Leibniz Universität IT Services

Leibniz Universität Hannover

C++ - Einführung in die Programmiersprache

Handbücher des IT Services

- C++ für C-Programmierer
- C und C++ für Java-Programmierer

Bücher

- Bjarne Stroustrup: Programming. Principles and Practice Using C++. Addision Wesley.
- Dirk Hardy: C++ für IT-Berufe.
- Ulla Kirch & Peter Prinz: C++ - Lernen und professionell anwenden.
- Ulla Kirch & Peter Prinz: C++. Das Übungsbuch.
- Ulrich Breymann: Der C++-Programmierer: C++ lernen – professionell anwenden – Lösungen nutzen. Carl Hanser Verlag.
- Torsten T. Will: C++. Das umfassende Handbuch. Rheinwerk.

Bücher zur Standardbibliothek

- Rainer Grimm: C++-Standardbibliothek - kurz & gut. O'Reilly. 1. Auflage.
- Peter Van Weert & Marc Gregoire: C++ Standard Library Quick Reference. APress.

PDF-Dokumente im Netz

- TU Braunschweig: https://www.tu-braunschweig.de/Medien-DB/statik/software/einfuehrung_in_cpp.pdf
- Universität Graz: https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf
- DHBW Stuttgart: http://wwwlehre.dhbw-stuttgart.de/~kfg/cpp/cpp.pdf

Tutorials im Web

- http://www.stroustrup.com/C++.html
- http://www.cplusplus.com/doc/tutorial/
- http://www.cprogramming.com/tutorial.html
- http://www.learncpp.com/
- http://www.etp.physik.uni-muenchen.de/kurs/Computing/ckurs/cxx.html
- https://de.wikibooks.org/wiki/C%2B%2B-Programmierung

Zusammenfassung

- http://www.eg.bucknell.edu/~kvollmay/caps_s2010/C++_summary.pdf

Spielerisches lernen

- https://www.codingame.com/start

STL und Standard-Library

- Erweiterung des Standard-Sprachumfanges von C++.
- STL (Standard Template Library). Definition von Vorlagen für Funktionen und Klassen.
- C++ Standard-Library. Erweiterung des Sprachumfangs von C++. Standardisierte Bibliothek von Funktionen.

Informationen im Web

- http://de.cppreference.com/w/cpp/header
- http://gcc.gnu.org/onlinedocs/libstdc++/
- http://www.cplusplus.com/reference/clibrary/
- https://de.wikibooks.org/wiki/C%2B%2B-Programmierung/_Die_STL
- http://www.hpc.canterbury.ac.nz/UCSC%20userdocs/ForUCSCWebsite/C/AIX/standlib.pdf

Style Guides

- http://www.stroustrup.com/bs_faq2.html
- http://stroustrup.com/JSF-AV-rules.pdf
- http://geosoft.no/development/cppstyle.html
- https://gcc.gnu.org/wiki/CppConventions
- https://users.ece.cmu.edu/~eno/coding/CppCodingStandard.html
- http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#S-naming
- http://cs.stmarys.ca/~porter/csc/ref/cpp_style.html
- https://google.github.io/styleguide/cppguide.html
- https://isocpp.org/wiki/faq/coding-standards

C++

- ISO genormte Programmiersprache.
- Objektorientierte und generische Programmierung.
- Komplexe, flexible Programmiersprache.
- Hardware-nahe Programmierung.

Geschichte

- Ca. 1979 arbeitet Bjarne Stroustrup an einer Programmiersprache unter dem Namen „C mit Klassen“.
- 1983 ändert sich der Name der Programmiersprache in C++.
- 1985 wird die erste Referenz zu der Programmiersprache von Bjarne Stroustrup veröffentlicht.
- 1990 wird Borland's Turbo C++ -Compiler als erste kommerzielles Produkt für die Programmiersprache angeboten.

Einsatzmöglichkeiten

- Treiber für die verschiedenste Hardware.
- Entwicklung von Systemprogrammen.
- Zeitkritische Spiele.
- Eine Liste von Beispielen finden Sie unter http://www.stroustrup.com/applications.html.

