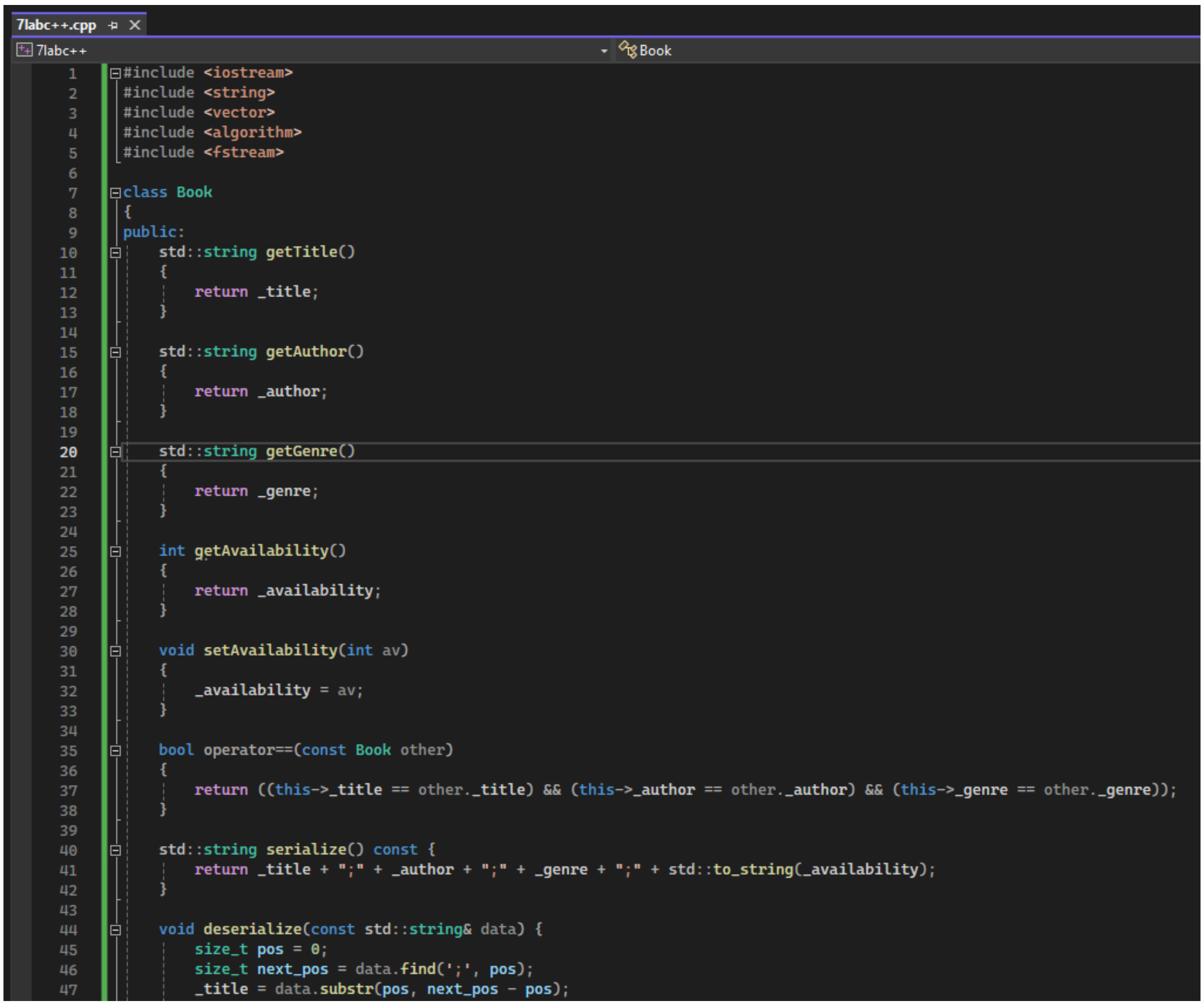
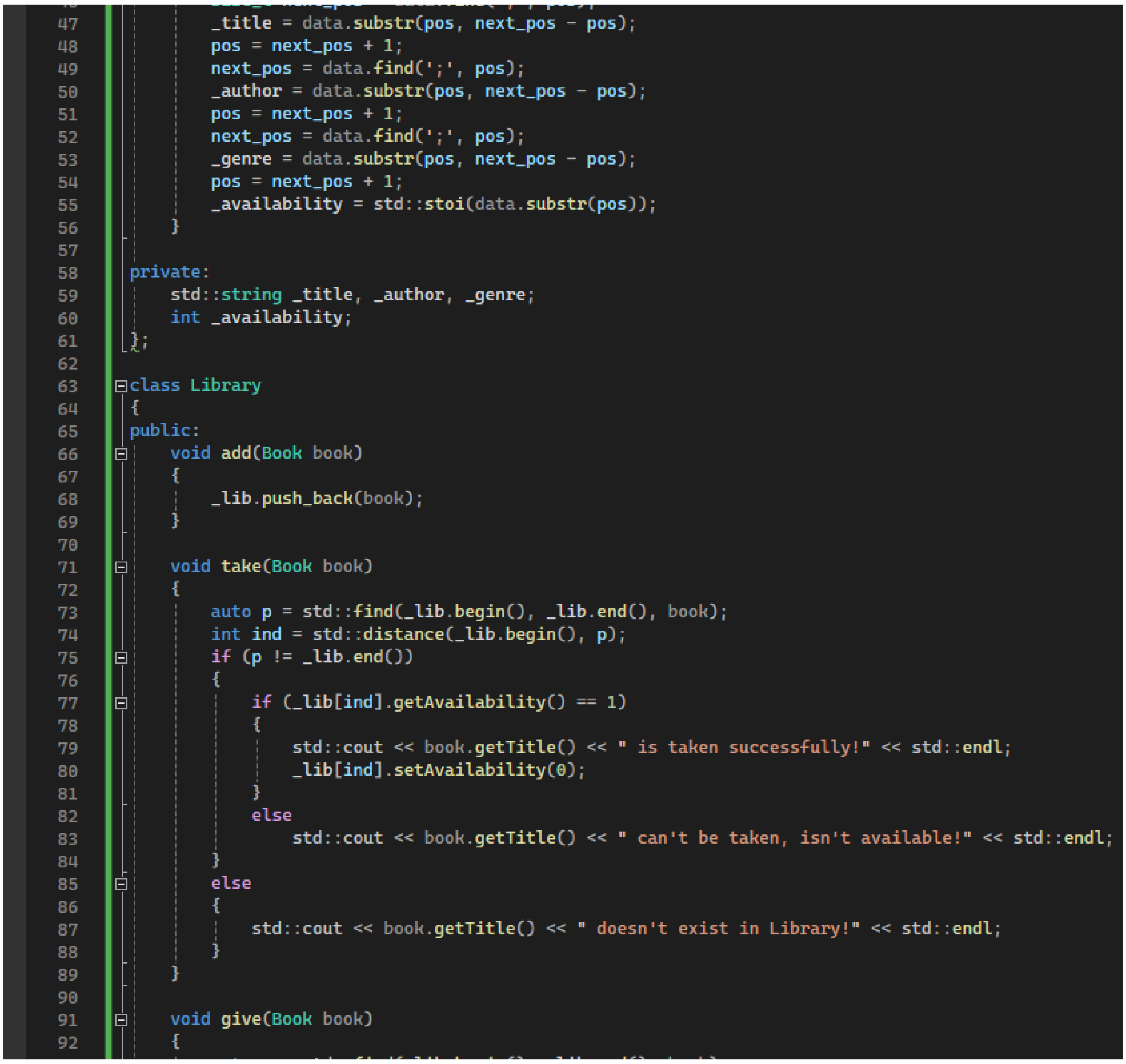
Моя программа:



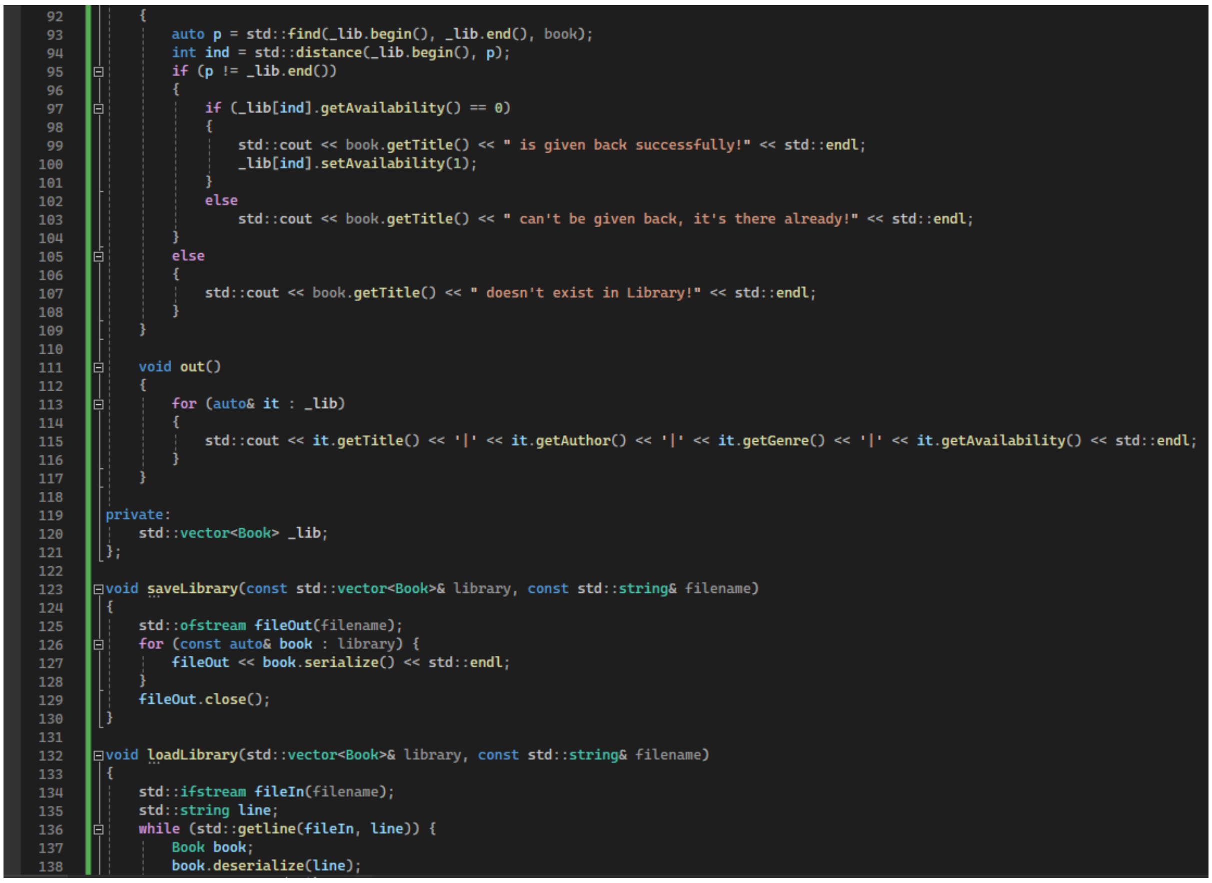
7labc++.cpp

7labc++ | Book

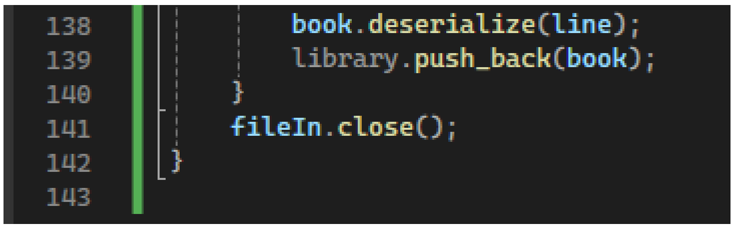
```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <fstream>

class Book
{
public:
    std::string getTitle()
    {
        return _title;
    }

    std::string getAuthor()
    {
        return _author;
    }

    std::string getGenre()
    {
        return _genre;
    }

    int getAvailability()
    {
        return _availability;
    }

    void setAvailability(int av)
    {
        _availability = av;
    }

    bool operator==(const Book other)
    {
        return ((this->_title == other._title) && (this->_author == other._author) && (this->_genre == other._genre));
    }

    std::string serialize() const {
        return _title + ";" + _author + ";" + _genre + ";" + std::to_string(_availability);
    }

    void deserialize(const std::string& data) {
        size_t pos = 0;
        size_t next_pos = data.find(';', pos);
        _title = data.substr(pos, next_pos - pos);
```



```cpp
        _title = data.substr(pos, next_pos - pos);
