

Программирование на C++



Минцифры
России

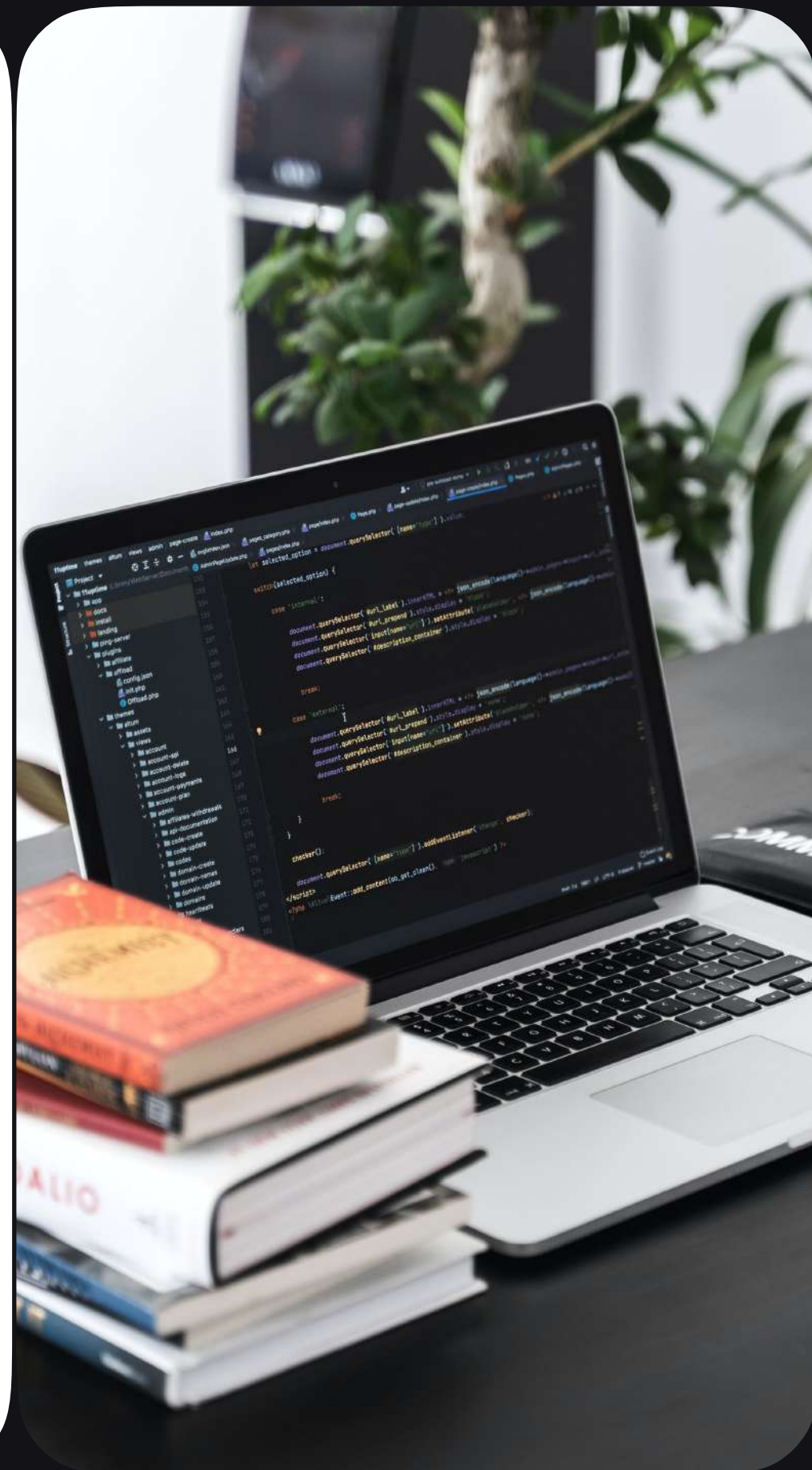
UCHi **DOMA**

20.35
УНИВЕРСИТЕТ

Урок 4 Модуль 4

Разработка динамической библиотеки

Полезные материалы



Цели урока



отработать на практике
написание алгоритмов
с использованием
динамических библиотек
на C++



Создать клиентское
приложение, которое
использует библиотеку DLL



Библиотека

Динамическая библиотека (также называемая **shared library**, «общая библиотека») состоит из подпрограмм, которые загружаются в приложение во время выполнения.

Когда компилируется программа, использующая динамическую библиотеку, библиотека не становится частью исполняемого файла — она остается отдельной единицей. В Windows динамические библиотеки обычно имеют расширение **.dll** (dynamic link library, библиотека динамической компоновки), а в Linux — расширение **.so** (shared object, общий объект).

Преимущества динамических библиотек



Многие программы могут совместно использовать одну копию библиотеки, что экономит место.



Динамическую библиотеку можно обновить до более новой версии без замены всех исполняемых файлов, которые ее используют.

Связывание исполняемого файла с библиотекой DLL

Исполняемый файл можно связать с библиотекой DLL (загрузить ее) одним из двух способов:



Неявное связывание



Явное связывание

Явное связывание

Явное связывание — операционная система загружает библиотеку DLL по запросу во время выполнения. Исполняемый файл, который использует библиотеку DLL, должен явно загружать и выгружать ее. Кроме того, в нем должен быть настроен указатель функции для доступа к каждой используемой функции из библиотеки DLL.

В отличие от вызовов функций в статически скомпонованной или неявно связанной библиотеке DLL, при работе с явно связанной DLL исполняемый файл клиента должен вызывать экспортированные функции с помощью указателей функций. Процесс явного связывания также иногда называют **динамической загрузкой** или **динамической компоновкой** времени выполнения.

Явное связывание применяется, когда



Имя библиотеки DLL, которую необходимо загружать, становится известно приложению только во время выполнения. Например, приложение может получать имя библиотеки DLL и экспортированные функции из файла конфигурации во время запуска.



Если при запуске не удастся найти нужную библиотеку DLL, процесс, в котором используется неявное связывание, завершается операционной системой. Процесс с явным связыванием в таких ситуациях не завершается и может попытаться восстановиться после ошибки. Например, процесс может уведомить пользователя об ошибке и запросить указать другой путь к библиотеке DLL.



Процесс с неявным связыванием также завершается, если в любой из связанных библиотек DLL функция DllMain завершается сбоем. Процесс с явным связыванием в таких ситуациях не завершается.



Приложение, в котором применяется неявное связывание с множеством библиотек DLL, может долго запускаться, поскольку операционная система Windows при загрузке приложения загружает все библиотеки DLL. Чтобы ускорить процесс запуска приложения, можно выполнять неявное связывание только с теми библиотеками DLL, которые необходимы непосредственно после загрузки. Другие библиотеки DLL могут загружаться позднее по мере необходимости посредством явного связывания.



При явном связывании приложению не требуется библиотека импорта.

Явное связывание с библиотекой DLL

Чтобы использовать библиотеку DLL посредством явного связывания, необходимо вызвать функцию для явной загрузки библиотеки DLL во время выполнения. Для явного связывания с библиотекой DLL приложение должно выполнить следующие действия:



Вызвать LoadLibraryEx или аналогичную функцию для загрузки библиотеки DLL и получения дескриптора модуля.



Вызвать GetProcAddress, чтобы получить указатель для каждой экспортированной функции, которую вызывает приложение. Поскольку приложения вызывают функции DLL с помощью указателя, компилятор не создает внешние ссылки и нет необходимости связываться с библиотекой импорта. Тем не менее необходимо использовать инструкцию **typedef** или **using**, определяющую сигнатуру вызова для вызываемых экспортированных функций.



По завершении работы с библиотекой DLL вызовите FreeLibrary.

Практика



С помощью интегрированной среды разработки (IDE) Visual Studio создадим собственную библиотеку динамической компоновки (DLL).

В библиотеке будут реализованы основные арифметические функции (+, −, *, /)

В ходе практики создадим два решения:

Первое решение создает библиотеку DLL, а второе — клиентское приложение.

Клиентское приложение использует **явную компоновку**.

Создание проекта динамической библиотеки

Для создания библиотеки будем использовать MS Visual Studio



Создаем библиотеку

Создание проекта

Последние шаблоны проектов

Консольное приложение

C++

Пустой проект

C++

Поиск шаблонов (ALT+"B")

Все языки

Все платформы

Все типы проектов



Пустой проект

Начать с нуля, используя C++ для Windows. Начальные файлы отсутствуют.

C++

Windows

Консоль



Консольное приложение

Выполнить код в терминале Windows. По умолчанию выводится фраза "Hello World".

C++

Windows

Консоль



Проект CMake

Создавайте современные кроссплатформенные приложения C++, не зависящие от файлов SLN или VCXPROJ.

C++

Windows

Linux

Консоль



Мастер классических приложений Windows

Создание собственного приложения Windows с помощью мастера.

C++

Windows

Рабочий стол

Консоль

Библиотека



Классическое приложение Windows

Проект приложения с графическим интерфейсом в Windows.