Standard

- C++98. ISO/IEC 14882:1998. Publiziert im Jahr 1998. Erste Standardisierung der Programmiersprache. Ergänzung der Norm im Jahr 2003.
- C++11. ISO/IEC 14882:2011. Publiziert im April 2011. Ersetzung der alten Versionen.
- C++14. ISO/IEC 14882:2014. Publiziert im Januar 2015. Erweiterungen und Ergänzungen zu C++11.
- C++17. ISO/IEC 14882:2017. Publiziert im März 2017. Die Änderung werden auf der Webseite https://isocpp.org/files/papers/p0636r0.html beschrieben.
- Aktueller Status: https://isocpp.org/std/status

Mittel, um ein C++-Programm zu schreiben

- Ein Texteditor wie Notepad oder eine Entwicklungsumgebung zum Schreiben des Codes.
- Ein Compiler, um aus den geschriebenen Code ein ausführbares Programm zu erstellen.
- Hinweis: Zuerst wird der Compiler und dann die IDE installiert.

Compiler

- Der Compiler übersetzt den, für einen Menschen lesbaren Programmtext in Maschinencode.
- Fehler in der Syntax des Codes führen zu einem Abbruch.
- Ohne den Compiler wird Code in C++ nicht lauffähig.

… für das Betriebssystem Linux / Unix

- Der Compiler gcc (https://www.gnu.org/software/gcc/) ist auf dem Betriebssystem Linux vorinstalliert.
- C++17 wird ab der Version gcc7 unterstützt.

… für das Betriebssystem Windows

- Die Compiler Mingw (http://mingw.org/) ist ein Derivate des Gnu-Compilers für Windows. Hinweis: Der Compiler sollte immer direkt unter dem Laufwerk C: abgelegt werden.
- Eine weitere Möglichkeit ist cygwin (http://www.sourceware.org/cygwin/).

Setzen der Systemvariable

- Anmeldung unter Windows als Administrator.
- *Systemsteuerung – System und Sicherheit – System. Erweiterte Systemeinstellungen*. Registerkarte *Erweitert*. Schaltfläche *Umgebungsvariablen*.
- Auswahl der Systemvariablen *PATH*. In den angegebenen Pfaden sucht die Befehlszeile nach ausführbaren Dateien.
- Schaltfläche *Bearbeiten*. Ergänzung um den Pfad *C:\MinGW\bin;*

Update von mingw

- Öffnen eines Terminals oder Konsole.
- Eingabe von *mingw-get update* in die Konsole. Die neuesten Pakete werden vom Internet geladen.
- Eingabe von *mingw-get upgrade*. Der Compiler wird erneuert.

Schreiben von Code

- Texteditor. Einige Texteditoren wie NotePad bieten ein farbliche Hervorhebung der Syntax der Programmiersprache an. Keinerlei Hilfe beim Compilieren des Programms.
- Entwicklungsumgebungen (IDE). Viele Einstellungsmöglichkeiten. Grafische Oberfläche für die Eingabe des Codes.

Entwicklungsumgebungen (IDE)

- ? Apache NetBeans (http://https://de.wikibooks.org/wiki/C%2B%2B-Programmierung/)
- Code::Blocks (http://www.codeblocks.org/)
- CodeLite (https://codelite.org/)
- Dev-C++ (http://orwelldevcpp.blogspot.com/)
- KDevelop (http://www.kdevelop.org/)
- Visual Studio Code (https://code.visualstudio.com/)