        pos = next_pos + 1;
        next_pos = data.find(';', pos);
        _author = data.substr(pos, next_pos - pos);
        pos = next_pos + 1;
        next_pos = data.find(';', pos);
        _genre = data.substr(pos, next_pos - pos);
        pos = next_pos + 1;
        _availability = std::stoi(data.substr(pos));
    }

private:
    std::string _title, _author, _genre;
    int _availability;
};

class Library
{
public:
    void add(Book book)
    {
        _lib.push_back(book);
    }

    void take(Book book)
    {
        auto p = std::find(_lib.begin(), _lib.end(), book);
        int ind = std::distance(_lib.begin(), p);
        if (p != _lib.end())
        {
            if (_lib[ind].getAvailability() == 1)
            {
                std::cout << book.getTitle() << " is taken successfully!" << std::endl;
                _lib[ind].setAvailability(0);
            }
            else
                std::cout << book.getTitle() << " can't be taken, isn't available!" << std::endl;
        }
        else
        {
            std::cout << book.getTitle() << " doesn't exist in Library!" << std::endl;
        }
    }

    void give(Book book)
    {
```

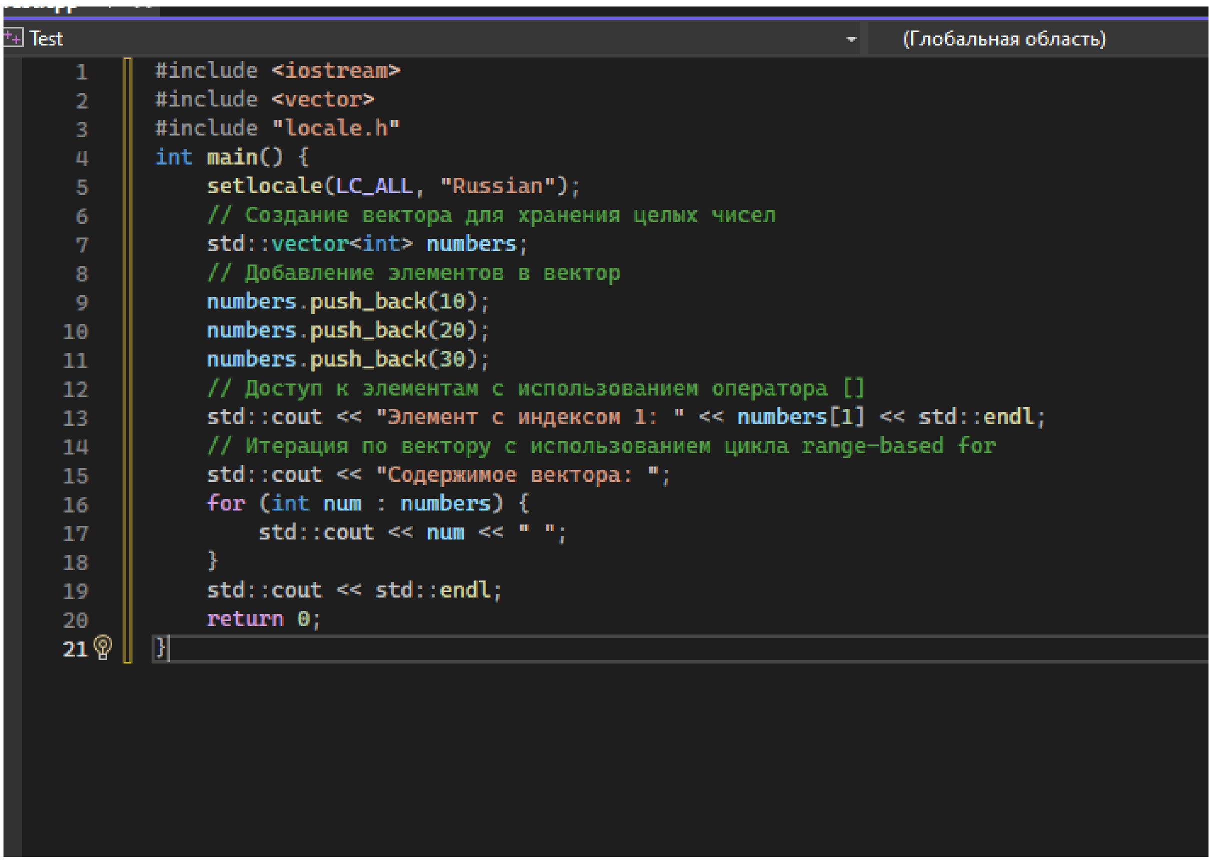


```cpp
    {
        auto p = std::find(_lib.begin(), _lib.end(), book);
        int ind = std::distance(_lib.begin(), p);
        if (p != _lib.end())
        {
            if (_lib[ind].getAvailability() == 0)
            {
                std::cout << book.getTitle() << " is given back successfully!" << std::endl;
                _lib[ind].setAvailability(1);
            }
            else
                std::cout << book.getTitle() << " can't be given back, it's there already!" << std::endl;
        }
        else
        {
            std::cout << book.getTitle() << " doesn't exist in Library!" << std::endl;
        }
    }

    void out()
    {
        for (auto& it : _lib)
        {
            std::cout << it.getTitle() << '|' << it.getAuthor() << '|' << it.getGenre() << '|' << it.getAvailability() << std::endl;
        }
    }

private:
    std::vector<Book> _lib;
};

void saveLibrary(const std::vector<Book>& library, const std::string& filename)
{
    std::ofstream fileOut(filename);
    for (const auto& book : library) {
        fileOut << book.serialize() << std::endl;
    }
    fileOut.close();
}

