Vorlesung

Objektorientiertes Programmieren in C++

Teil 1 - WS 2024/25

Detlef Wilkening
www.wilkening-online.de

1 Allgemeines

Thema

- Objektorientiertes Programmieren in C++
- Achtung – sowohl die Sprache als auch die Bibliotheken sind so umfangreich, dass die Vorlesung aus Zeitmangel bei weitem nicht alle Elemente abdecken kann, und bei den abgedeckten auch viele Details auslässt.

Voraussetzungen

- Kenntnisse einer beliebigen Programmiersprache.
- Die Vorlesung setzt voraus, dass Sie die Grundlagen der Programmierung kennen. Sie sollten z.B. wissen was eine Variable, ein Typ oder eine Funktion ist, und warum und wofür man sie benutzt. Sie sollten die Begriffe Scope (Block), Lebensdauer, Speicher, Stack und Heap zumindest ungefähr zuordnen können. Über den Vorteil einer konsistenten und durchgängigen vorstellen Formatierung und Benamung sollte ich kein Wort mehr verlieren müssen. Ihnen sollte klar sein, warum die Auftrennung in möglichst unabhängige Module sehr sinnvoll ist. Und zu guter Letzt sollten Sie zumindest in Ansätzen in der Lage sein, ein Problem in Teil-Probleme zu zerlegen, es zu strukturieren, und einfache Algorithmische Lösungen zu verstehen und erarbeiten zu können.

Praktikum

- Kleine und große Aufgaben in C++ lösen

Tools

- Sie benötigen für die Vorlesungen, Übungen und Praktika zumindest einen guten Editor und einen modernen C++ Compiler auf Ihrem Rechner. Sie dürfen auch gerne mit einer Integrierten Entwicklungs-Umgebung (IDE) arbeiten.
- Da wir – bis auf eine Ausnahme – nur ISO C++ machen, ist sowohl das Betriebssystem als auch die Umgebung, mit der Sie arbeiten, für die Vorlesung egal. Sie können gerne

sowohl unter Windows, aber auch unter Linux, Mac OS-X, oder einem anderes System arbeiten. Auch die benutzten Tools stelle ich Ihnen frei.

Ich werde in der Vorlesung benutzen:

- Microsoft Visual Studio 2022
  - https://visualstudio.microsoft.com/de/vs/community/
  - Freie integrierte Entwicklungsumgebung für Windows mit Editor, C++ Compiler und mehr
  - Achtung – die Screenshots beziehen sich noch auf das alte Microsoft Visual Studio 2019. Es hat sich aber an den hier wichtigen Dingen nichts Grundlegendes geändert.

Alternativen

- Compiler: GCC
  - https://de.wikipedia.org/wiki/GNU_Compiler_Collection
  - https://de.wikipedia.org/wiki/MinGW
- Compiler: CLANG/LLVM
  - https://de.wikipedia.org/wiki/Clang
  - https://de.wikipedia.org/wiki/LLVM
- IDEs
  - https://de.wikipedia.org/wiki/Liste_von_Integrierten_Entwicklungsumgebungen
- Editoren:
  - https://code.visualstudio.com/
  - http://notepad-plus-plus.org/
  - https://de.wikipedia.org/wiki/Vim
  - https://de.wikipedia.org/wiki/Emacs

Literatur und WWW

Es gibt hunderte von Büchern zu C++, Zeitschriften, und zig-Millionen Artikel im Web. Die folgenden Anlaufstellen im Internet sind eine gute erste Wahl für Informationen:

- http://en.cppreference.com/w/
- http://www.cplusplus.com/
- http://isocpp.org/
- https://stackoverflow.com/questions/tagged/c%2b%2b

2 C++

Historie

- Entwickelt von Bjarne Stroustrup
- Start der Entwicklung Mai 1979
- Entstanden aus C
- C++ basiert auf ISO C

ISO Standards

- C++98 von 1998

- C++03 von 2003
- C++11 von 2011
- C++14 von 2014
- C++17 von 2017
- C++20 von 2020
  Genau genommen immer noch (Stand 25.9.2024) der aktuelle Standard
  https://www.iso.org/standard/79358.html
- C++23 von eigentlich 2023, er ist aber immer noch nicht endgültig erschienen
  Aktueller Status (25.9.2024): „Stage: 60.00 (Under publication)“
  https://www.iso.org/standard/83626.html
- Der nächste Standard C++26 wird im Herbst 2026 erscheinen
  Der Final Draft wird im Herbst 2025 erwartet

Die Vorlesung wird prinzipiell C++23 machen und sogar schon auf C++26 schauen, aber dann im Detail nur sehr wenig von C++20, 23 & 26 nutzen.

3 Einführung

3.1 Das erste C++ Programm

Das erste C++ Programm:

```
// Das erste C++ Programm

#include <iostream>

int main()
{
   std::cout << "Hallo Welt";
}
```

Allgemein

- C++ Quelltext ist formlos
  bis auf drei Ausnahmen:
  - Zeilen-Kommentare
  - Präprozessor Direktive
  - Zeichenketten-Konstanten

Kommentare

- Von // bis zum Zeilenende ist Kommentar und wird vom Compiler ignoriert.
  Dies ist eine der drei Ausnahmen bzgl. formlosen Quelltexts.

#include <iostream>

- Präprozessor Direktive
- allgemein:
  #include <xyz> bzw. #include “xyz“
  macht weitere Deklarationen und Definitionen bekannt

bindet den Header xyz ein

- Hier:
  macht einen Teil der I/O C++ Standard-Bibliothek bekannt, die damit benutzt werden kann
  bindet den Header „iostream“ ein
- Auch Präprozessor Direktiven sind nicht formlos:
  sie müssen abgesehen von Kommentaren allein in der Zeile stehen

int main() { ... }

- Definiert eine Funktion – hier „main“
- Eine Funktion ist ein Programmteil, das einen Namen hat, und von überall her benutzt werden kann.
- Für uns hier erstmal wichtig ist, dass es in C++ zwei spezielle Funktionen gibt, die den Startpunkt des Programms bildet. D.h. genau eine von ihnen muß in einem Programm vorkommen.
  Funktion 1: int main()
  Funktion 2: int main(int argc, char* argv[ ])
  Weiteren Varianten von „main“, die ihr Compiler möglicherweise unterstützt, sind nicht Teil von ISO C++. Wie beschränken uns hier auf Variante 1.
- Alles weitere zu Funktionen später. Nur noch ein Hinweis für alle, die schon etwas C oder C++ Kenntnisse haben: Ja, es ist richtig und ernstgemeint, dass die Funktion „main“ keinen „int-Wert“ zurückgibt, obwohl sie mit dem Rückgabe-Typ „int“ definiert ist. Für die beiden ISO „main“ Funktionen definiert der Standard die Ausnahme, dass das return optional ist, und wenn nicht vorhanden einem „return 0;“ entspricht.