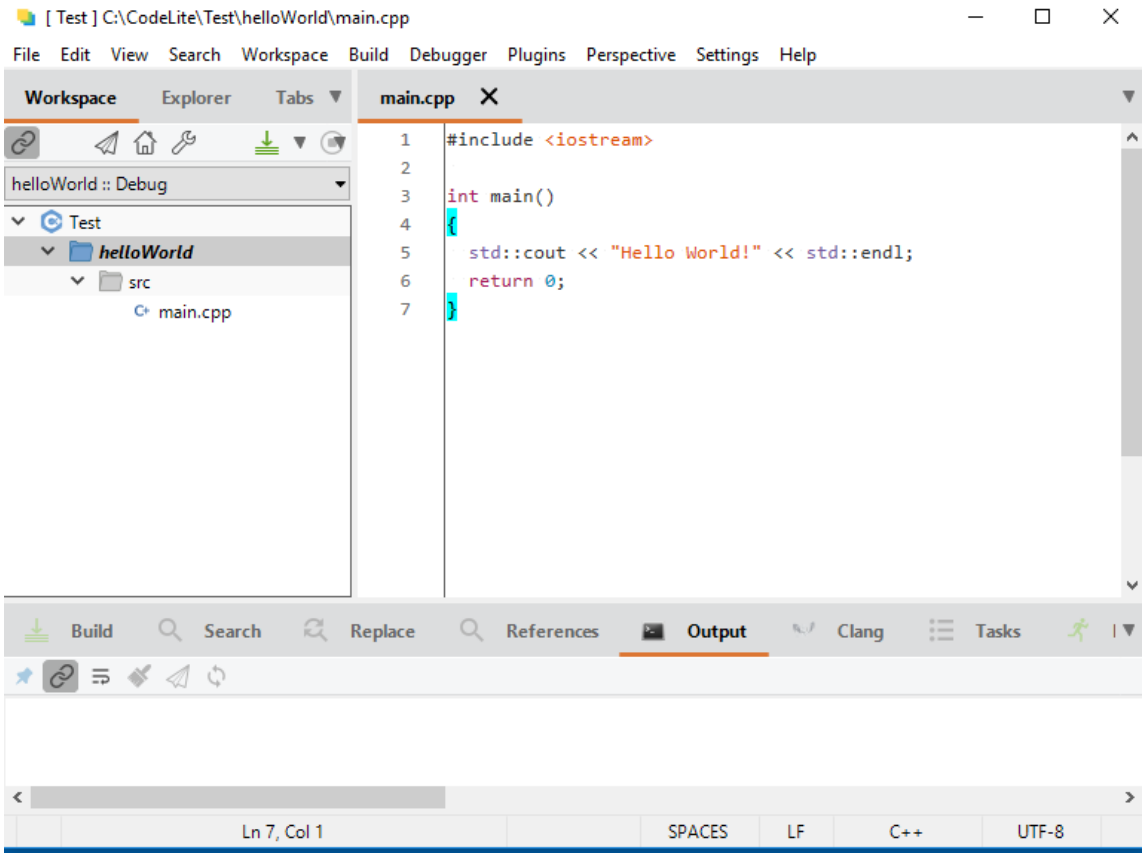
Hinweise zu Apache NetBeans

- Zur Erzeugung der IDE aus einem zip-File wird Oracles Java oder Open JDK und Apache Ant benötigt.
- Ab Apache NetBeans 10.0 ist kein Plug-In mehr für die Programmiersprache C++ vorhanden. In einer zukünftigen Versionen soll es wieder ein Plug-In geben.

Online-Editoren zum Testen von Code

- http://coliru.stacked-crooked.com/
- https://godbolt.org/
- http://cpp.sh/
- https://wandbox.org/
- https://repl.it/repls/UniqueOffbeatCgi