void loadLibrary(std::vector<Book>& library, const std::string& filename)
{
    std::ifstream fileIn(filename);
    std::string line;
    while (std::getline(fileIn, line)) {
        Book book;
        book.deserialize(line);
```

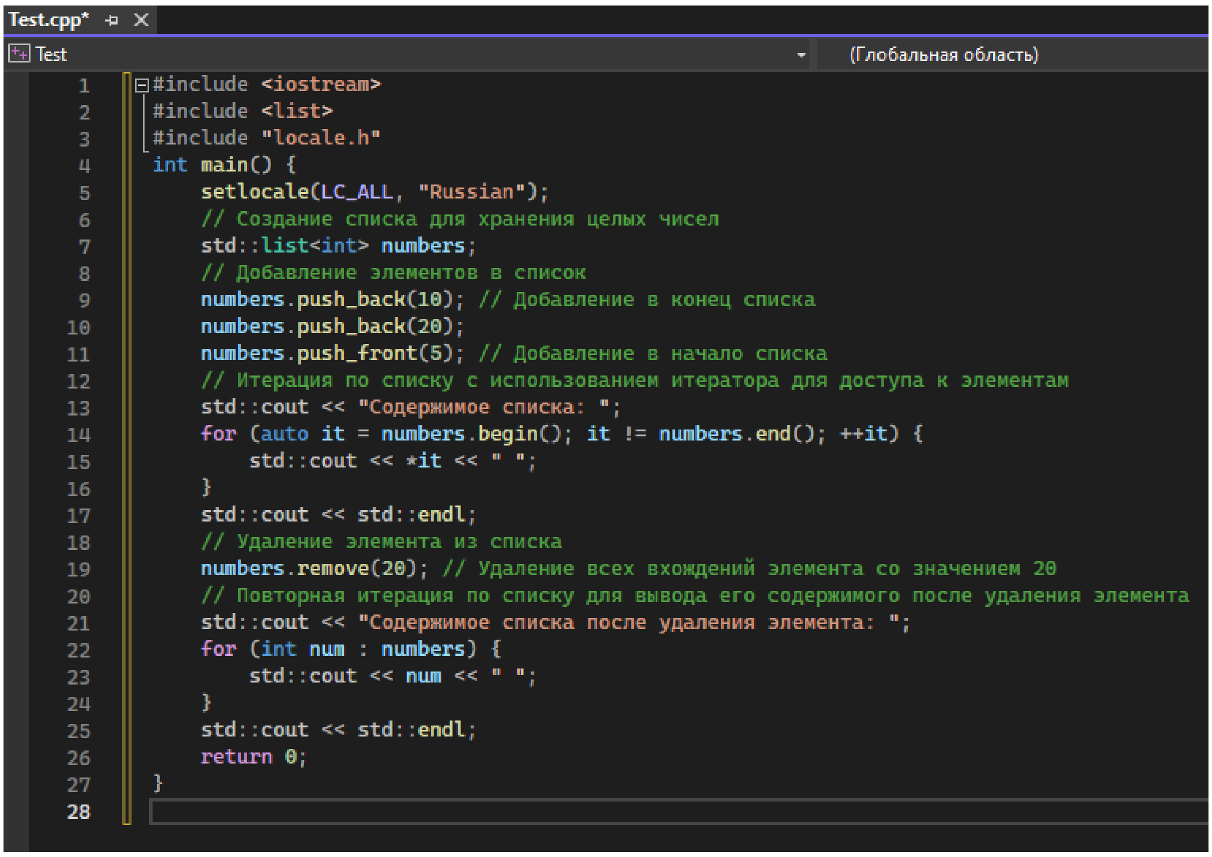


```cpp
        book.deserialize(line);
        library.push_back(book);
    }
    fileIn.close();
}
```

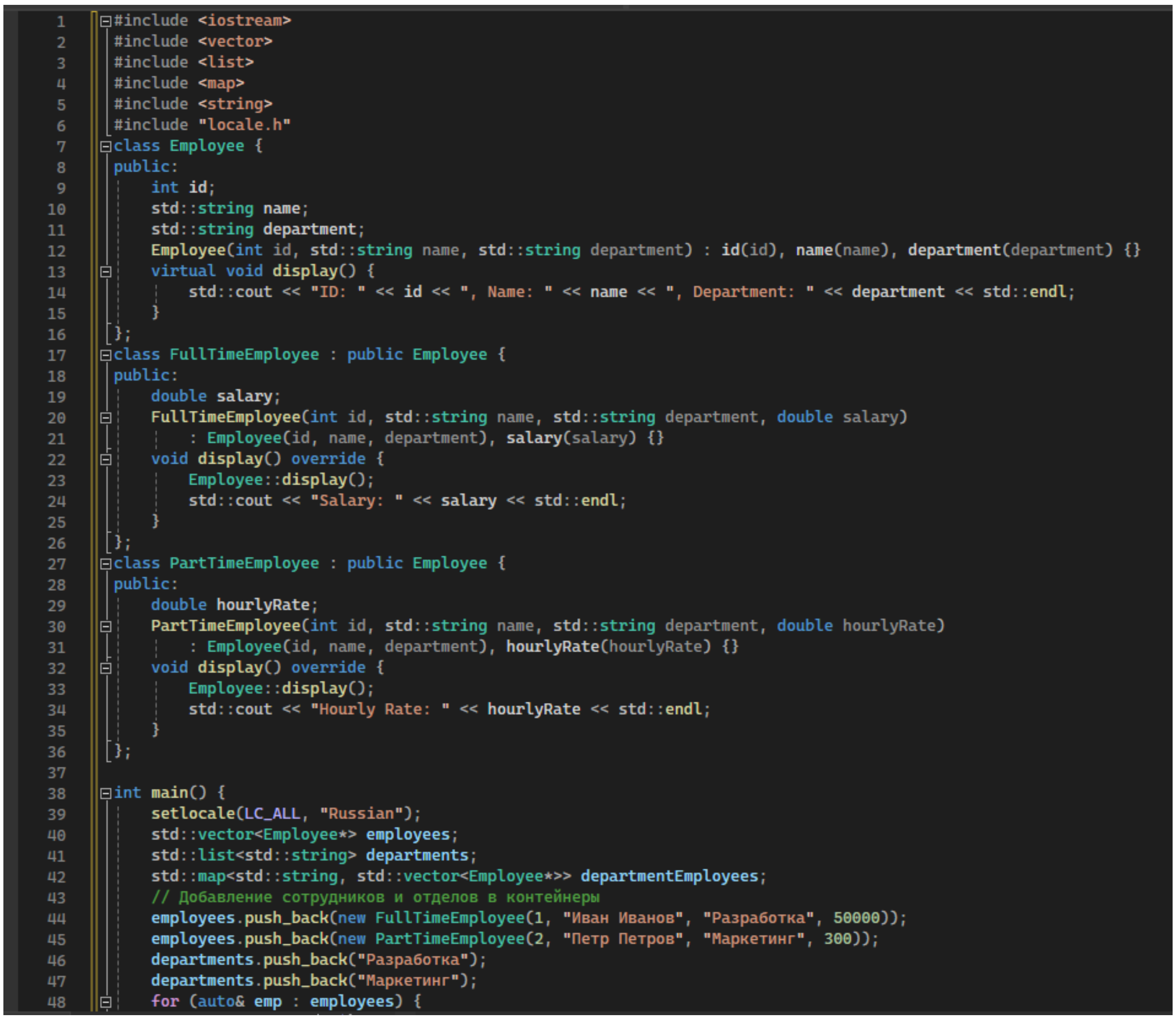
Примеры:



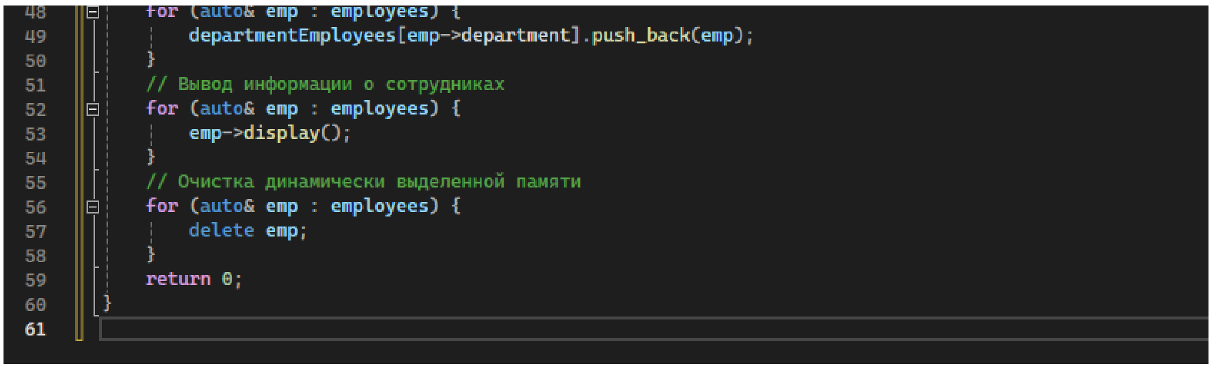
```cpp
#include <iostream>
#include <vector>
#include "locale.h"
int main() {
    setlocale(LC_ALL, "Russian");
    // Создание вектора для хранения целых чисел
    std::vector<int> numbers;
    // Добавление элементов в вектор
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_back(30);
    // Доступ к элементам с использованием оператора []
    std::cout << "Элемент с индексом 1: " << numbers[1] << std::endl;
    // Итерация по вектору с использованием цикла range-based for
    std::cout << "Содержимое вектора: ";
    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    return 0;
}
```



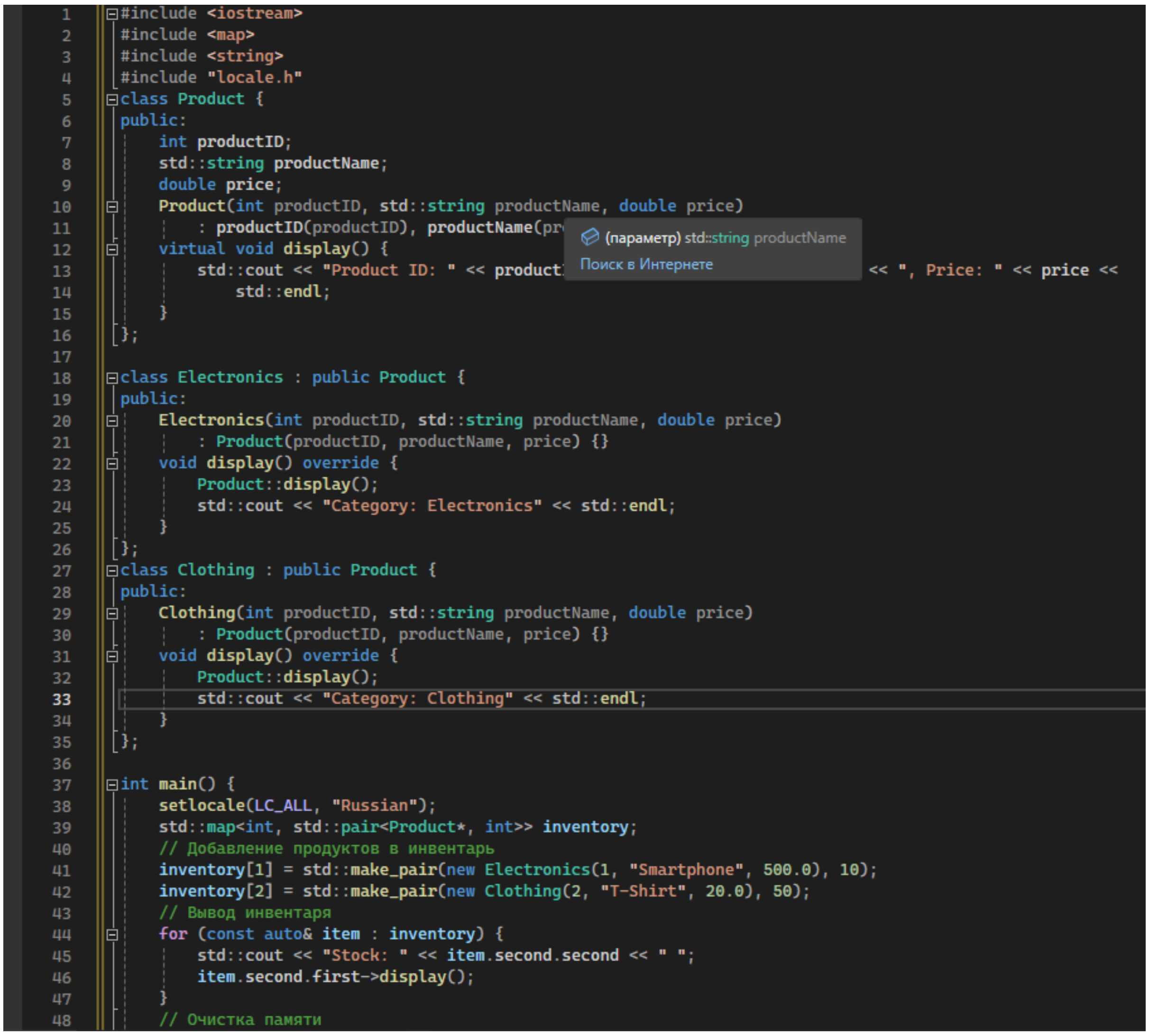
```cpp
#include <iostream>
#include <list>
#include "locale.h"
int main() {