{ }

- Zwei geschweifte Klammern bilden einen Block, der eine Menge von Elementen (auch eine leere Menge) zu einer Einheit zusammenfasst.
- In diesem Fall bildet der Block die Implementierung der Funktionen, d.h. eine Menge von Deklarationen, Definitionen, und vor allem Anweisungen.
- Innerhalb einer Funktion kann ein Block auch stellvertretend für eine Anweisung stehen. Dies wird z.B. bei den Kontrollstrukturen und Schleifen häufig benötigt und genutzt.

std::cout

- „std“ ist der Namensraum für (fast) alle Elemente der C++ Standard-Bibliothek
- :: ist der Scope-Resolution-Operator (Bereichs-Zuordnungs-Operator), der dem folgenden Namen (hier „cout“) einem Bereich zuordnet (hier dem Namensraum „std“).
- cout steht für char-output und ist ein Stream-Objekt dass mit der Console für die Ausgabe verbunden ist. D.h. alle Ausgabe-Elemente die in std::cout hineingeschoben werden, erzeugen eine Ausgabe auf der Console.
- Der Typ von std::cout ist std::ostream.
- Für dieses Objekt (und die auf es mögichen Funktionen) mußte der Header iostream eingebunden werden, da ein C++ Compiler nur Dinge compiliert, die ihm *bekannt* sind.
- Ein Stream ist ein Ein- oder Ausgabestrom von Zeichen, der mit einem Gerät, einer Datei, o. ä. verbunden ist.

<<
- Ausgabe-Operator (in C heißt er Links-Schiebe-Operator)
- Mit ihm wird das rechts-stehende Objekt (2ter Operand) in den Stream (1ter Operand) hineingeschoben.
- Viele Objekte (aber bei weitem nicht alle) lassen sich mit dem Ausgabe-Operator in einen Stream schieben und damit ausgeben. Zumindest für alle Literale und alle elementaren Datentypen, sowie Strings trifft dies zu.

“Hallo Welt“
- Zeichenkettenkonstanten stehen in doppelten Anführungszeichen.
- Zeichenkettenkonstanten dürfen die von C bekannten Backslash-Sequenzen enthalten wie z.B.: „\n“ für einen Zeilenumbruch, „\t“ für einen Tabulator, „\““, „\\“, usw.

std::cout << “Hallo Welt“;
- Bildet eine Anweisung, d.h. etwas was der Computer ausführen soll.
  In diesem Fall wird die Zeichenkettenkonstante „Hallo Welt“ in den Ausgabestrom std::cout geschoben, d.h. auf der Console ausgegeben.
- Eine Anweisung wird immer durch ein Semikolon „;“ abgeschlossen.
- Anweisungen dürfen nur in Funktionen stehen.
- Anweisungen werden nacheinander – in der Reihenfolge ihres Auftauchens im Quelltext – abgearbeitet. Diese sequentielle Abarbeitung kann durch Schleifen, Kontrollstrukturen, (implizite und explizite) Funktionsaufrufe und Exceptions beeinflußt werden. Die Abarbeitung beginnt beim Start des Programms in der Funktion main. Hinweis – dieses Verfahren nennt man imperativ, weshalb C++ zu den sogenannten imperativen Sprachen gehört.
- Es darf auch leere Anweisungen geben.

3.2 Vereinfachung

Das ständige Schreiben von „std::“ vor den Symbolen der C++ Standard-Bibliothek ist auf Dauer recht aufwändig. Namensräume machen zwar sehr viel Sinn – hier beim Lernen erzeugen sie aber nur Schreibarbeit. Natürlich gibt es in C++ eine Lösung dafür.

Mit einer Using-Anweisung werden alle Symbole eines Namenraums in den aktuellen *importiert* – und können damit ohne Namensraum-Qualifizierung angesprochen werden.

```
#include <iostream>

using namespace std;      // Using-Anweisung

int main()
{
   cout << "Besser";      // Benutzung ohne std::
}
```

Eine alternative Lösung ist die Nutzung der Using-Deklaration „using std::cout“, die nur das Symbol „cout“ aus „std“ direkt ansprechbar macht. Bei mehreren Symbolen können mehrere

Using-Deklarationen genutzt werden.

```
#include <iostream>

using std::cout;          // Using-Deklaration

int main()
{
   cout << "Oder so";     // Benutzung ohne std::
}
```

Der Haupt-Unterschied zwischen einer Using-Anweisung und –Deklaration ist, dass erstere alle Symbole *importiert*, während zweitere nur das eine Symbol *importiert*. In der Praxis ist die Using-Anweisung einfacher in der Nutzung (nur eine Zeile für alle Symbole) – kann aber zu Namenskonflikten führen (ein *importiertes* Symbol gibt es auch woanders und wird auch von dort importiert). Dann muss man entweder Using-Deklarationen verwenden oder vollständig referenzieren.

Achtung – in Produktiv Code sollten Sie niemals eine Using-Anweisung verwenden. Da sie alle Symbole importiert, sind Konflikte und Probleme vorprogrammiert.

4 Microsoft Visual Studio

Sie benötigen mindestens einen Workspace (Solution, Projektmappe) und ein Projekt.

4.1 Solution und Projekt

Nach dem Startbildschirm öffnet sich der Projekt-Auswahl Dialog.

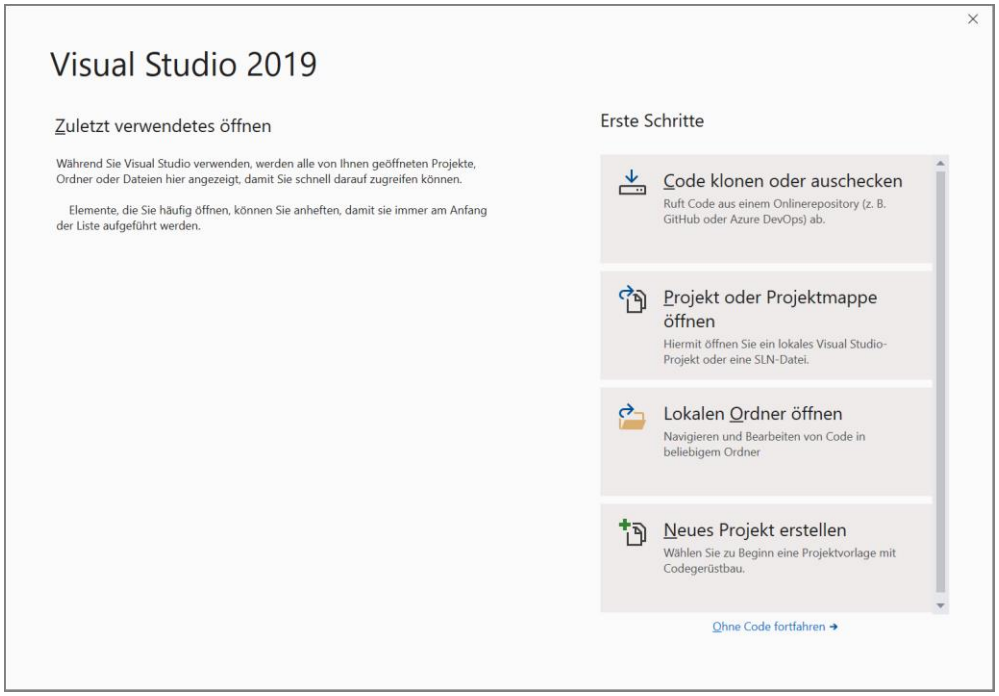

Abb. 4-1 : Projekt-Auswahl Dialog

Wir wollen ein neues Projekt beziehungsweise zuerst eine neue Projektmappe erstellen – darum wählen Sie „Neues Projekt erstellen“ aus. Es öffnet sich der Dialog zum Erstellen neuer Projekte.