C++

Windows

Рабочий стол

Назад

Далее

Создадим новый пустой проект:

Файл → Создать → Проект

Название проекта

Настроить новый проект

Пустой проект

C++

Windows

Консоль

Имя проекта

DynemicLib

Расположение

C:\Practice\

Имя решения ⓘ

DynemicLib

☐ Поместить решение и проект в одном каталоге

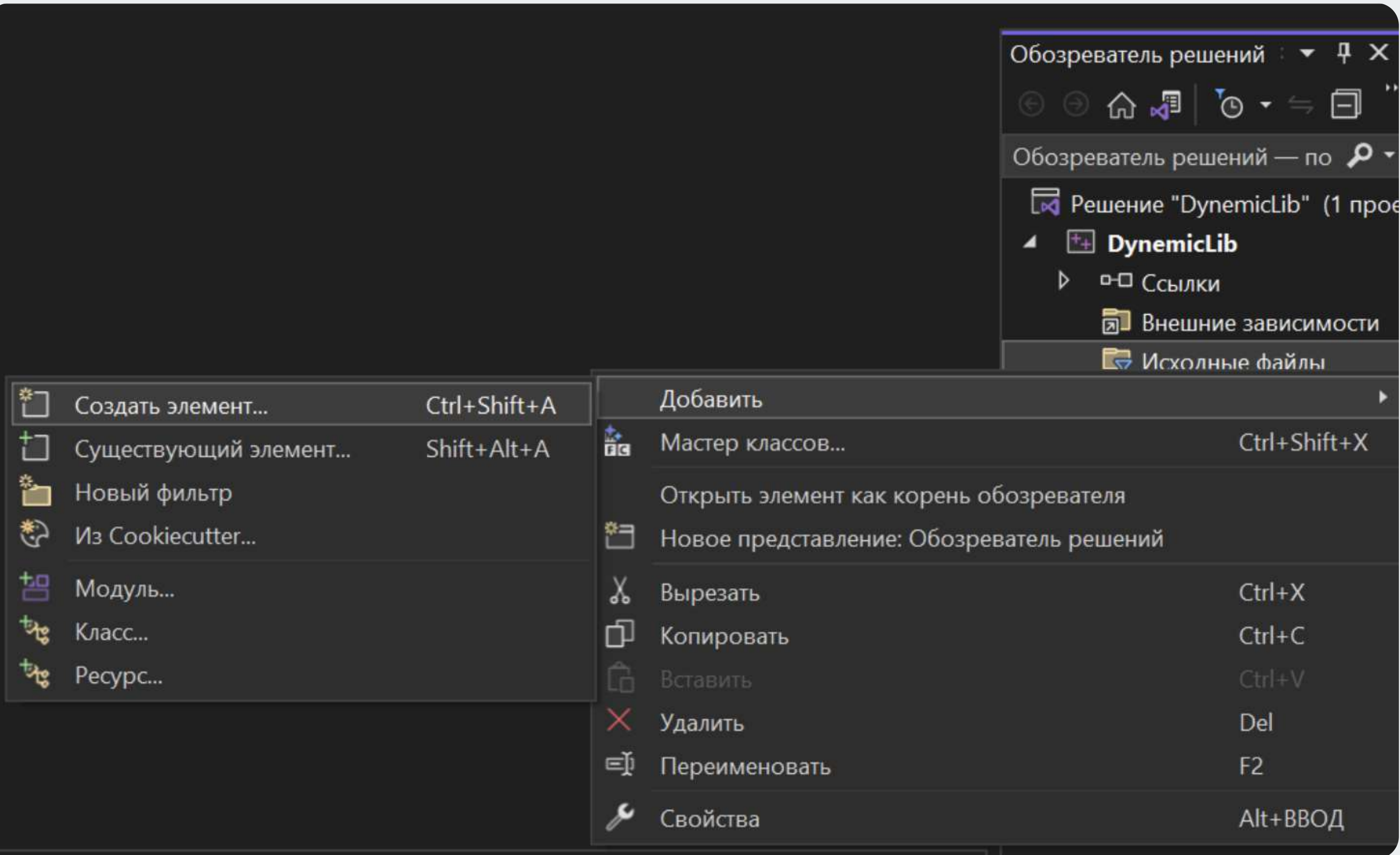
Назад

Создать

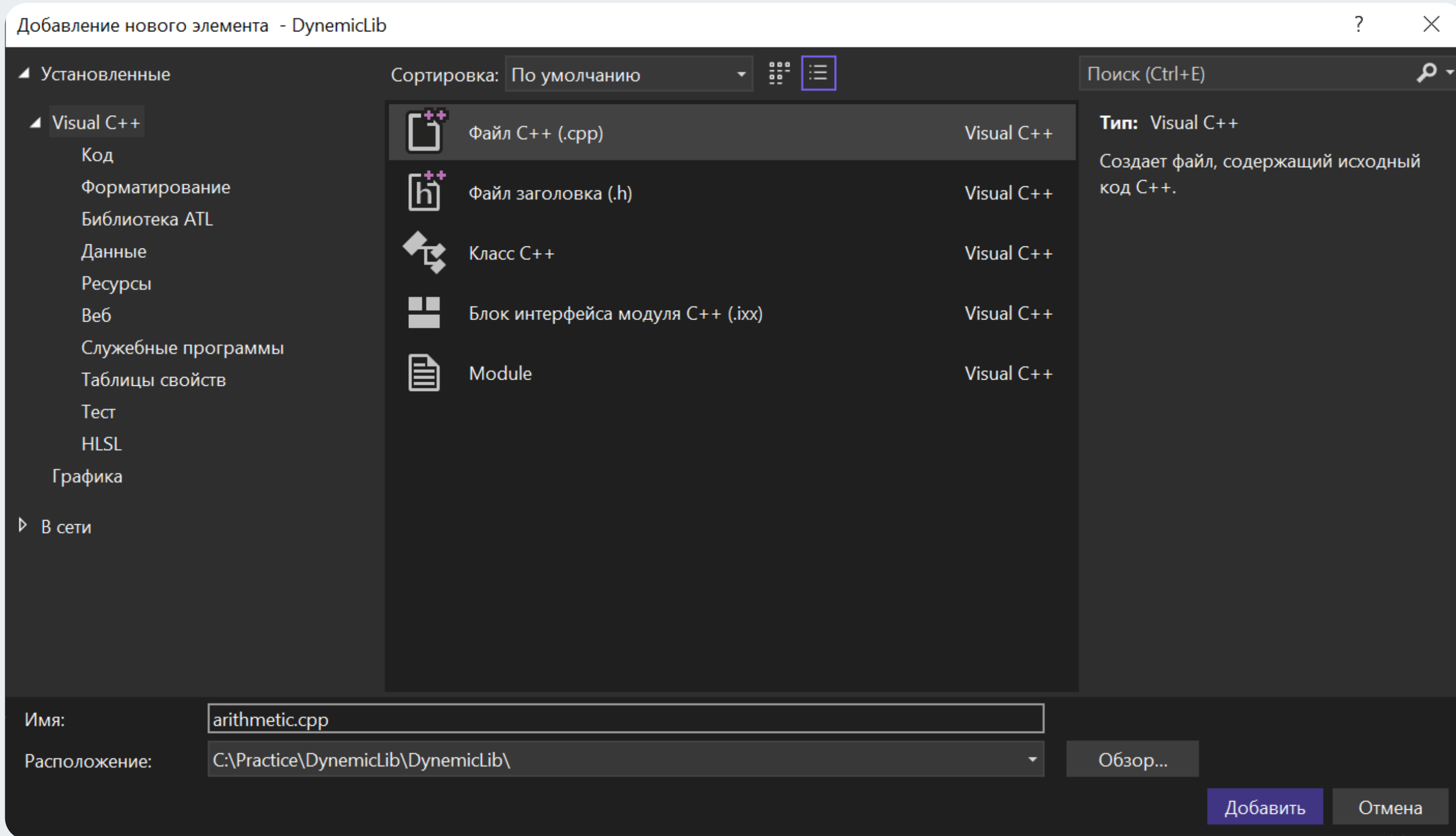
Назовите проект «DynemicLib».

Нажмите кнопку «Создать»