    setlocale(LC_ALL, "Russian");
    // Создание списка для хранения целых чисел
    std::list<int> numbers;
    // Добавление элементов в список
    numbers.push_back(10); // Добавление в конец списка
    numbers.push_back(20);
    numbers.push_front(5); // Добавление в начало списка
    // Итерация по списку с использованием итератора для доступа к элементам
    std::cout << "Содержимое списка: ";
    for (auto it = numbers.begin(); it != numbers.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;
    // Удаление элемента из списка
    numbers.remove(20); // Удаление всех вхождений элемента со значением 20
    // Повторная итерация по списку для вывода его содержимого после удаления элемента
    std::cout << "Содержимое списка после удаления элемента: ";
    for (int num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;
    return 0;
}
```



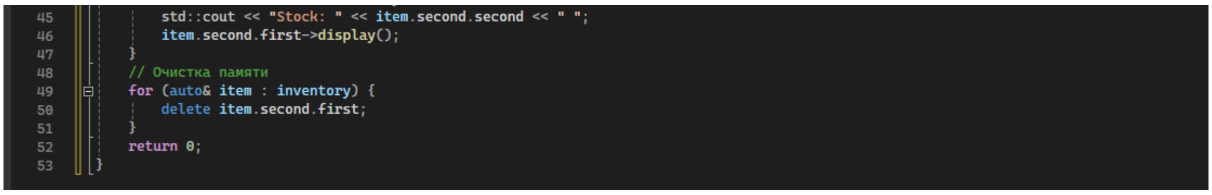
```cpp
#include <iostream>
#include <vector>
#include <list>
#include <map>
#include <string>
#include "locale.h"
class Employee {
public:
    int id;
    std::string name;
    std::string department;
    Employee(int id, std::string name, std::string department) : id(id), name(name), department(department) {}