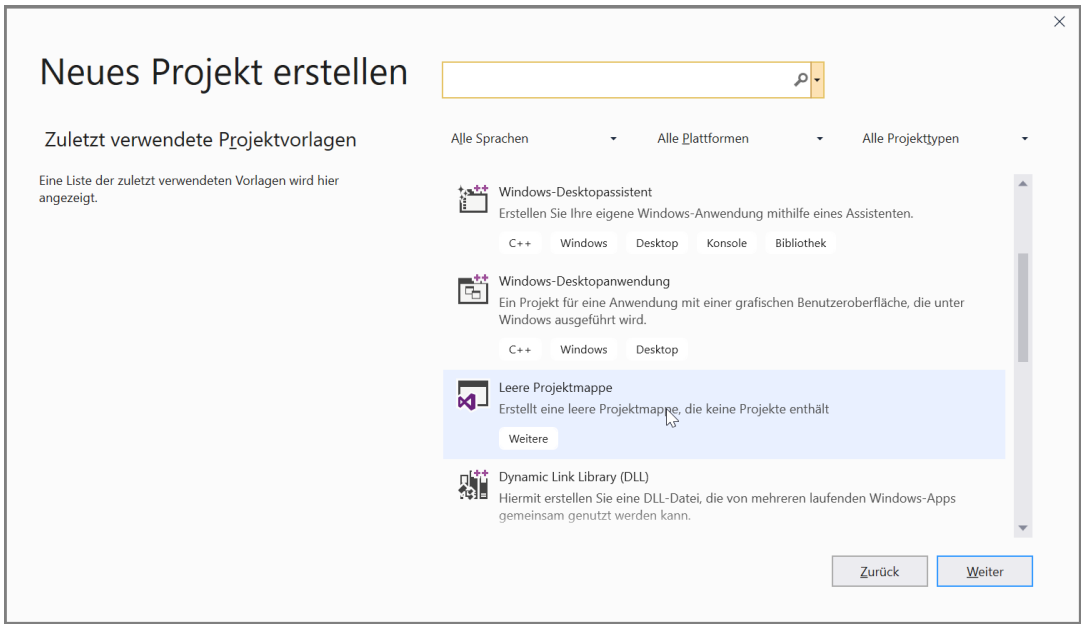

Abb. 4-2 : Projekt-Erstellungs-Dialog

Wählen Sie also im Projekt-Erstellungs-Dialog „Leere Projektmappe“ aus.

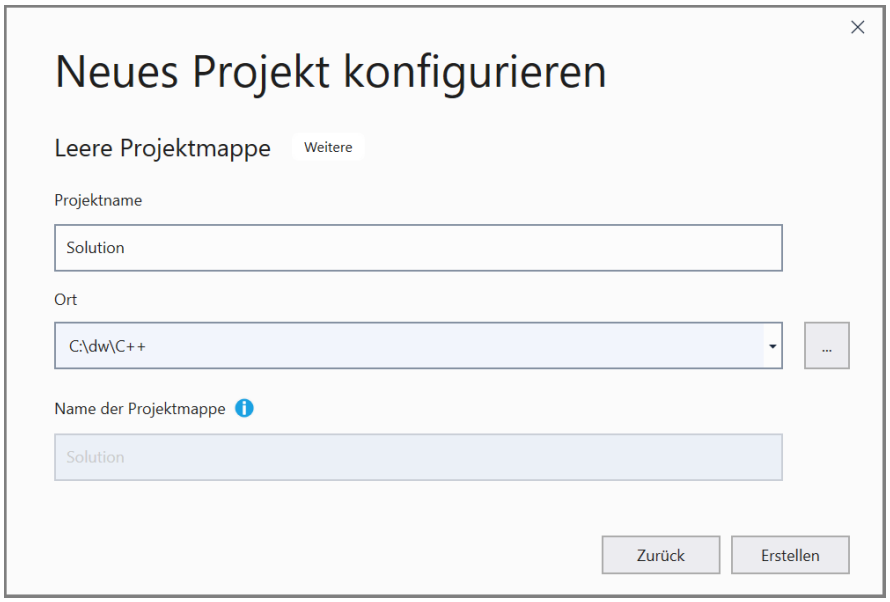

Abb. 4-3 : Dialog zum Anlegen einer leeren Projektmappe

Mit dem Button „Erstellen“ erstellen Sie dann die leere Projektmappe.

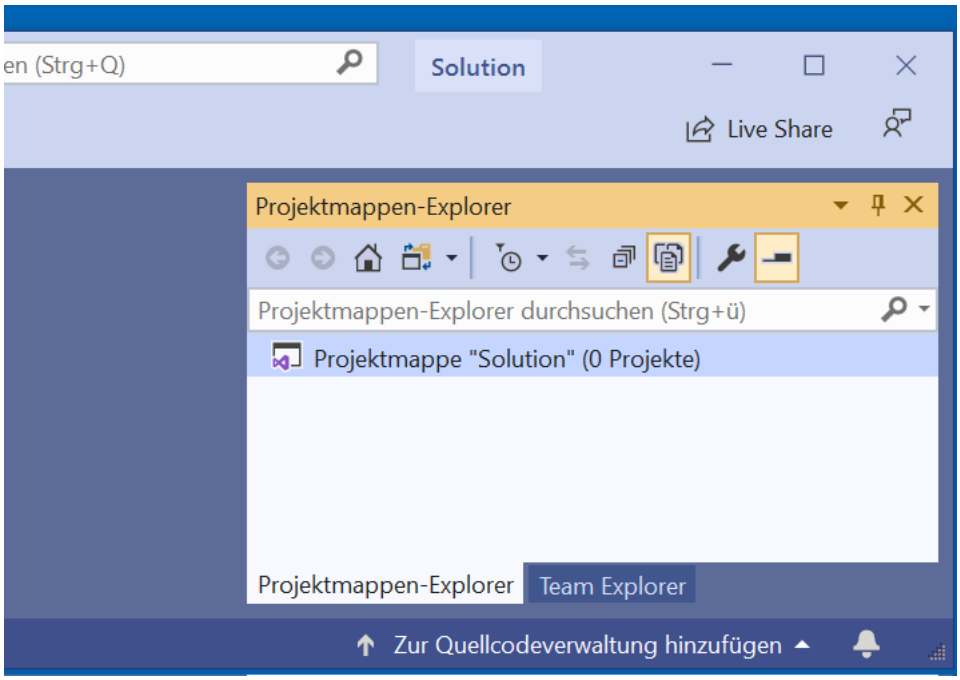

Abb. 4-4 : Leere Projektemappe im Microsoft Visual Studio

Erzeugung eines Projekts innerhalb der Projektmappe. Dazu selektieren Sie die Projektmappe und öffnen mit der rechten Maustaste das Kontextmenü.