Добавьте файл в проект



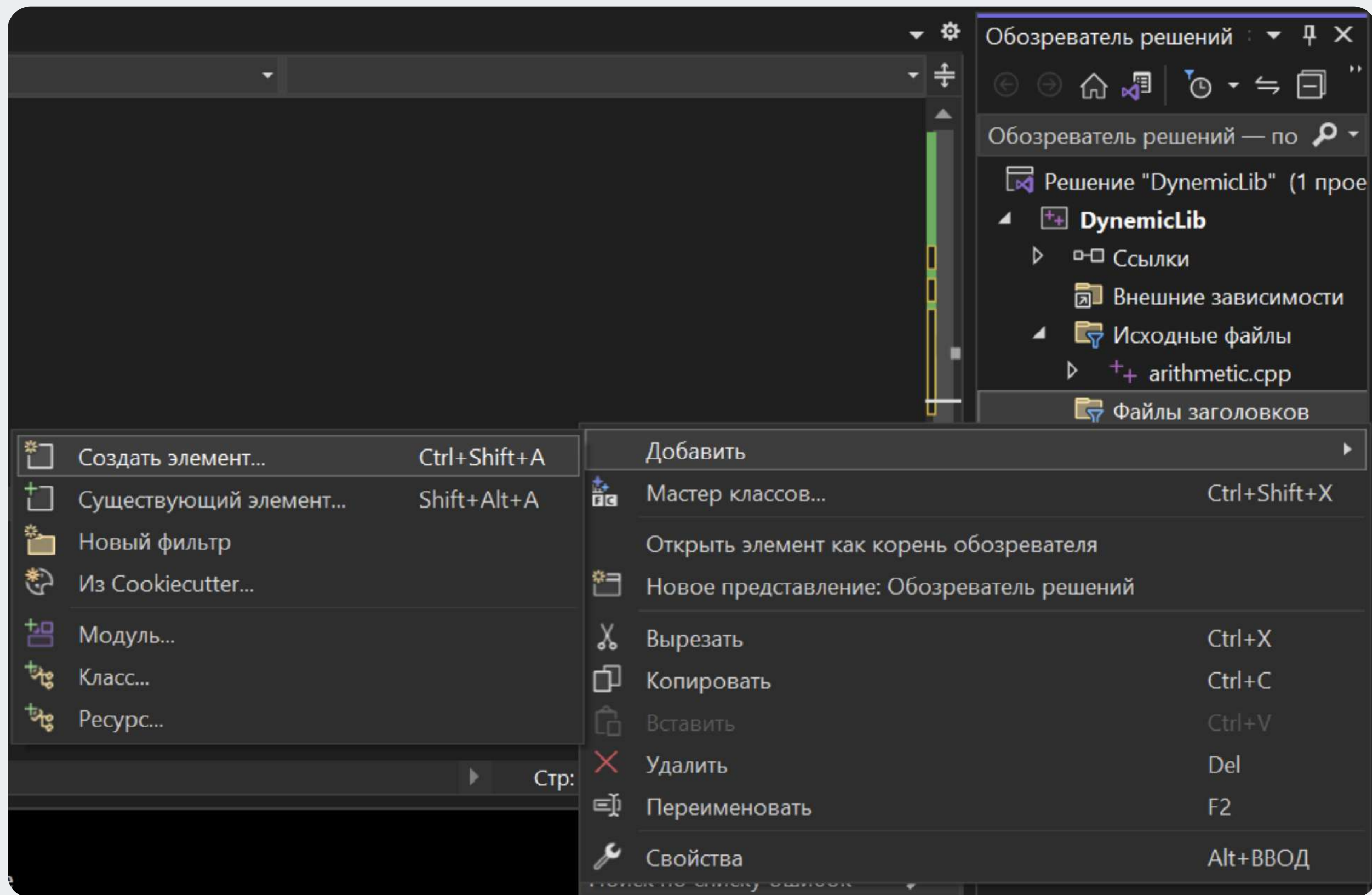
Добавим файл кода, назовем его «arithmetic.cpp»



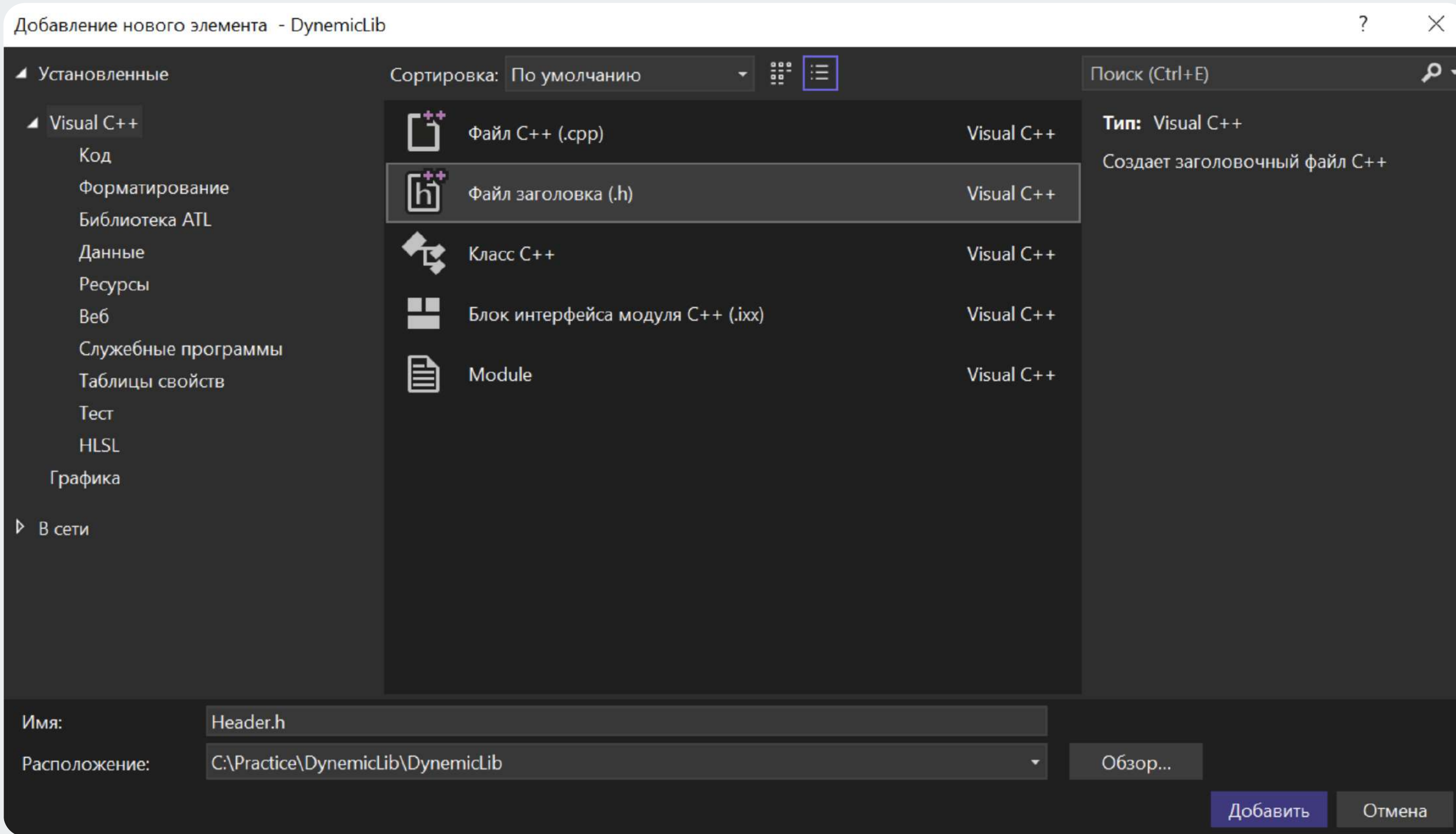
Создаём функции для арифметических операций

```
+ DynamicLib (Глобальная область)
1 extern "C" __declspec(dllexport) int Sum(int a, int b)
2 {
3     return a + b;
4 }
5 extern "C" __declspec(dllexport) int Min(int a, int b)
6 {
7     return a - b;
8 }
9 extern "C" __declspec(dllexport) int Mult(int a, int b)
10 {
11     return a * b;
12 }
13 extern "C" __declspec(dllexport) double Div(int a, int b)
14 {
15     return a / b;
16 }
```

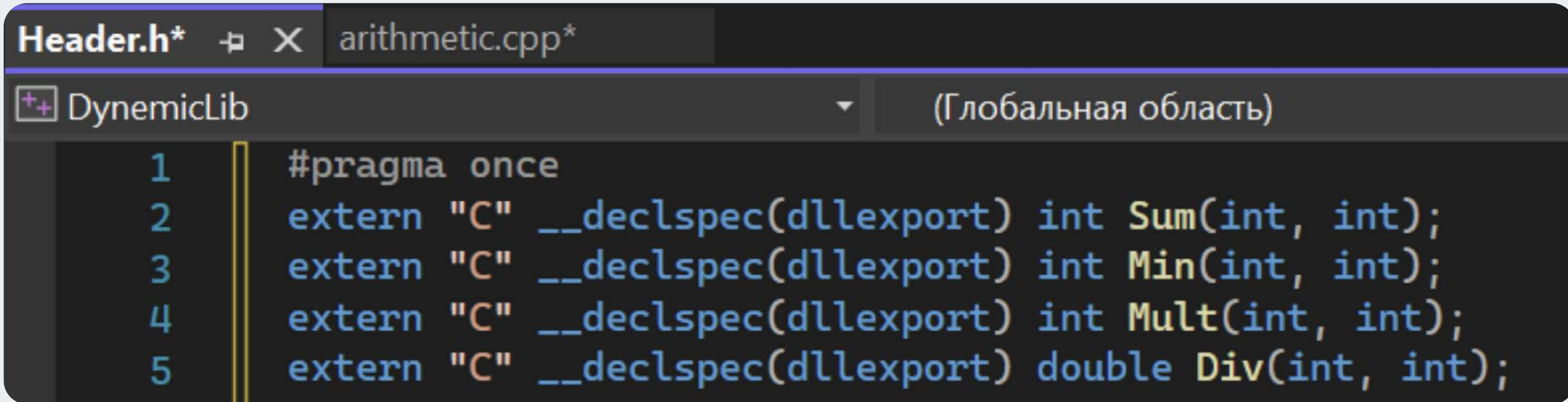
Добавим в проект заголовочный файл



Создаем заголовочный файл «Header.h»