    virtual void display() {
        std::cout << "ID: " << id << ", Name: " << name << ", Department: " << department << std::endl;
    }
};
class FullTimeEmployee : public Employee {
public:
    double salary;
    FullTimeEmployee(int id, std::string name, std::string department, double salary)
        : Employee(id, name, department), salary(salary) {}
    void display() override {
        Employee::display();
        std::cout << "Salary: " << salary << std::endl;
    }
};
class PartTimeEmployee : public Employee {
public:
    double hourlyRate;
    PartTimeEmployee(int id, std::string name, std::string department, double hourlyRate)
        : Employee(id, name, department), hourlyRate(hourlyRate) {}
    void display() override {
        Employee::display();
        std::cout << "Hourly Rate: " << hourlyRate << std::endl;
    }
};

int main() {
    setlocale(LC_ALL, "Russian");
    std::vector<Employee*> employees;
    std::list<std::string> departments;
    std::map<std::string, std::vector<Employee*>> departmentEmployees;
    // Добавление сотрудников и отделов в контейнеры
    employees.push_back(new FullTimeEmployee(1, "Иван Иванов", "Разработка", 50000));
    employees.push_back(new PartTimeEmployee(2, "Петр Петров", "Маркетинг", 300));
    departments.push_back("Разработка");
    departments.push_back("Маркетинг");
    for (auto& emp : employees) {
```



```cpp
    for (auto& emp : employees) {
        departmentEmployees[emp->department].push_back(emp);
    }
    // Вывод информации о сотрудниках
    for (auto& emp : employees) {
        emp->display();
    }
    // Очистка динамически выделенной памяти
    for (auto& emp : employees) {
        delete emp;
    }
    return 0;
}
```



```cpp
#include <iostream>
#include <map>
#include <string>
#include "locale.h"
class Product {
public:
    int productID;
    std::string productName;
    double price;
    Product(int productID, std::string productName, double price)
        : productID(productID), productName(pr
    virtual void display() {
        std::cout << "Product ID: " << product                              << ", Price: " << price <<
            std::endl;
    }
};

class Electronics : public Product {
public:
    Electronics(int productID, std::string productName, double price)
        : Product(productID, productName, price) {}
    void display() override {
        Product::display();
        std::cout << "Category: Electronics" << std::endl;
    }
};
class Clothing : public Product {
public:
    Clothing(int productID, std::string productName, double price)
        : Product(productID, productName, price) {}
    void display() override {
        Product::display();
        std::cout << "Category: Clothing" << std::endl;
    }
};

int main() {
    setlocale(LC_ALL, "Russian");
    std::map<int, std::pair<Product*, int>> inventory;
    // Добавление продуктов в инвентарь
    inventory[1] = std::make_pair(new Electronics(1, "Smartphone", 500.0), 10);
    inventory[2] = std::make_pair(new Clothing(2, "T-Shirt", 20.0), 50);
    // Вывод инвентаря
    for (const auto& item : inventory) {
        std::cout << "Stock: " << item.second.second << " ";
        item.second.first->display();
    }
    // Очистка памяти
```



```cpp
        std::cout << "Stock: " << item.second.second << " ";
        item.second.first->display();
    }
    // Очистка памяти
```

```cpp
    for (auto& item : inventory) {
        delete item.second.first;
    }
    return 0;
}
```