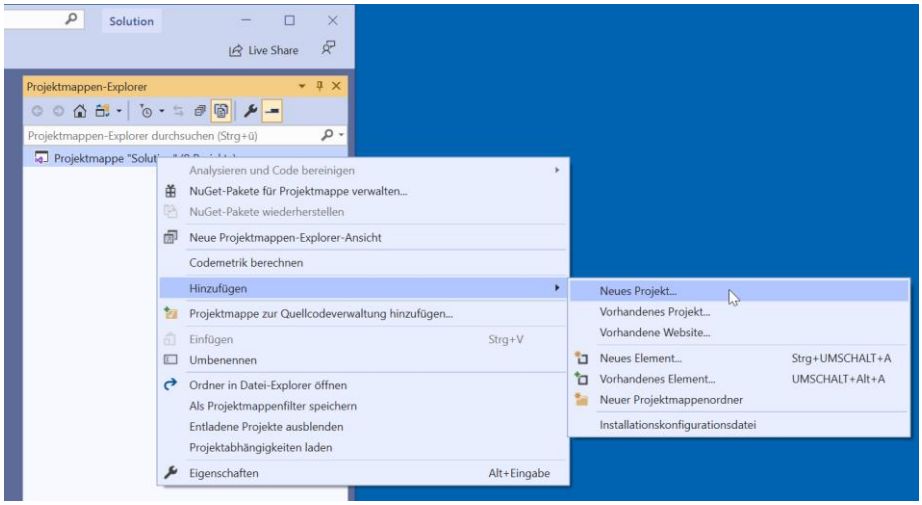

Abb. 4-5 : Neues Projekt in der Projektmappe anlegen

Im Untermenü „Hinzufügen“ wählen Sie den Punkt „Neues Projekt…“ aus.

Auch wenn es eine fertige Projekt-Vorlage für ein Konsolen-Projekt gibt, wählen Sie hier bitte die Vorlage „Leere Projekt“ aus.

Abb. 4-6 : Hinzufügen eines neuen leeren Projekts

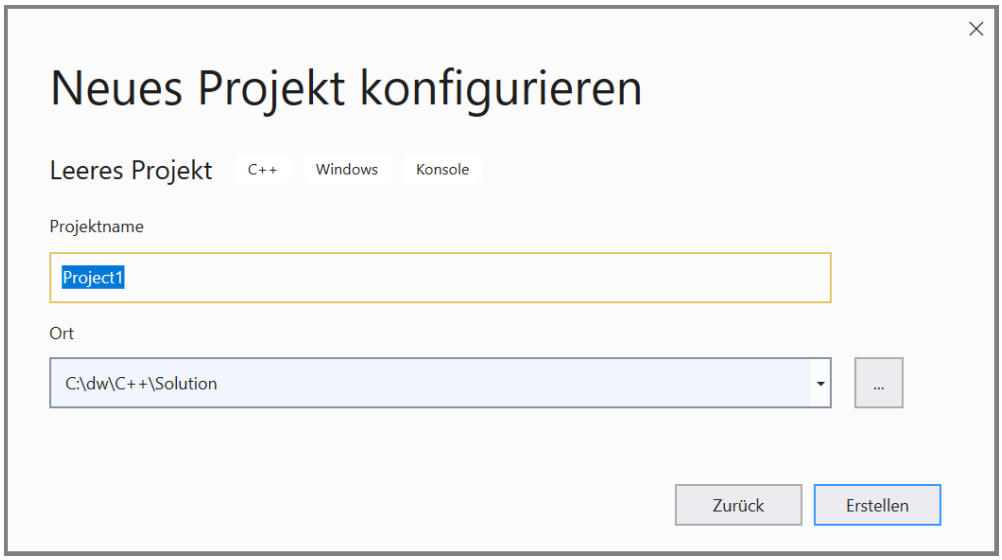

Abb. 4-7 : Dialog zum Anlegen eines leeren Projekts

Mit „Erstellen“ wird das Projekt erstellt. Im Projektmappen-Explorer ist jetzt das Projekt in der Projektmappe vorhanden.

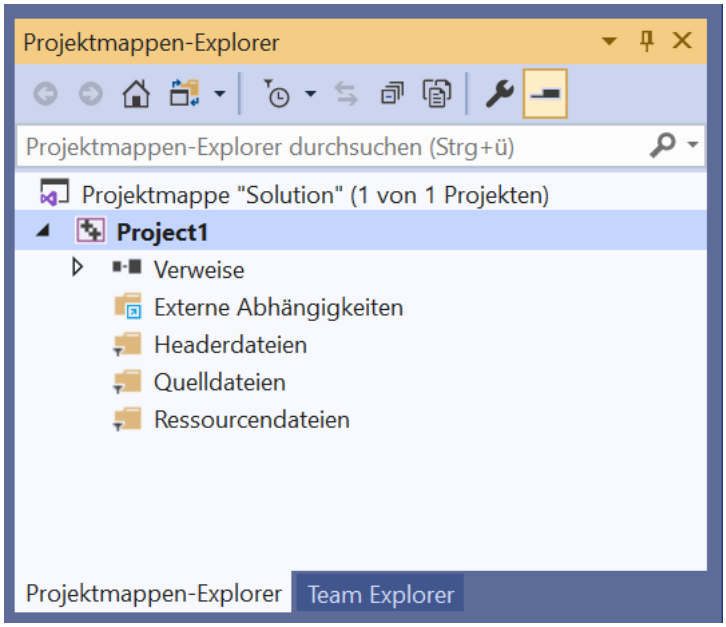

Abb. 4-8 : Projektmappe mit unserem ersten Projekt

Das Projekt mit Leben zu füllen. Dazu müssen wir ein neues Element (eine C++ Datei mit Endung „.cpp“) in das Projekt einfügen. Öffnen Sie das Kontext-Menü, und wählen Sie hier den Eintrag „Neues Element…“ im Untermenü „Hinzufügen“ aus.

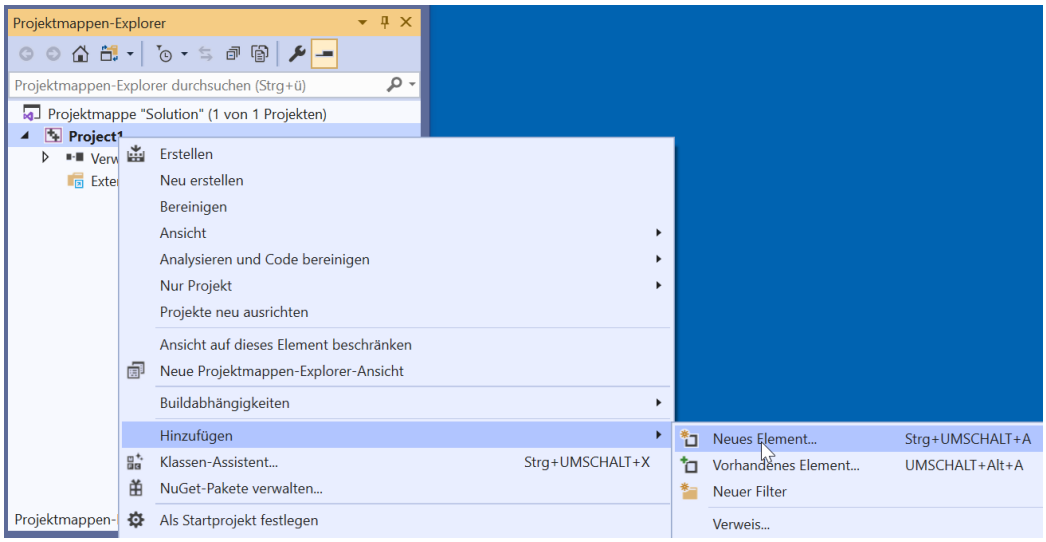

Abb. 4-9 : Neues Element dem Projekt hinzufügen

Es erscheint dann der Dialog zum Hinzufügen von neuen Elementen.