Объявляем функции в заголовочном файле

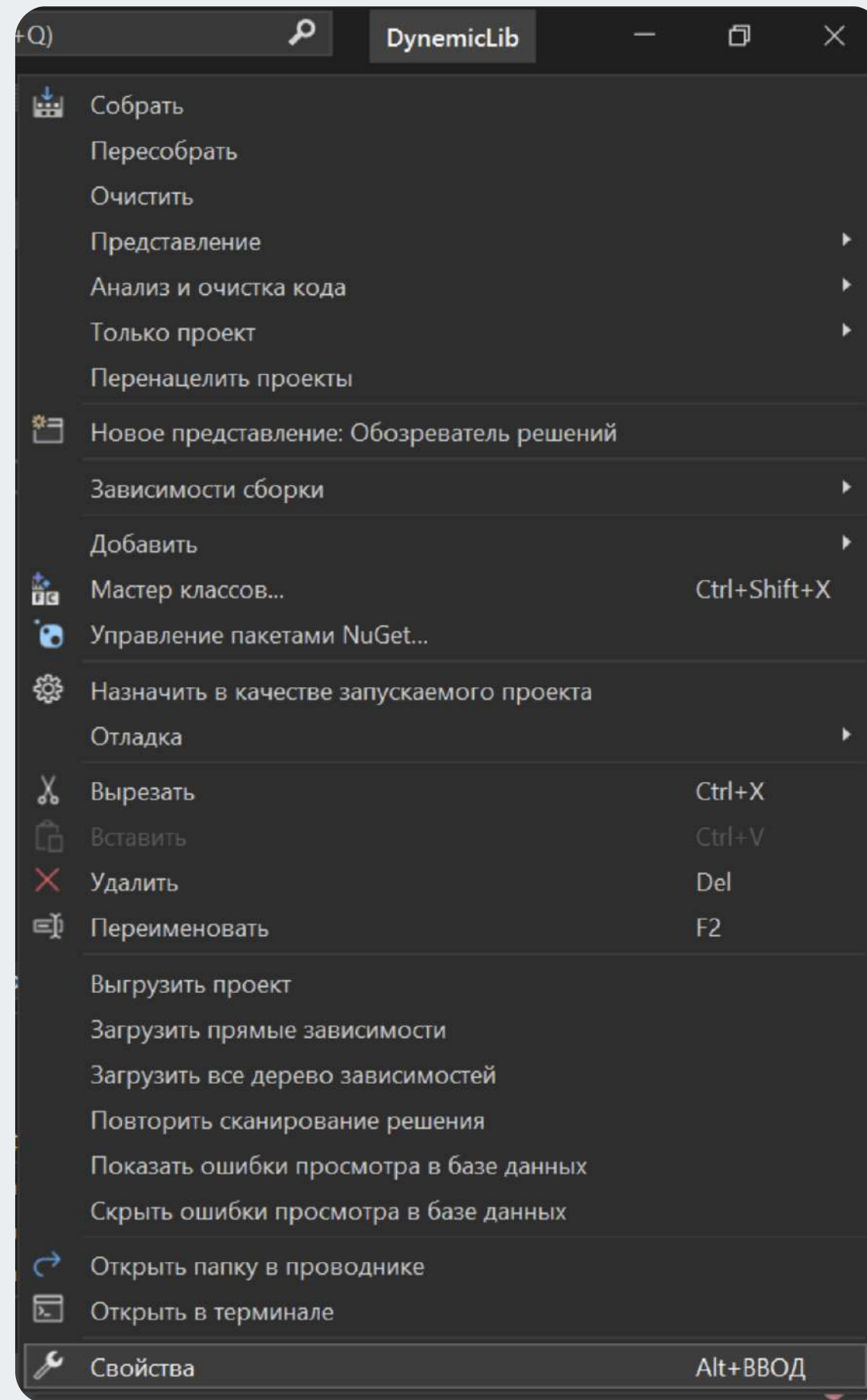
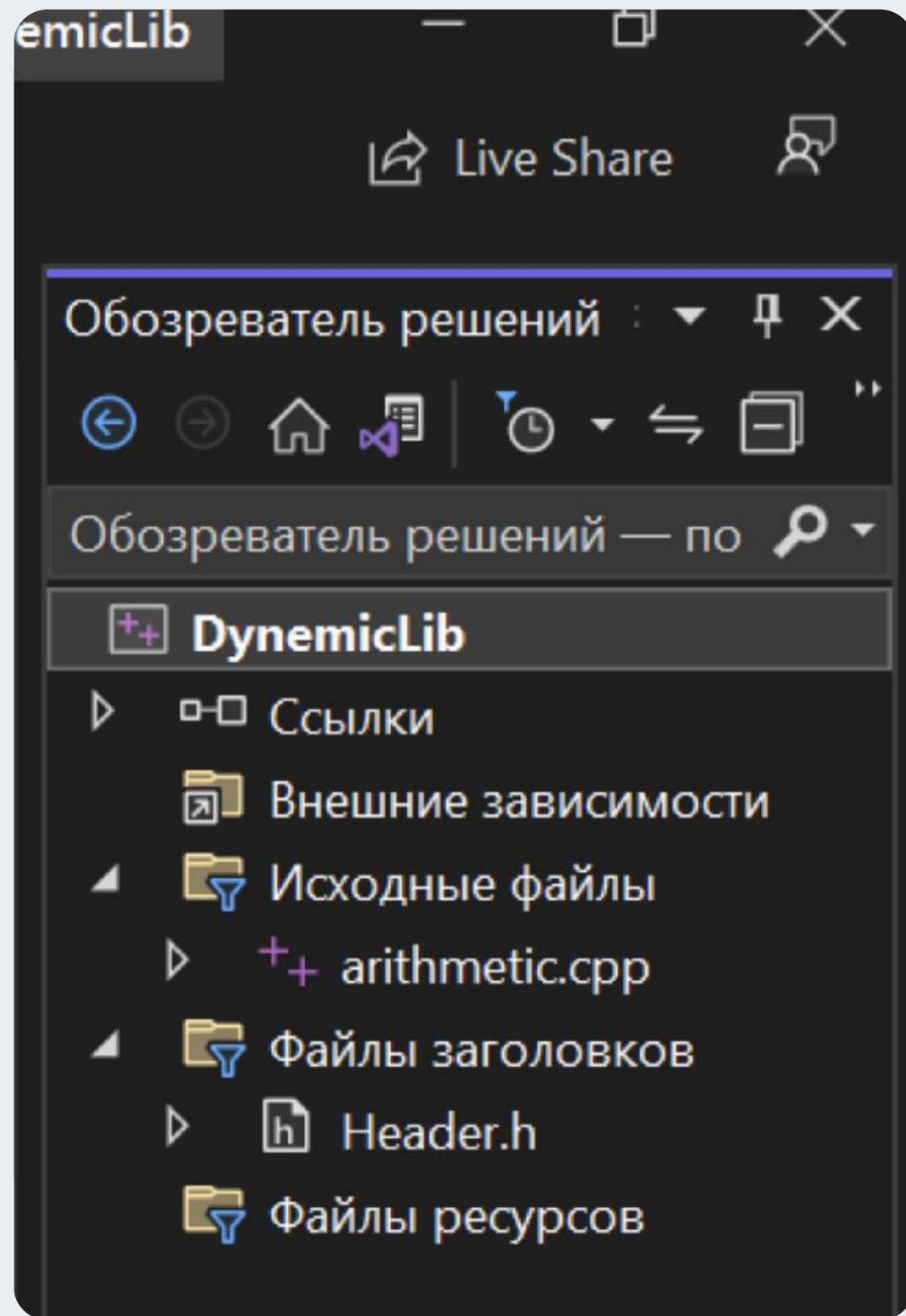


The image shows a code editor window with two tabs: 'Header.h*' and 'arithmetic.cpp*'. The 'arithmetic.cpp*' tab is active. The editor shows a C++ header file with the following code:

```
1 #pragma once
2 extern "C" __declspec(dllexport) int Sum(int, int);
3 extern "C" __declspec(dllexport) int Min(int, int);
4 extern "C" __declspec(dllexport) int Mult(int, int);
5 extern "C" __declspec(dllexport) double Div(int, int);
```

The code is written in a dark-themed editor. The line numbers 1 through 5 are on the left. The code uses color syntax highlighting: keywords like 'extern', 'C', and 'declspec' are in blue, 'dllexport' is in orange, and function names 'Sum', 'Min', 'Mult', and 'Div' are in yellow. The return types 'int' and 'double' are in light blue. The parameter types 'int' are in light blue. The closing semicolons are in light blue. The editor has a toolbar at the top with icons for undo, redo, and search. The status bar at the bottom shows 'DynemicLib' and '(Глобальная область)'.

Настроим свойства проекта



Выберите тип конфигурации — «Динамическая библиотека (.dll)»

Страницы свойств DynemicLib?×

Конфигурация: Активная (Debug) Платформа: Активная (x64) Диспетчер конфигураций...

