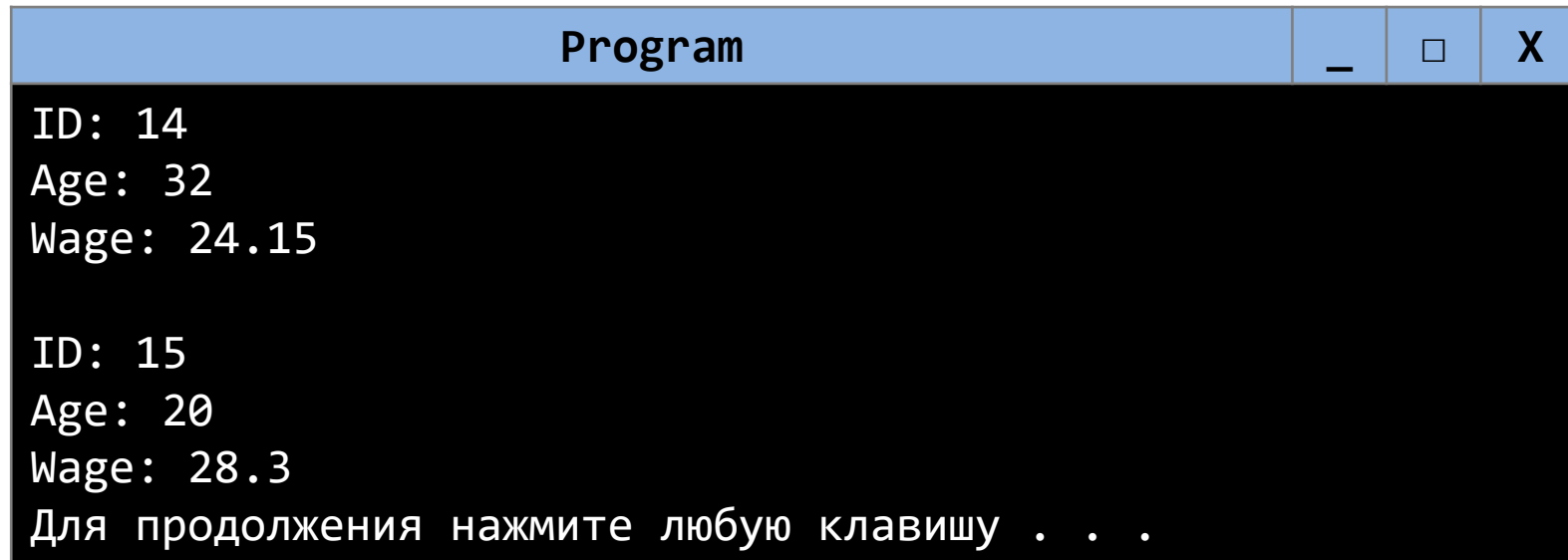
```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```

```
void printInformation(Employee employee) {  
    std::cout << "ID: " << employee.id << "\n";  
    std::cout << "Age: " << employee.age << "\n";  
    std::cout << "Wage: " << employee.wage << "\n";  
}
```

## Передача структуры как параметр в функцию

```
int main() {  
    Employee joe = { 14, 32, 24.15 };  
  
    printInformation(joe);  
    std::cout << "\n";  
  
    printInformation({ 15, 20, 28.3 });  
    return 0;  
}
```

## Передача структуры как параметр в функцию



```
Program
ID: 14
Age: 32
Wage: 24.15

ID: 15
Age: 20
Wage: 28.3
Для продолжения нажмите любую клавишу . . .
```

The image shows a screenshot of a Windows console window. The title bar is light blue and contains the text "Program" followed by standard minimize, maximize, and close buttons. The console area has a black background with white text. The output displays two sets of data: the first set for ID 14 with age 32 and wage 24.15, and the second set for ID 15 with age 20 and wage 28.3. At the end, there is a prompt in Russian asking the user to press any key to continue, followed by three dots.

## Передача структуры в функцию через указатель

```
void printInformation(Employee *employee) {  
    std::cout << "ID: " << (*employee).id << "\n";  
    std::cout << "Age: " << (*employee).age << "\n";  
    std::cout << "Wage: " << (*employee).wage << "\n";  
}
```