IDE im Kurs: CodeLite



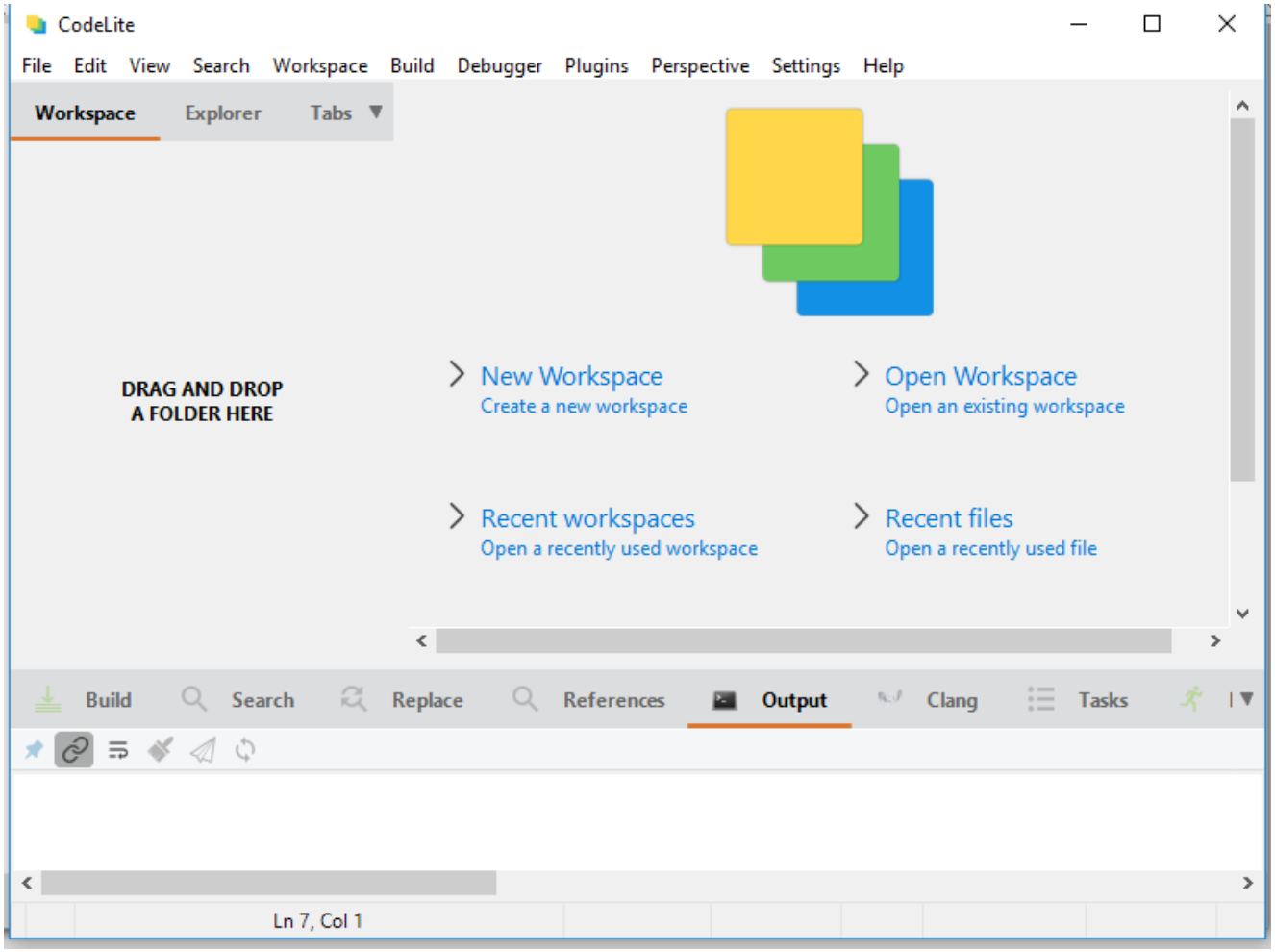
Hilfe zum Editor

- https://www.it.iitb.ac.in/frg/wiki/images/2/27/CodeLite_Manual.pdf
- https://wiki.codelite.org/pmwiki.php

Start von CodeLite

- *Icon auf dem Desktop.*
- Windows 8 und höher: Suchen …
- Im Installationspfad von CodeLite: codelite.exe.