▲ Свойства конфигурации

Общие

Дополнительно

Отладка

Каталоги VC++

▸ C/C++

▸ Компоновщик

▸ Инструмент манифеста

▸ Генератор XML-документов

▸ Информация об исходном к

▸ События сборки

▸ Настраиваемый этап сборки

▸ Code Analysis

▼ Общие свойства

Выходной каталог	\$(SolutionDir)\$(Platform)\\$(Configuration)\
Промежуточный каталог	\$(Platform)\\$(Configuration)\
Имя целевого объекта	\$(ProjectName)
Тип конфигурации	Динамическая библиотека (.dll)
Версия пакета S	Файл makefile
Набор инструме	Приложение (.exe)
Стандарт языка C	Динамическая библиотека (.dll)
Стандарт языка C	Статическая библиотека (.lib)
	Служебная программа
	<наследовать от родителя или от значений по умолчанию для проекта>

Тип конфигурации

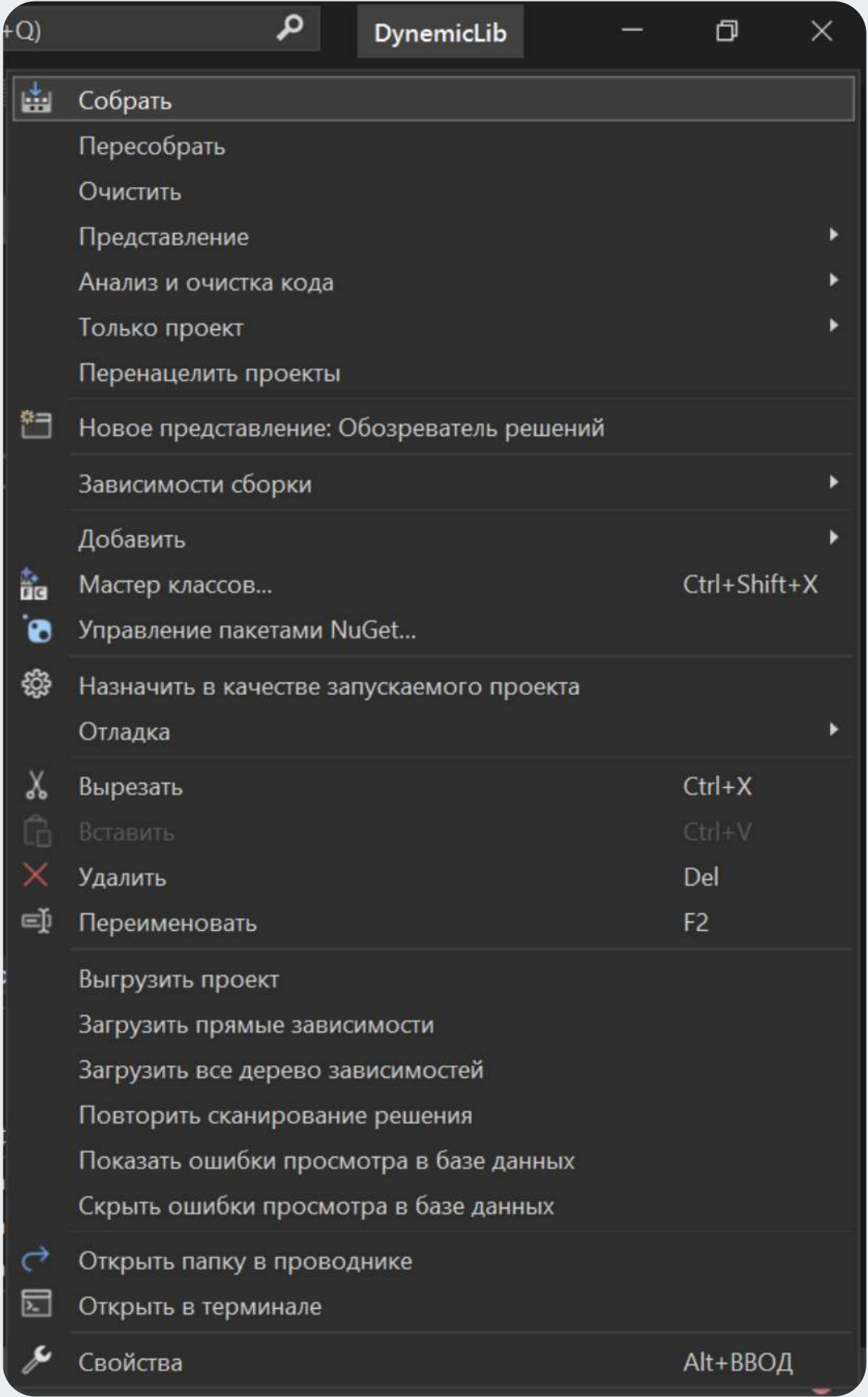
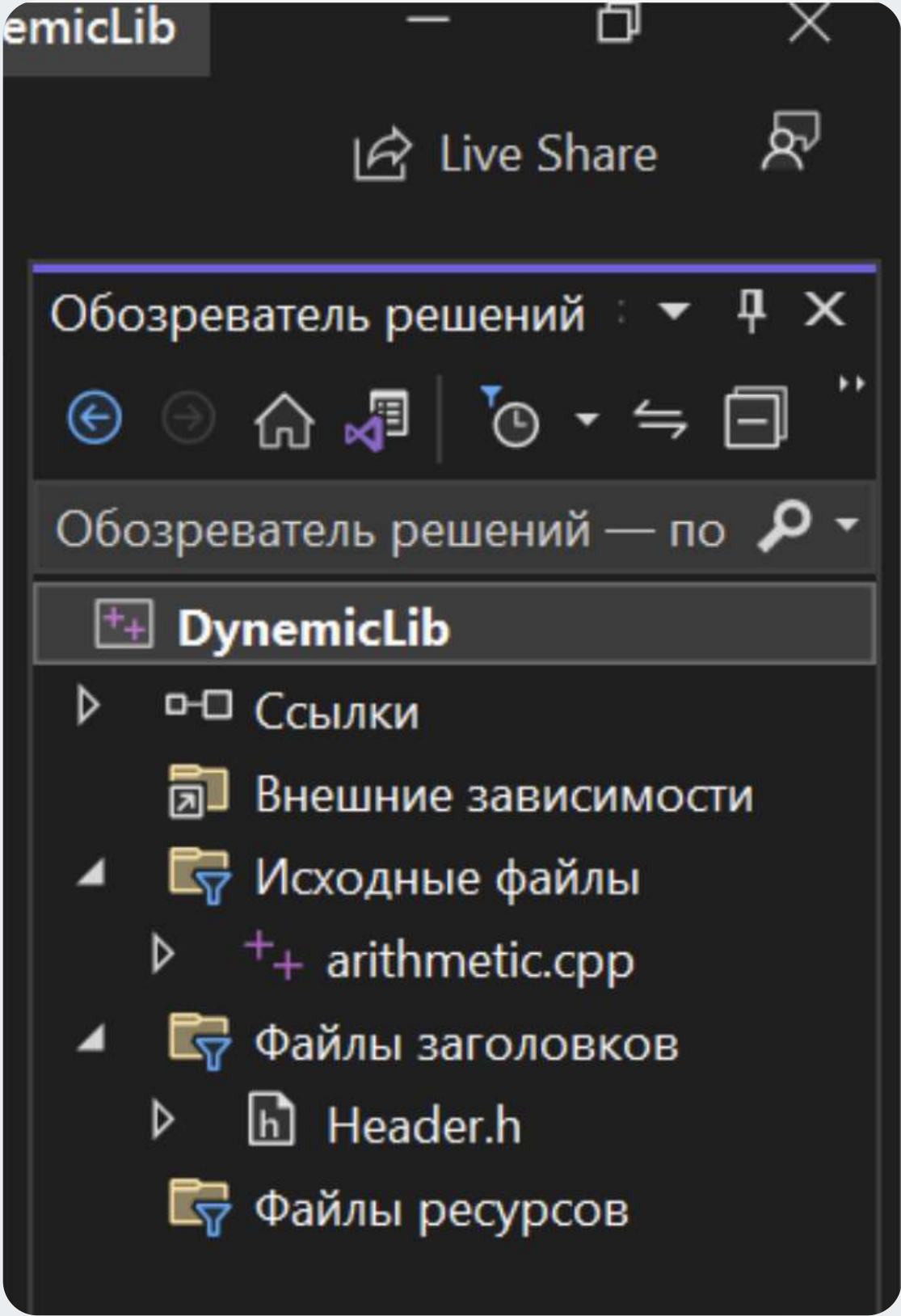
Указывает тип создаваемой программы (например, исполняемый файл, статическая библиотека, динамическая библиотека...)