```
void printInformation(Employee *employee) {  
    std::cout << "ID: " << employee->id << "\n";  
    std::cout << "Age: " << employee->age << "\n";  
    std::cout << "Wage: " << employee->wage << "\n";  
}
```

# Возврат структур из функций

```
struct Point3d {  
    double x, y, z;  
};
```

```
Point3d getZeroPoint() {  
    Point3d temp = { 0.0, 0.0, 0.0 };  
    return temp;  
}
```

```
int main() {  
    Point3d zero = getZeroPoint();  
    return 0;  
}
```

Дополнительные сведения

# Разные типы

```
struct Point3d {  
    double x, y, z;  
};
```

```
struct Vector3d {  
    double x, y, z;  
};
```

```
Point3d p = { 0.0, 0.0, 0.0 };
```

```
Vector3d v;
```

```
v = p; // Ошибка. У v и p разные типы
```



# Массив структур

```
struct Point3d {  
    double x, y, z;  
};
```

```
Point3d p[2] = {{}, {1.0, 2.0, 3.0}};
```

```
p[0].x = 1.0;
```

```
std::cout << p[0].x << ' ' << p[0].y << ' ' << p[0].z;
```

# Вложенные структуры

```
struct Employee {  
    short id;  
    int age;  
    float wage;  
};
```

```
struct Company {  
    Employee CEO;    // CEO – это структура  
    int numberOfEmployees;  
};
```

```
Company myCompany = {{ 1, 42, 60000.0f }, 5 };  
std::cout << myCompany.CEO.id;
```

## Размер структуры и выравнивание I

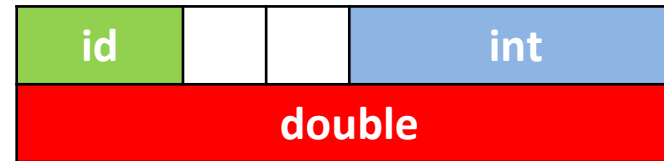
```
struct Employee {  
    short id;    // sizeof(short) == 2  
    int age;     // sizeof(int) == 4  
    double wage; // sizeof(double) == 8  
};  
  
sizeof(Employee); // 16 != ( 2 + 4 + 8 )
```

## Размер структуры и выравнивание II

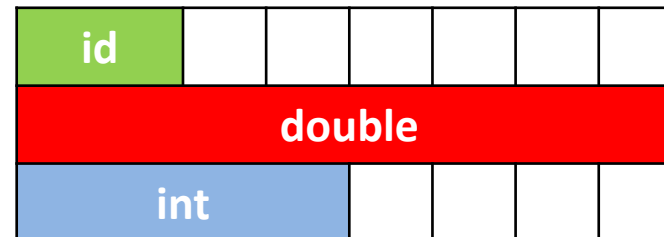
```
struct Employee {  
    short id;      // sizeof(short) == 2  
    double wage;   // sizeof(double) == 8  
    int age;       // sizeof(int) == 4  
};  
  
sizeof(Employee); // 24 != ( 2 + 4 + 8 )
```

## Размер структуры и выравнивание II

```
struct Employee {  
    short id;  
    int age;  
    double wage;  
};
```



```
struct Employee {  
    short id;  
    double wage;  
    int age;  
};
```



Объединения

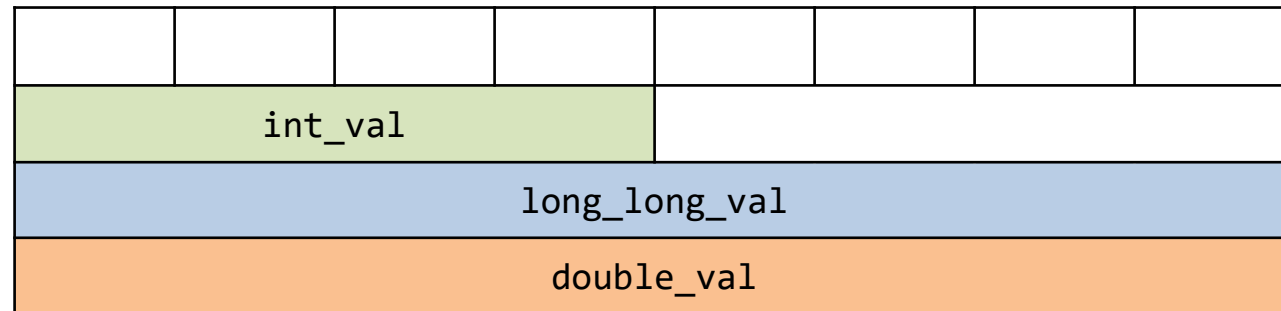
## Что такое объединение?

**Объединение** — это пользовательский тип данных, который может хранить в пределах *одной области* памяти разные типы данных, но в каждый момент времени только один из них.

Размер объединения определяется размером крупнейшего поля.

# Объявление