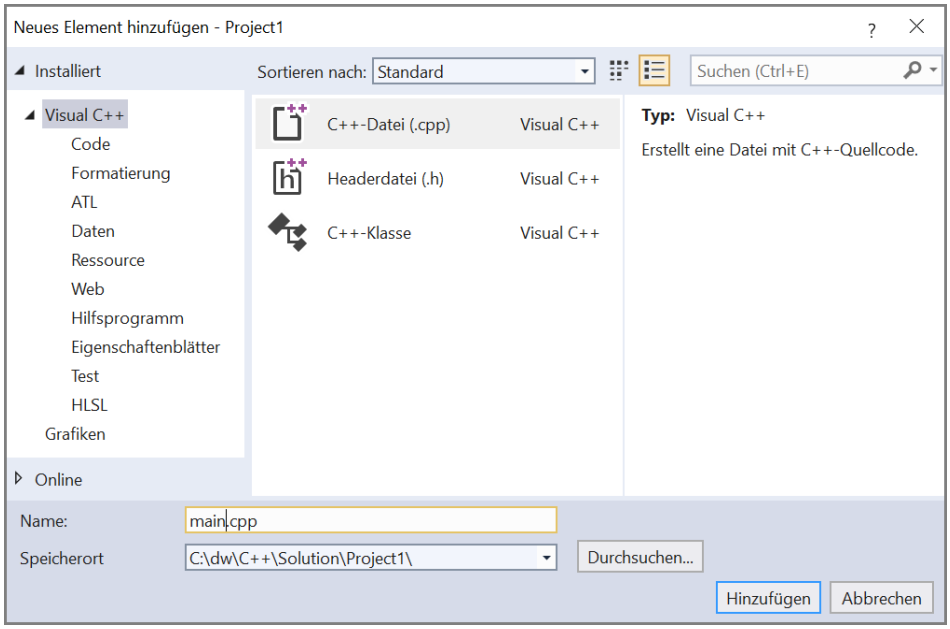

Abb. 4-10 : Dialog zum Hinzufügen von neuen Elementen

Wählen Sie „C++ Datei“ aus, und geben Sie im Textfeld unten den Namen für die C++ Datei an.

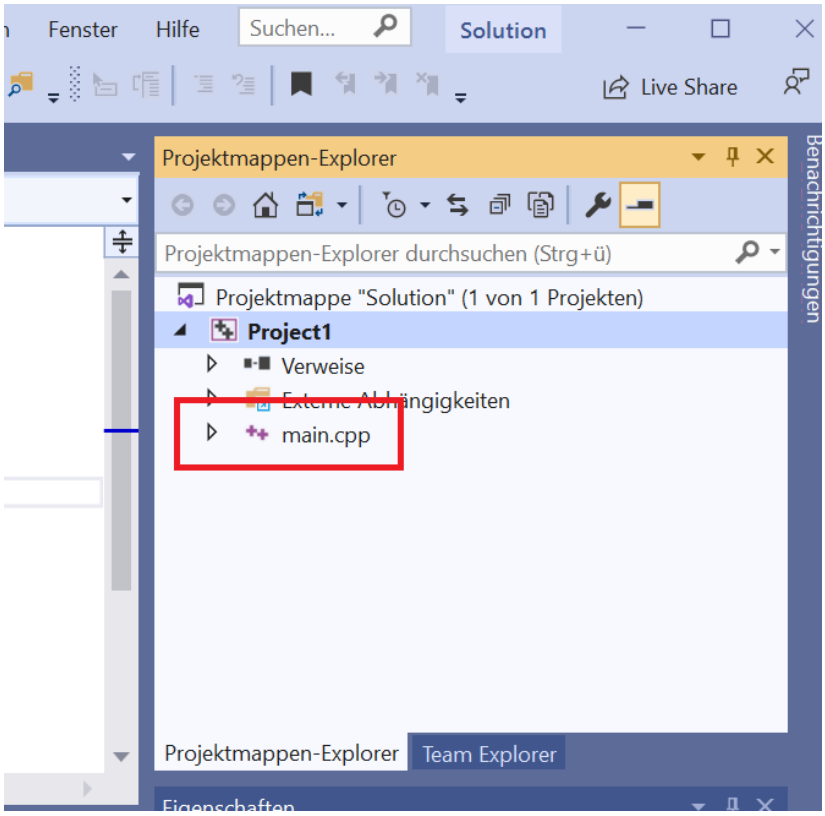

Abb. 4-11 : Projekt mit C++ Datei „main.cpp“

Geben Sie nun unser erstes C++ Programm im Microsoft Visual Studio in Ihre C++ Datei ein.

```
// Das erste C++ Programm

#include <iostream>

int main()
{
   std::cout << "Hallo Welt";
}
```

Nach der Eingabe müssen Sie das Projekt bauen. Zum Beispiel im Menü „Erstellen“ können Sie mit „Project1 erstellen“ Ihr Projekt compilieren.

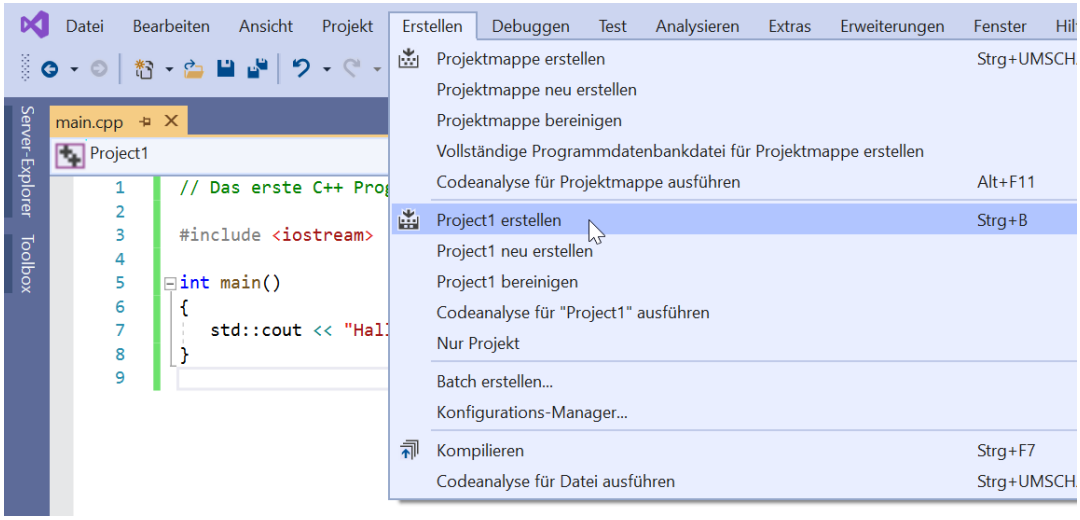

Abb. 4-12 : Projekt erstellen

Der Compilier-Vorgang dauert ein paar Sekunden. Während des Bauens öffnet das Microsoft Visual Studio einen neuen Ausgabe-View. In diesem sehen Sie unter anderem, was gerade compiliert wird.