OK

Отмена

Применить

Производим сборку



Сборка прошла успешно

Вывод

Показать выходные данные из: Сборка

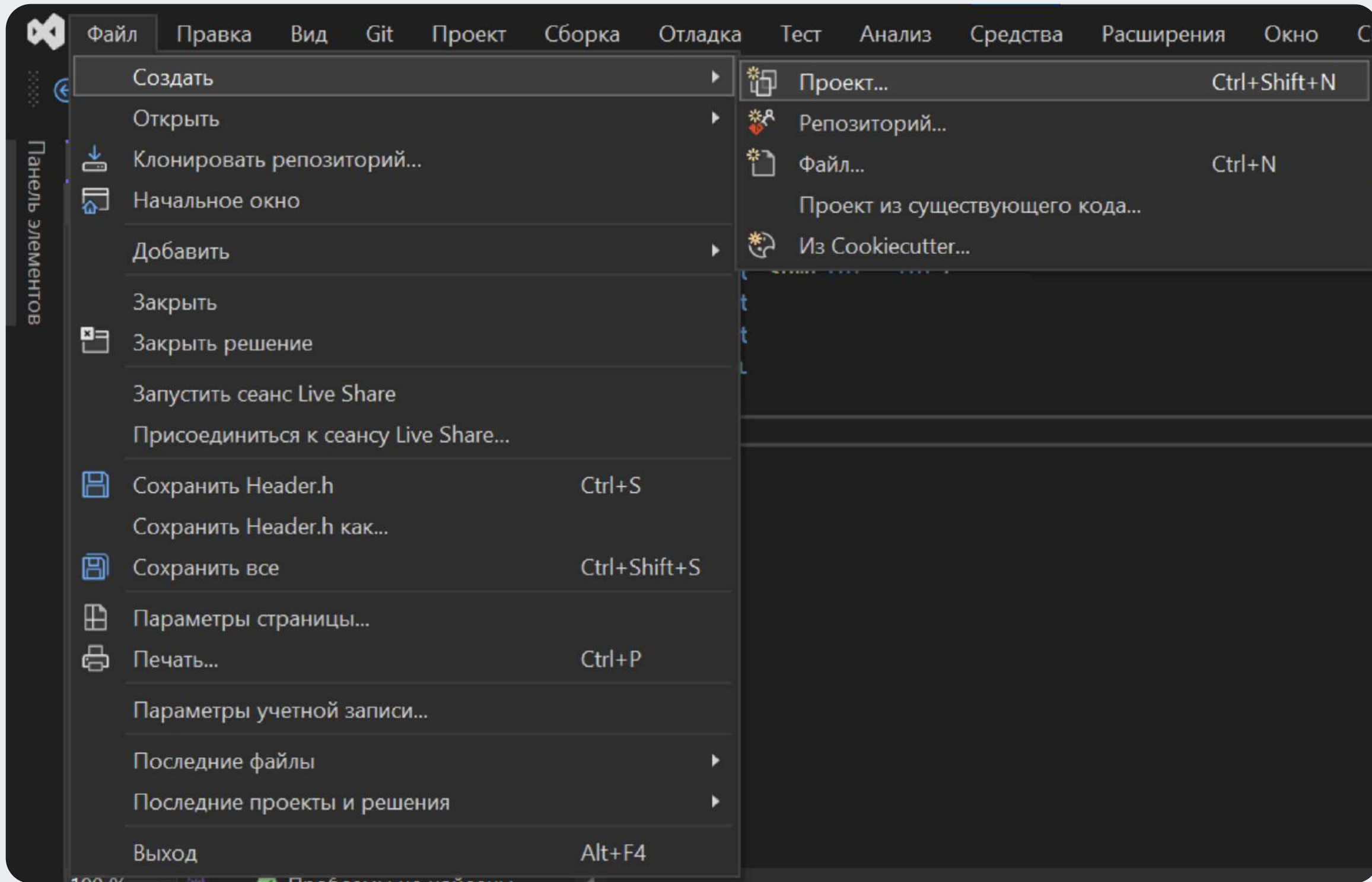
```
1> Создается библиотека C:\Practice\DynamicLib\x64\Debug\DynamicLib.lib и объект C:\Practice\DynamicLib\x64\Debug\DynamicLib.exp
1>DynamicLib.vcxproj -> C:\Practice\DynamicLib\x64\Debug\DynamicLib.dll
===== Сборка: успешно: 1, сбой: 0, в актуальном состоянии: 0, пропущено: 0=====
```

Список ошибок Вывод



Библиотека готова!


Создадим клиентские приложение,
которое будет использовать нашу dll библиотеку




Создайте новый пустой проект

Создание проекта

Последние шаблоны проектов

 Пустой проект

C++

 Консольное приложение

C++

Поиск шаблонов (ALT+"B")



Все языки

Все платформы

Все типы проектов



Пустой проект

Начать с нуля, используя C++ для Windows. Начальные файлы отсутствуют.

C++

Windows

Консоль



Консольное приложение

Выполнить код в терминале Windows. По умолчанию выводится фраза "Hello World".

C++

Windows

Консоль



Проект CMake

Создавайте современные кроссплатформенные приложения C++, не зависящие от файлов SLN или VCXPROJ.

C++

Windows

Linux

Консоль



Мастер классических приложений Windows

Создание собственного приложения Windows с помощью мастера.

C++

Windows

Рабочий стол

Консоль

Библиотека



Классическое приложение Windows

Проект приложения с графическим интерфейсом в Windows.

C++

Windows

Рабочий стол

Далее

Назовём проект «Test»

Настроить новый проект

Пустой проект

C++

Windows

Консоль

Имя проекта

Test

Расположение

C:\Practice

...

Решение

Создать новое решение

Имя решения ⓘ








Test









☐ Поместить решение и проект в одном каталоге

Назад

Создать

Добавим файл в проект

	Создать элемент...	Ctrl+Shift+A
	Существующий элемент...	Shift+Alt+A
	Новый фильтр	
	Из Cookiecutter...	
<hr/>		
	Модуль...	
	Класс...	
	Ресурс...	

Добавить		
	Мастер классов...	Ctrl+Shift+X
<hr/>		
Открыть элемент как корень обозревателя		
	Новое представление: Обозреватель решений	
<hr/>		
	Вырезать	Ctrl+X
	Копировать	Ctrl+C
	Вставить	Ctrl+V
	Удалить	Del
	Переименовать	F2
<hr/>		
	Свойства	Alt+BВОД

Обозреватель решений : ▾ 🔍 ✕

← → 🏠 📄 | ⌚ ▾ ⇌ 📁 ⌵

Обозреватель решений — по 🔍 ▾

📁 Решение "Test" (1 проекта 1)