```
union one4all {  
    int int_val;  
    long long long_long_val;  
    double double_val;  
};
```





# Инициализация

```
union one4all {  
    int int_val;  
    long long long_long_val;  
    double double_val;  
};
```

// Выражение вычисляется и присваивается  
первому полю в объединении

```
one4all num = {10.1};  
cout << num.int_val; // 10
```

## Использование I

```
one4all pail;  
pail.int_val = 15;           // сохранение int  
cout << pail.int_val;       // 15  
  
pail.double_val = 1.38;     // сохранение double  
cout << pail.double_val;    // 1.38  
cout << pail.int_val;       // -515396076
```

## Использование II

```
struct widget {  
    char brand[20];  
    int type; // Определяет что лежит в id_val  
    union id {  
        long id_num;  
        char id_char[20];  
    } id_val;  
};
```

## Анонимные объединения

```
union {                                // Нет имени
    long id_num;
    char id_char[20];
};                                     // Нет переменных
```

Две переменные работающие с одной областью памяти

## Анонимные объединения

```
int main(){
    union {
        int i;
        double d;
    };
    i = 12345678;
    cout << i << '\n';    // 12345678

    d = 12345678;
    cout << i << '\n';    // -1073741824
}
```

# Перечисления

# Что такое перечисление

Перечисление – это пользовательский тип данных, определяющий набор целочисленных констант.

Зачем нужен:

- Сделать код более читабельным путём замены «магических чисел» на элементы перечисления;  
Пример: `return 0;` `return SUCCESS;`
- Как дополнительный контроль, защищающий от случайных, автоматических преобразований типов.

# Объявление I

```
enum Color {  
    // Элементы перечисления называются перечислителями  
    // Они определяют все допустимые значения данного типа  
    COLOR_BLACK,    // Перечислители разделяются запятыми  
    COLOR_RED,      // Обычно они пишутся заглавными буквами  
    COLOR_BLUE,     // но это не обязательно  
    COLOR_GREEN,  
    COLOR_WHITE,  
    COLOR_CYAN,  
    COLOR_YELLOW,  
    COLOR_MAGENTA,  // В C++11 можно ставить запятую в конце  
};                // Точка с запятой обязательна
```



## Объявление II

```
enum Color {  
    COLOR_BLACK,          // Присваивается целое значение 0  
    COLOR_RED,            // 1  
    COLOR_BLUE = 7,       // Можно присвоить своё значение  
    COLOR_GREEN,          // 8 Нумерация продолжается  
    COLOR_WHITE = 7,      // Можно дублировать значения  
    COLOR_CYAN,           // 8  
    COLOR_YELLOW,         // 9  
    COLOR_MAGENTA = -1    // отрицательные тоже допускаются  
};
```

## Объявление III

```
enum Color {  
    YELLOW,  
    BLACK, // имя BLACK теперь занято  
    PINK  
};
```

```
enum Feelings {  
    SAD,  
    ANGRY,  
    BLACK // ошибка, BLACK уже использован в Colors  
};
```

```
int BLACK = 3; // ошибка
```

# Переменные

```
enum Color {  
    YELLOW,  
    BLACK,  
    PINK  
} a;           // Создание во время объявления
```

```
enum {  
    ONE,  
    TWO,  
    THREE  
} b;           // Создание из анонимного перечисления
```