Abb. 4-13 : Compiler-Ausgabe im Ausgabe View

Nun können wir unser Programm endlich das erste Mal starten. Hierfür finden Sie im Menü „Debuggen“ zum Beispiel den Eintrag „Starten ohne Debugging“.

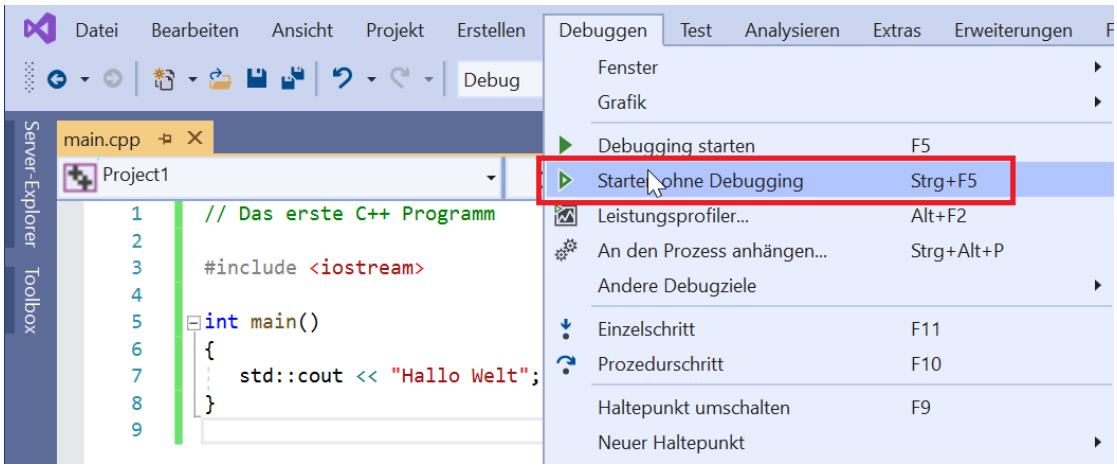

Abb. 4-14 : Starten unseres Programms

Da wir eine Konsolen-Anwendung geschrieben hat, öffnet sich eine Konsole und wir sehen dort die Ausgabe von „Hallo Welt“.

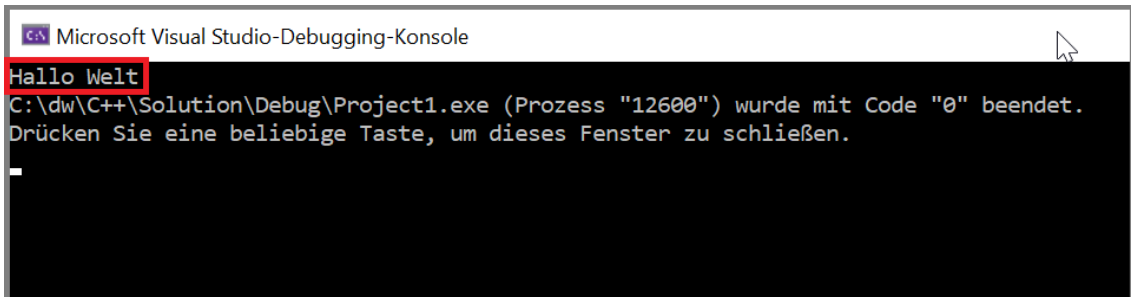

Abb. 4-15 : Unser erstes Programm läuft in der Visual Studio Konsole

Herzlichen Glückwunsch zu Ihrem ersten C++ Programm!

4.2 Das zweite C++ Projekt

Legen Sie ein zweites Projekt „Project2“ in Ihrer Projektmappe an.

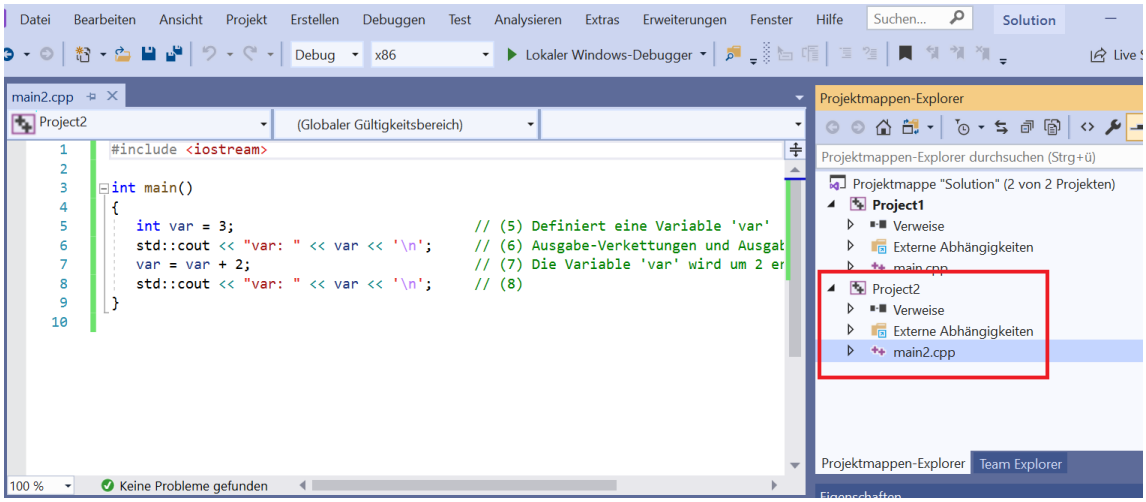

Abb. 4-16 : Das zweite Projekt in der Projektmappe

Wenn Sie jetzt das Programm starten wollen, dann wird leider wieder das „Hallo Welt“ Programm gestartet. Das Problem ist, dass die Projektmappe viele Projekte enthalten kann – und es muss klar sein, welches Projekt gestartet werden soll. Dazu definiert die Projektmappe ein sogenanntes Start-Projekt. Sie können das Start-Projekt umdefinieren, indem Sie auf dem Projekt das Kontext-Menü öffnen, und dort dieses Projekte „Als Startprojekt festlegen“.

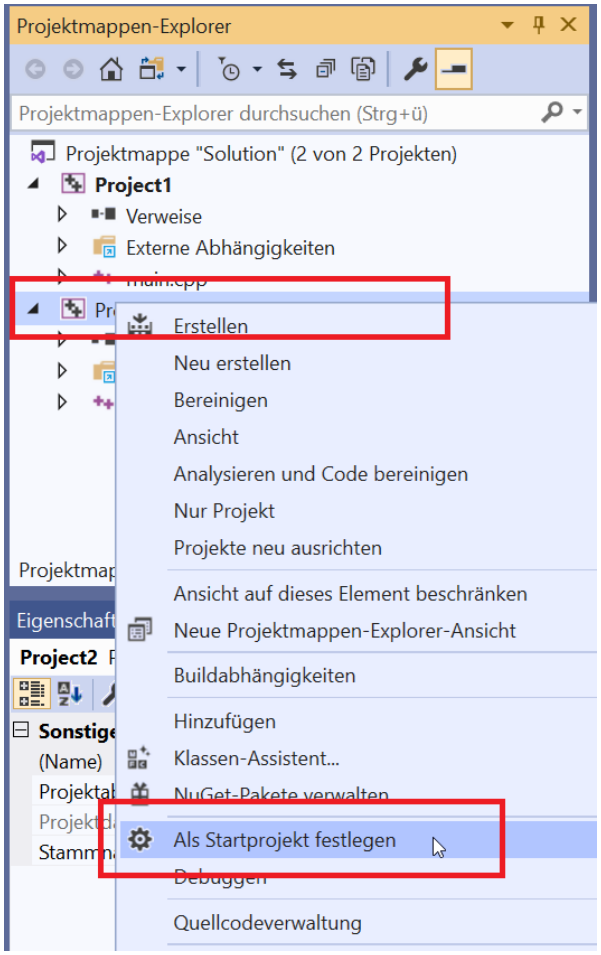

Abb. 4-17 : Startprojekt definieren

Sie erkennen das aktuelle Startprojekt auch daran, dass es im Projektmappen-Explorer fett dargestellt wird.