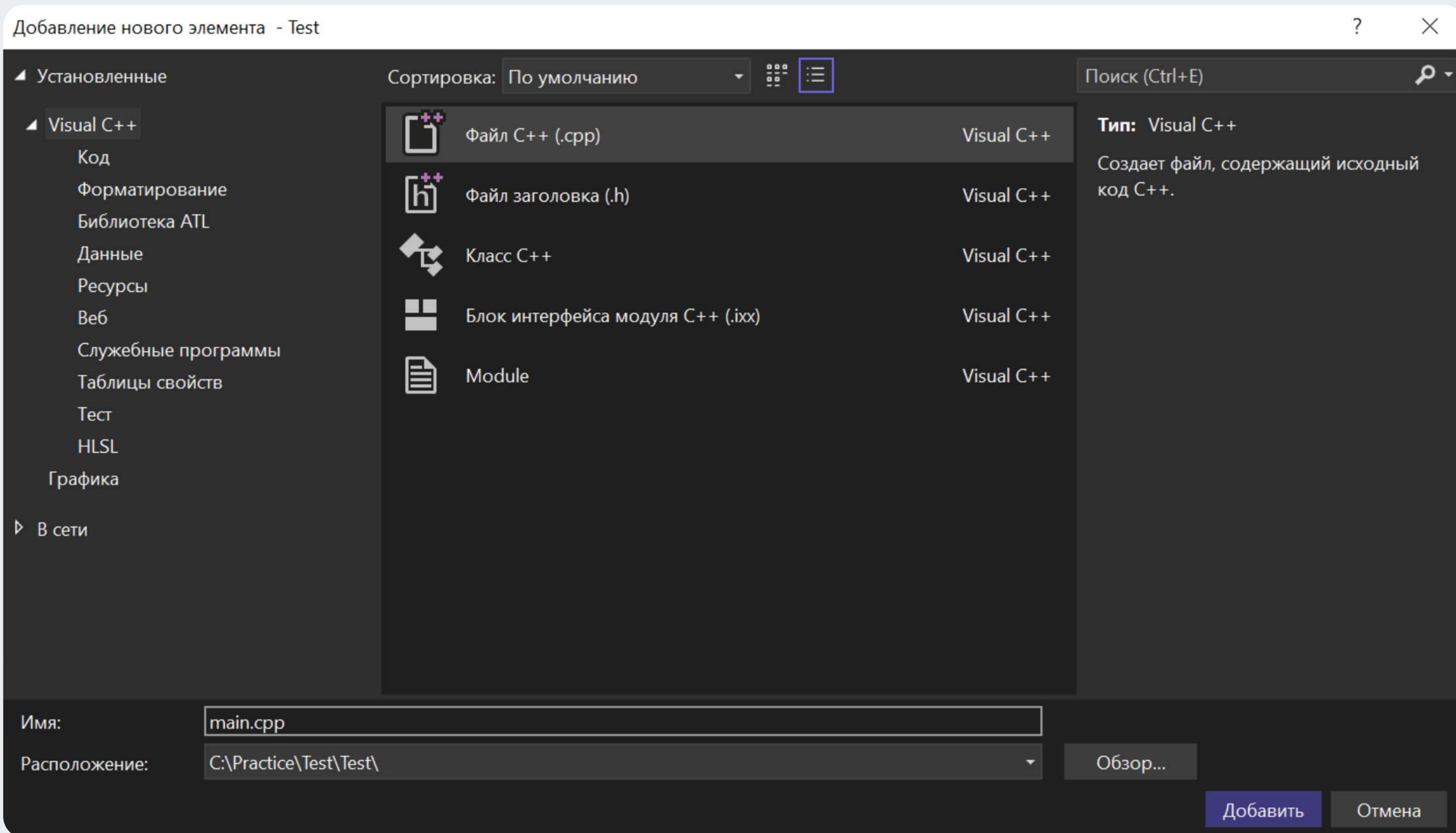
▴ 📁 **Test**

▸ 📁 Ссылки

📁 Внешние зависимости

📁 Исходные файлы

Добавим файл кода с именем main.cpp

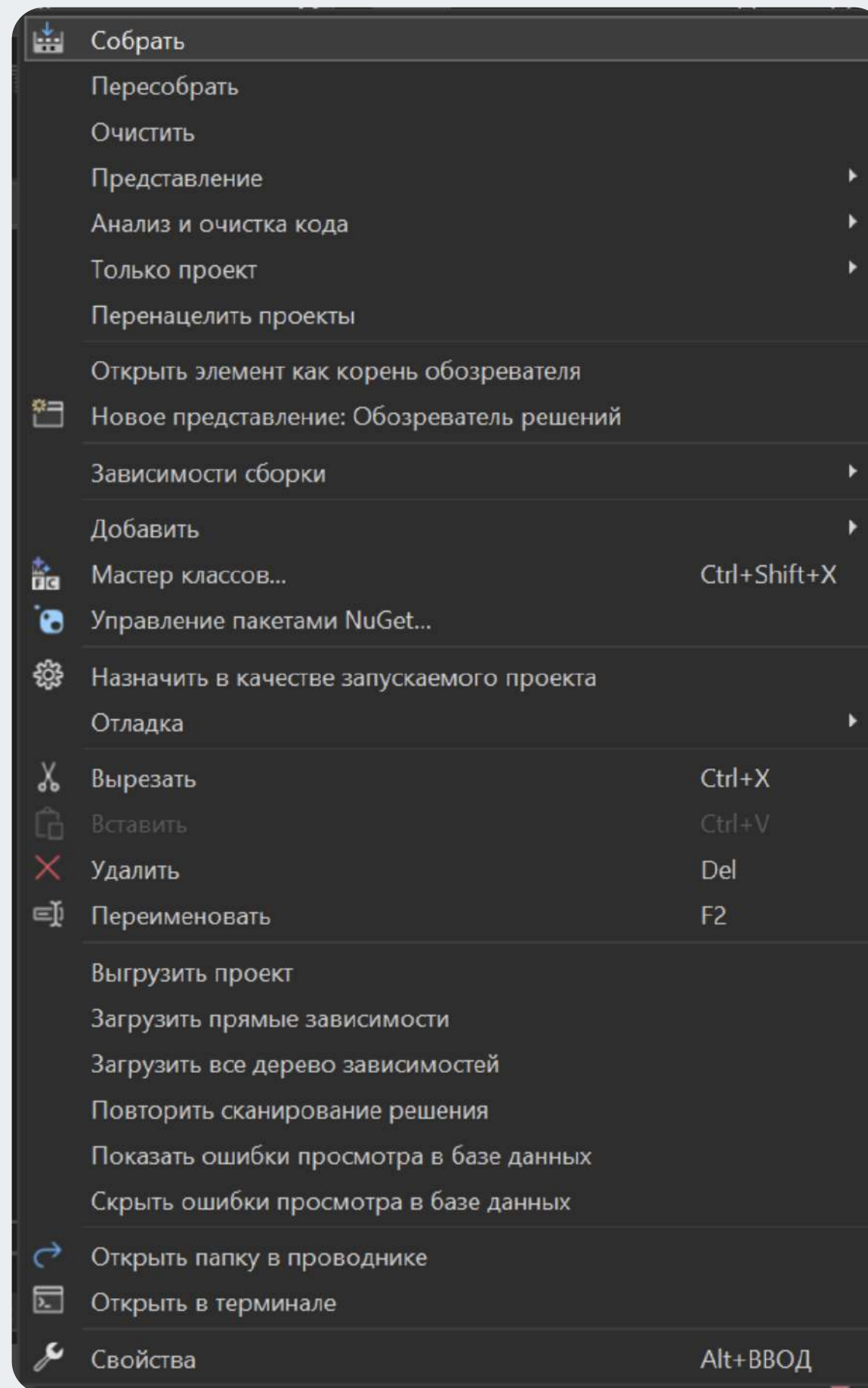
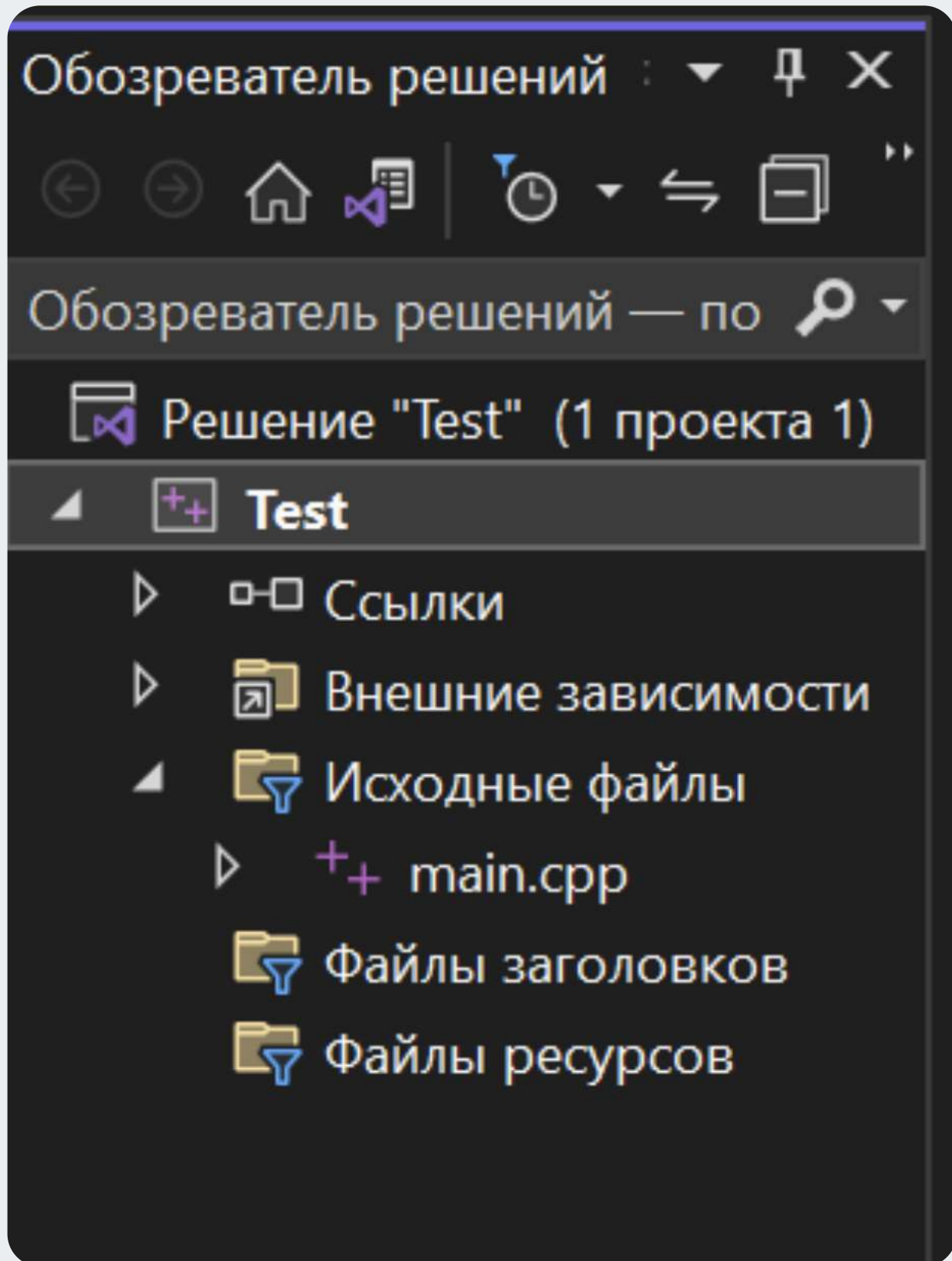


Код приложения

```
main.cpp*  Test  (Глобальная область)

1  #include <iostream>
2  #include <Windows.h>
3  int main()
4  {
5      HINSTANCE load;
6      load = LoadLibrary(L"DynemicLib.dll");
7      typedef int (*sum) (int, int);
8      typedef int (*min) (int, int);
9      typedef int (*mult) (int, int);
10     typedef double (*div) (int, int);
11     sum Sum;
12     min Min;
13     mult Mult;
14     div Div;
15     Sum = (sum)GetProcAddress(load, "Sum");
16     Min = (min)GetProcAddress(load, "Min");
17     Mult = (mult)GetProcAddress(load, "Mult");
18     Div = (div)GetProcAddress(load, "Div");
19     int a = 100;
20     int b = 200;
21     std::cout << Sum(a, b)<<"\n";
22     std::cout << Min(a, b)<<"\n";
23     std::cout << Mult(a, b)<< "\n";
24     std::cout << Div(a, b)<< "\n";
25
26     FreeLibrary(load);
27
28 }
29
```