```
Color c;       // Обычным способом
```

## Инициализация / Присваивание

```
enum Color {  
    YELLOW,  
    BLACK,  
    PINK  
};
```

```
Color c = YELLOW;  
Color pig(PINK);  
Color zebr = Color::BLACK;  
Color window = 0;  // Ошибка
```

## Ввод / Вывод

```
enum Color {  
    YELLOW,  
    BLACK,  
    PINK  
};
```

```
Color pig(PINK);  
cout << pig;      // Преобразуется в число (2)  
cin >> pig;       // Ошибка компиляции
```

```
int input;  
cin >> input;  
pig = static_cast<Color>(input);
```

# Операции I

```
enum Color { YELLOW, BLACK, PINK };
```

Перечисления преобразуются в целое число автоматически:

```
Colors c = BLACK;
```

```
int i = 5 + c;          // i = 5 + 1;
```

```
int j = 5 + PINK;       // j = 5 + 2;
```

Переменной перечисляемого типа можно присвоить только перечислитель соответствующего типа:

```
Colors dor = YELLOW;
```

```
c = dor;
```

```
c = 0;    // Ошибка    c != YELLOW
```

```
c = static_cast<Color>(0); // Явное преобразование можно
```

## Операции II

```
enum Color { YELLOW, BLACK, PINK };  
Colors c = BLACK, pig = PINK;
```

Переменные перечисляемого типа часто используются в:

- Операторах ветвления:  

```
if (pig == PINK) ...;  
switch(c){  
    case YELLOW : ...; break;  
    case BLACK  : ...; break;  
}
```
- В качестве возвращаемого значения:  

```
return ERROR_OPENING_FILE;  
return SUCCESS;
```

## Преобразования в перечисление

```
enum Color { YELLOW, BLACK, PINK = 10 };
```

```
Colors c = BLACK;
```

```
c = static_cast<Color>(0);
```

```
c = Color(0);    // в стиле Си
```

```
c = (Color) 0;   // в стиле Си
```

// Допускается, но поведение будет не определено

```
c = static_cast<Color>(5);
```



# Enum class

```
#include <iostream>

enum class Color{
    YELLOW,
    BLACK,
    PINK
};

enum class Feelings{
    SAD,
    ANGRY,
    BLACK
};

int main(){
    std::cout << Color::BLACK;    // Нормально
    std::cout << Feelings::BLACK; // Нормально
    std::cout << BLACK;           // Ошибка
}
```

Тот же Enum, но перечислители объявляются не в той же области видимости, что и Enum, а заворачиваются в пространство имён с именем перечисления.

Множество

# Множество

```
#include <iostream>
#include <set>

int main()
{
    // пустое множество чисел int
    std::set<int> numbers;
    // инициализация
    std::set<int> numbers2 {1, 2, 3, 4, 5};
}
```

# Размер множества

```
#include <iostream>
#include <set>

int main()
{
    std::set<int> numbers{1, 2, 3};

    std::cout << "Empty: " << std::boolalpha
               << numbers.empty() << std::endl;           // Empty: false
    std::cout << "Size: " << numbers.size() << std::endl; // Size: 3
}
```

# Перебор множества

```
#include <iostream>
#include <set>

int main()
{
    std::set<int> numbers{1, 2, 3, 4, 5};

    for (int n : numbers)
        std::cout << n << "\t";
    std::cout << std::endl;
}
```

# Добавление элементов

```
#include <iostream>
#include <set>

int main()
{
    std::set<int> numbers{3, 4, 5};

    numbers.insert(1);
    numbers.insert(2);
    numbers.insert(2);
    numbers.insert(2);
    numbers.insert(6);

    for (int n : numbers)
        std::cout << n << " "; // 1 2 3 4 5 6
    std::cout << std::endl;
}
```

# Удаление элементов

```
#include <iostream>
#include <set>

int main()
{
    std::set<int> numbers{2, 3, 4, 5};

    numbers.erase(1);
    numbers.erase(2);
    numbers.erase(3);

    for (int n : numbers)
        std::cout << n << " "; // 4 5
    std::cout << std::endl;
}
```

# Проверка наличия элемента