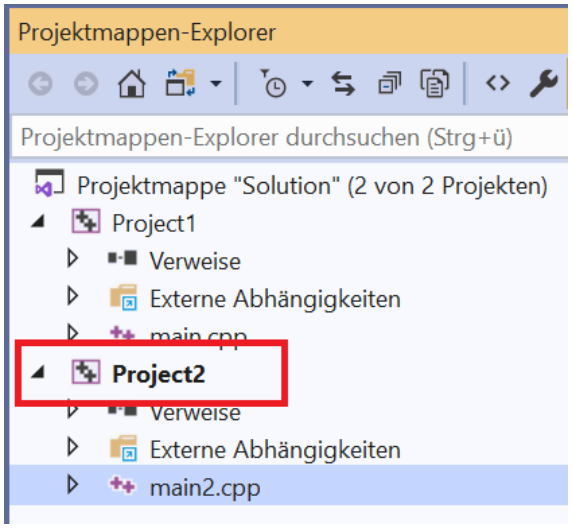

Abb. 4-18 : Das Startprojekt wird fett dargestellt

4.3 C++23 und C++26

Das Microsoft Visual Studio 2022 unterstützt defaultmäßig C++14 ist. Möchten Sie C++17, C++20, C++23 (soweit vorhanden), oder vielleicht sogar die ersten C++26 Features nutzen (und das wollen wir), dann müssen Sie dies explizit für jedes Projekt aktivieren. Sie öffnen die Projekteigenschaften mit dem Kontext-Menü auf dem Projekt im Projektmappen-Explorer.

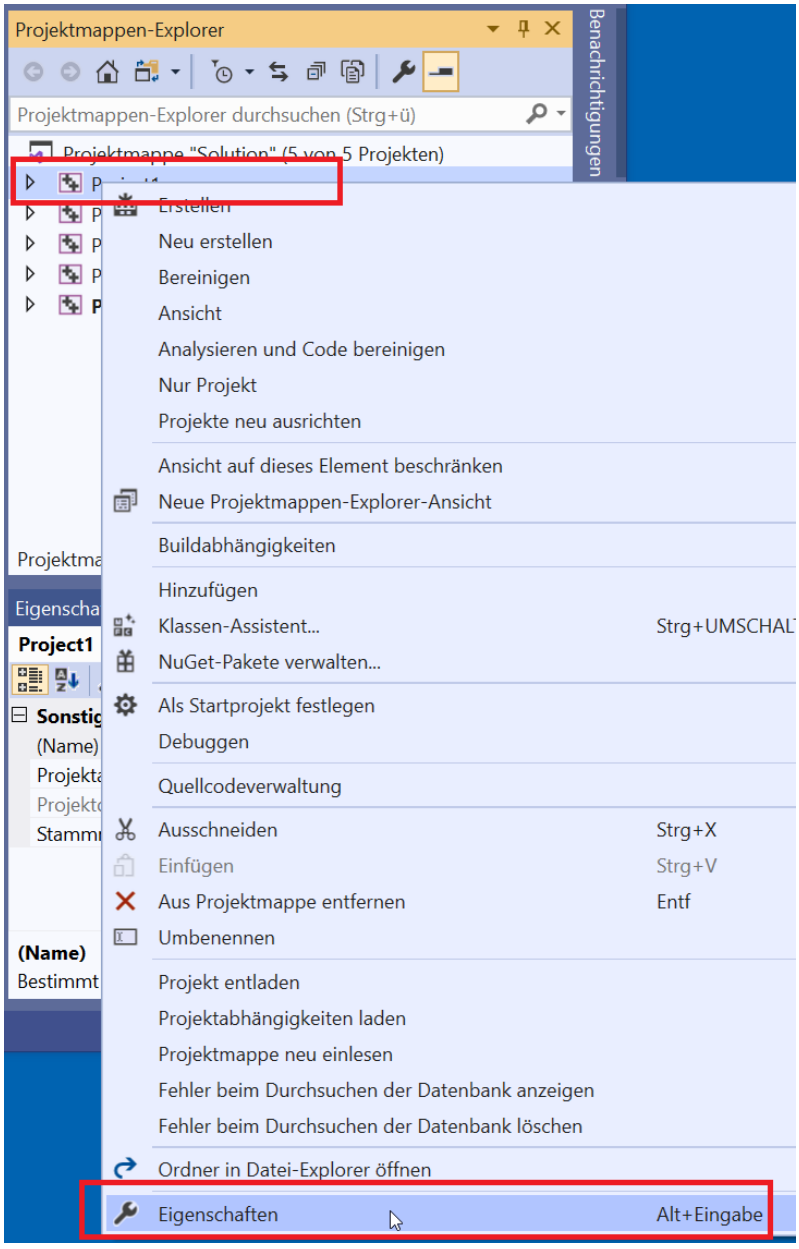

Abb. 4-19 : Projekteigenschaften im Kontext-Menü des Projektmappen-Explorer

Im oberen Bereich des Projekteigenschaften-Dialogs selektieren Sie bitte sowohl „Alle Konfigurationen“ als auch „Alle Plattformen“. In den Eigenschaften wählen Sie „Vorschau – Features aus dem aktuellem…“ bzw. die Option „/std:c++latest“ aus.