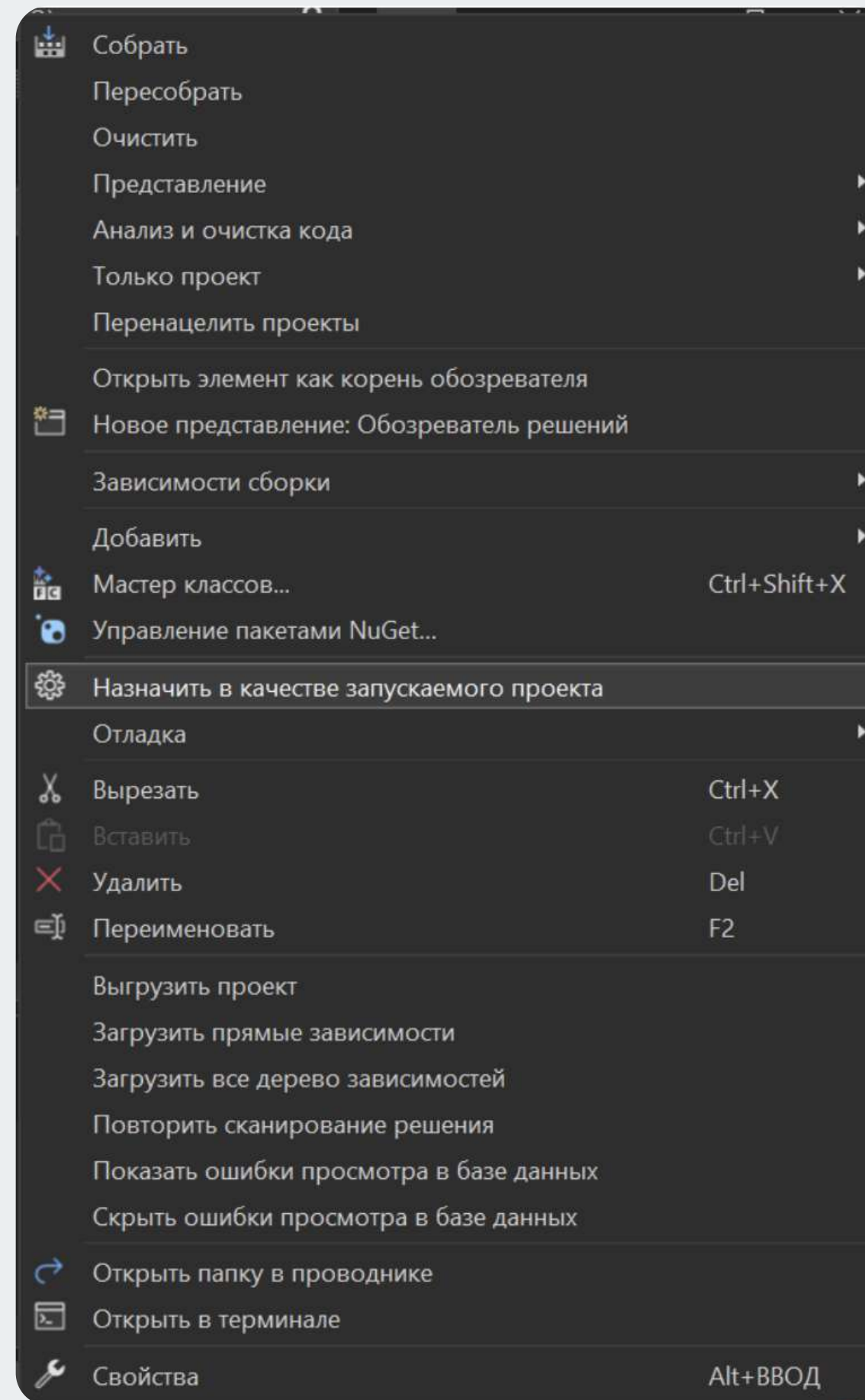
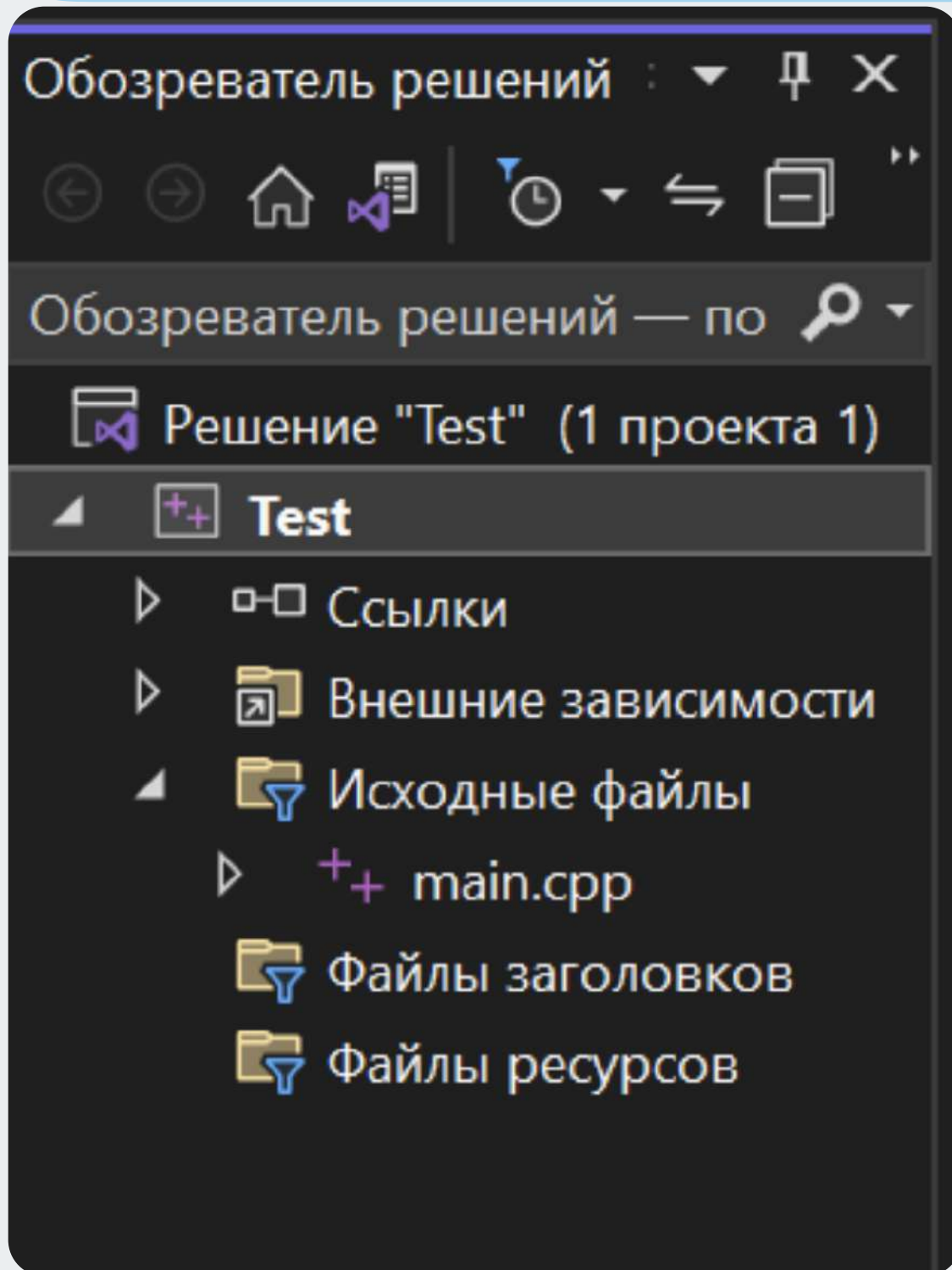

Сборка



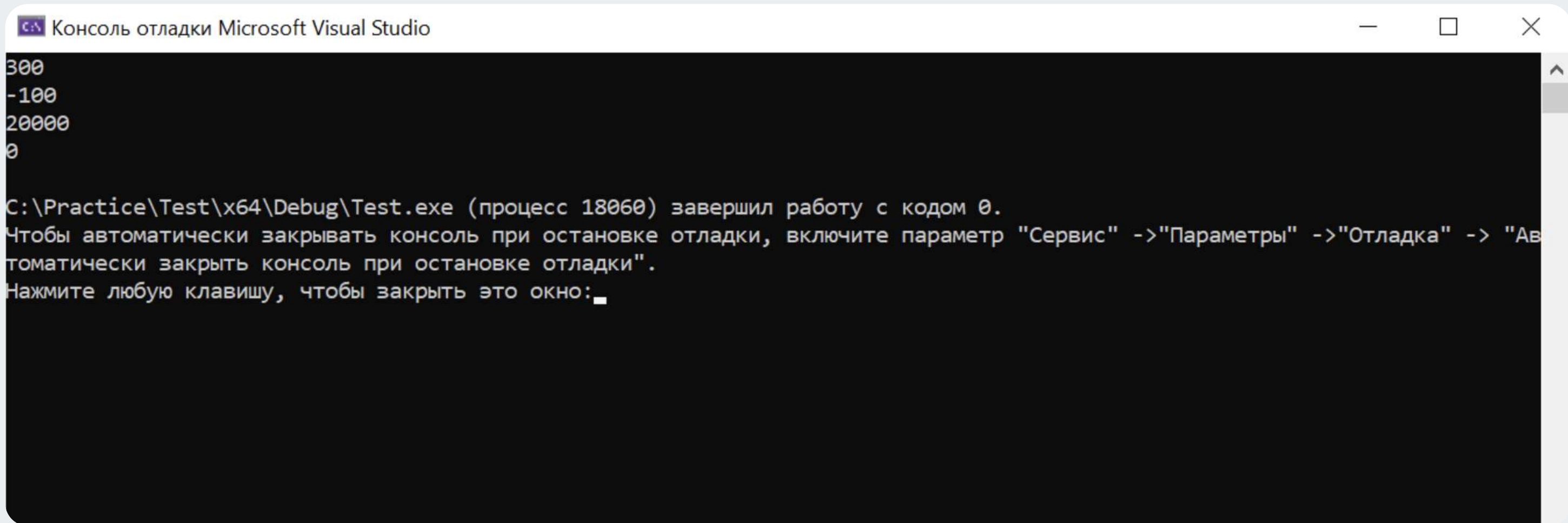
Успешная сборка

```
Вывод
Показать выходные данные из: Сборка
Повторная сборка начата...
1>----- Перестроение всех файлов начато: проект: Test, Конфигурация: Debug x64 -----
1>main.cpp
1>Test.vcxproj -> C:\Practice\Test\x64\Debug\Test.exe
===== Перестроить все: успешно – 1, неудачно – 0, пропущено – 0 =====
```

Назначим наш проект в качестве запускаемого проекта



Запуск



The image shows a screenshot of the 'Консоль отладки Microsoft Visual Studio' (Microsoft Visual Studio Debug Console) window. The window has a white title bar with the Visual Studio logo and standard minimize, maximize, and close buttons. The main area is a black console with white text. The text in the console shows the end of a program execution with return code 0, followed by instructions on how to automatically close the console using the 'Service' parameter in the 'Debug' menu.

```
300  
-100  
20000  
0  
  
C:\Practice\Test\x64\Debug\Test.exe (процесс 18060) завершил работу с кодом 0.  
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окно: _
```