```
#include <iostream>
#include <set>

int main()
{
    std::set<int> numbers{2, 3, 4, 5};

    std::cout << "10 in set: " << numbers.count(10) << std::endl; // 10 in set: 0
    std::cout << "2 in set: " << numbers.count(2) << std::endl;    // 2 in set: 1
}
```



# Проверка наличия элемента (C++20)

```
#include <iostream>
#include <set>

int main()
{
    std::set<int> numbers{2, 3, 4, 5};

    std::cout << "10 is in set: " << std::boolalpha
               << numbers.contains(10) << std::endl;    // false
    std::cout << "2 is in set: " << std::boolalpha
               << numbers.contains(2) << std::endl;    // true
}
```

# Неупорядоченное множество unordered\_set

```
#include <iostream>
#include <unordered_set>
#include <set>

int main()
{
    std::set<int> numbers{3, 2, 5, 4};
    numbers.insert(1);
    numbers.insert(6);

    std::unordered_set<int> numbers2{3, 2, 5, 4};
    numbers2.insert(1);
    numbers2.insert(6);

    for (int n : numbers)
        std::cout << n << " "; // 1 2 3 4 5 6
    std::cout << std::endl;

    for (int n : numbers2)
        std::cout << n << " "; // 6 1 4 5 2 3
    std::cout << std::endl;
}
```

# Словарь

## Неупорядоченное множество unordered\_set

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products;
}
```

# Обращение к элементам

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products;
    // установка значений
    products["bread"] = 30;
    products["milk"] = 80;
    products["apple"] = 60;

    // получение значений
    std::cout << "bread\t" << products["bread"] << std::endl;
    std::cout << "milk\t" << products["milk"] << std::endl;
    std::cout << "apple\t" << products["apple"] << std::endl;
}
```

# Перебор элементов

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products;
    // установка значений
    products["bread"] = 30;
    products["milk"] = 80;
    products["apple"] = 60;

    for (const auto& element : products)
        std::cout << element.first << "\t" << element.second << std::endl;
}
```

# Перебор элементов (C++17)

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products;
    // установка значений
    products["bread"] = 30;
    products["milk"] = 80;
    products["apple"] = 60;

    for (const auto& [product, price] : products)
        std::cout << product << "\t" << price << std::endl;
}
```

# Инициализация элементов

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products
    {
        std::pair<std::string, unsigned>{"bread", 30},
        std::pair<std::string, unsigned>{"milk", 80},
        std::pair<std::string, unsigned>{"apple", 60}
    };
}
```



# Инициализация элементов

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products
    {
        {"bread", 30},
        {"milk", 80},
        {"apple", 60}
    };
}
```

# Удаление элементов

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products
    {
        {"bread", 30}, {"milk", 80}, {"apple", 60}
    };

    products.erase("milk");    // удаляем элемент с ключом "milk"

    for (const auto& [product, price] : products)
        std::cout << product << "\t" << price << std::endl;

    // apple    60
    // bread    30
}
```

# Размер словаря

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products
    {
        {"bread", 30}, {"milk", 80}, {"apple", 60}
    };

    std::cout << "count: "
                << products.size() << std::endl; // count: 3
    std::cout << "is empty: "
                << std::boolalpha << products.empty() << std::endl; // is empty: false
}
```

# Проверка наличия элемента

```
#include <iostream>
#include <map>

int main()
{
    std::map<std::string, unsigned> products
    {
        {"bread", 30}, {"milk", 80}, {"apple", 60}
    };
    std::cout << "Apple\t" << products.count("apple")<< std::endl;    // Apple  1
    std::cout << "Orange\t" << products.count("orange")<< std::endl; // Orange  0

    std::cout << "Apple\t" << std::boolalpha
                << products.contains("apple")<< std::endl;    // Apple  true
    std::cout << "Orange\t" << std::boolalpha
                << products.contains("orange")<< std::endl;    // Orange false
}
```

# Неупорядоченные словари

```
#include <iostream>
#include <unordered_map>

int main()
{
    std::unordered_map<std::string, unsigned> products
    {
        {"bread", 30}, {"milk", 80}, {"apple", 60}
    };

    for (const auto& [product, price] : products)
        std::cout << product << "\t" << price << std::endl;
}
```