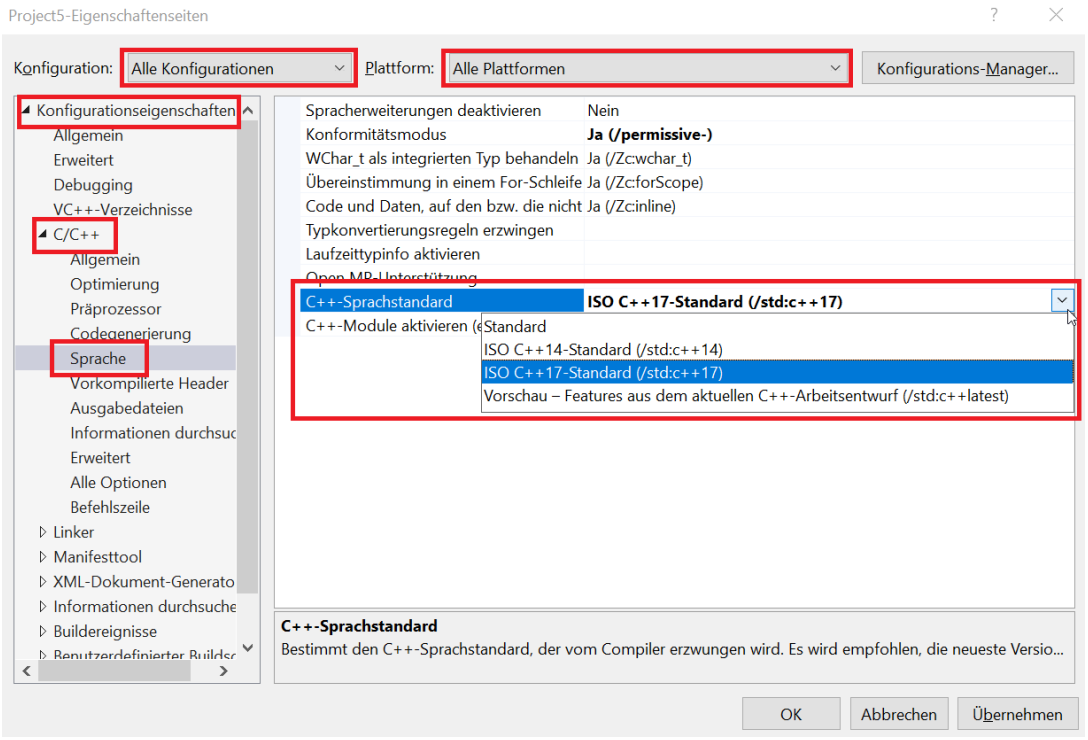

Abb. 4-20 : Projekteigenschaften auf den neusten C++ Standard stellen

4.4 Alternative „Hallo Welt“ Variante in C++23

In C++23 kann man das „Hallo Welt“ Programm auch ganz anders, viel moderner, schreiben.

4.4.1 Module statt Header

- In C++20 sind u.a. die sogenannten Module als moderne Alternative zu den 60 Jahre alten Headerdateien eingeführt worden. Module bieten einige Vorteile gegenüber Headern an. Wir werden beide Themen (sowohl die Header, als auch die neuen Module) aus Zeitmangel leider nicht in der Vorlesung besprechen können.
- In C++23 ist dann das Modul-Konzept auch für die C++ Standard-Bibliothek eingeführt worden. Das aktuelle Microsoft Visual Studio 17.7.4 unterstützt dies schon. Achtung, neben der Einstellung auf mindestens „C++23“ für den C++ Standard (siehe letztes Kapitel) muss dies auch noch in den Eigenschaften unter „Konfigurationseigenschaften“, „C/C++“, „Sprache“ und „ISO C++23 Standardbibliotheksmodule erstellen“ aktiviert werden.
- Damit können die „#include“ Anweisungen durch ein „import std;“ ersetzt werden.

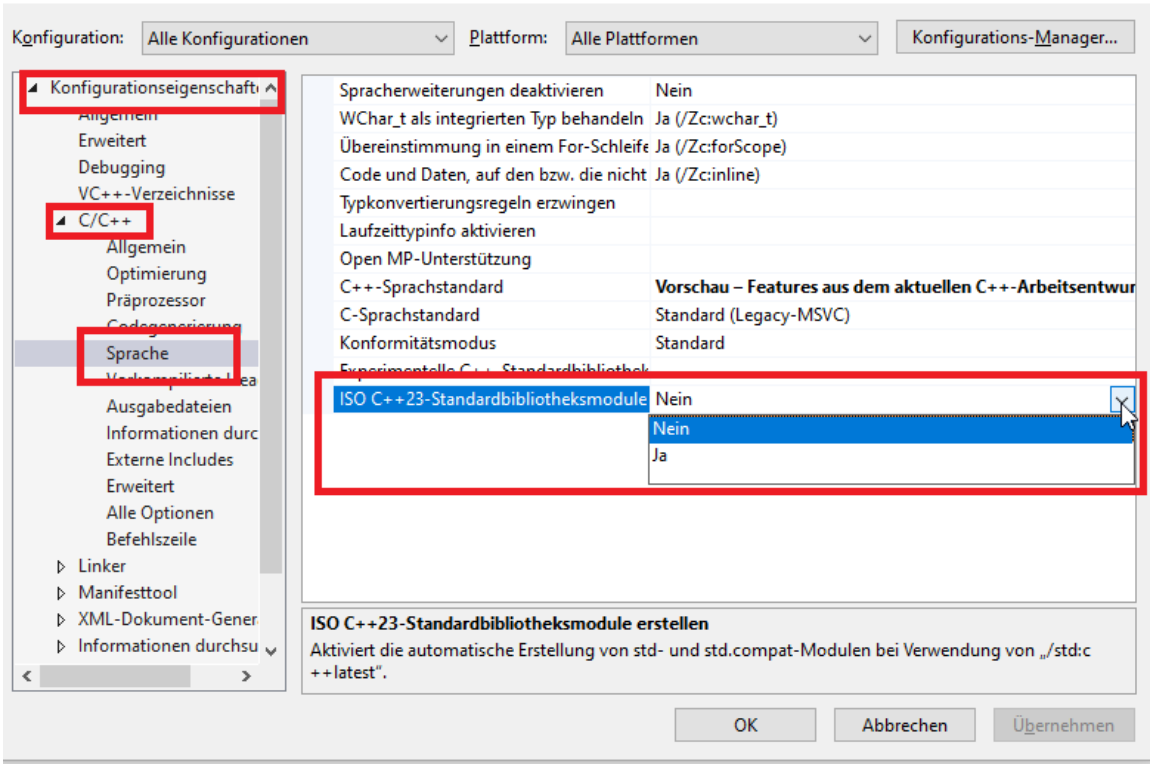

Abb. 4-21 : C++23 Standardmodule aktivieren

4.4.2 Formatierungen und Ausgabe

- Seit C++98 wird die Ausgabe mit „cout“ und dem Ausgabe-Operator „<<“ gemacht. Formatierungen (siehe Teil 3 des C++ Skripts) werden dabei mit Manipulatoren und Element-Funktionen durchgeführt. Leider ist dies sehr aufwändig und nicht gut parametrisierbar.
- Die C++ Community war hiermit nie sehr glücklich. Darum haben sich schon sehr früh Fremdbibliotheken etabliert, die eine bessere Formatierung anbieten.
- Mit C++20 ist endlich eine gute „Format“ Bibliothek in die C++ Standard-Bibliothek eingeflossen, die einen großen Fortschritt darstellt.
- Leider hat die C++20 Format-Bibliothek in Verbindung mit „cout“ den Nachteil, dass die Formatierung den Umweg über „Strings“ nehmen muss. Dies ist nicht effizient.
- Darum wurden in C++23 mehrere spezielle Print-Funktionen (u.a. „print“ und „println“) eingeführt, die die Formatierung und Ausgabe direkt übernehmen, und damit u.a. „cout“ ersetzen bzw. ergänzen.

4.4.3 Ergebnis

Schreibt man nun ein Programm mit Modulen und „println“, so sieht der „Hallo Welt“ Code folgendermaßen aus:

```
import std;

int main()
```

```
{
    std::println("Hallo Welt");
}
```

4.5 Weitere Themen

Die folgenden Themen sind für den Anfang auch wichtig, finden sich aber noch nicht in der Doku. Wir werden sie in der Vorlesung besprechen:

- Internationalisierung und Zeichensätze
- Debug- und Release-Mode
- Undefined Behaviour (UB)