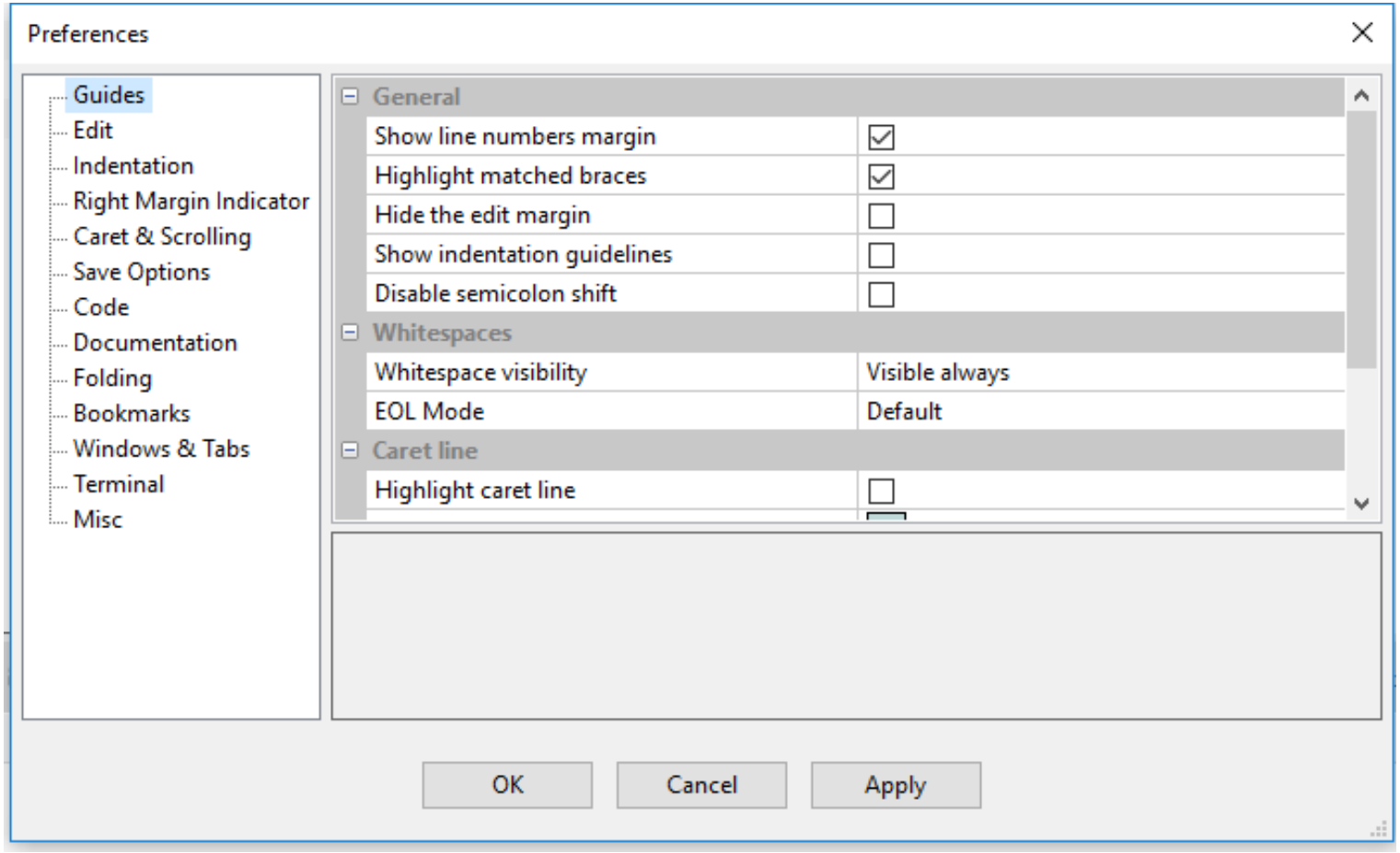
Startseite


CodeLite
File Edit View Search Workspace Build Debugger Plugins Perspective Settings Help
Workspace Explorer Tabs
DRAG AND DROP
A FOLDER HERE
New Workspace
Create a new workspace
Open Workspace
Open an existing workspace
Recent workspaces
Open a recently used workspace
Recent files
Open a recently used file
Build Search Replace References Output Clang Tasks
Ln 7, Col 1

Einstellungen

- *Settings – Preferences.*
- *Settings – Colours and Fonts.*
- *Settings – Build Settings*
- *Settings – GDB Settings.*

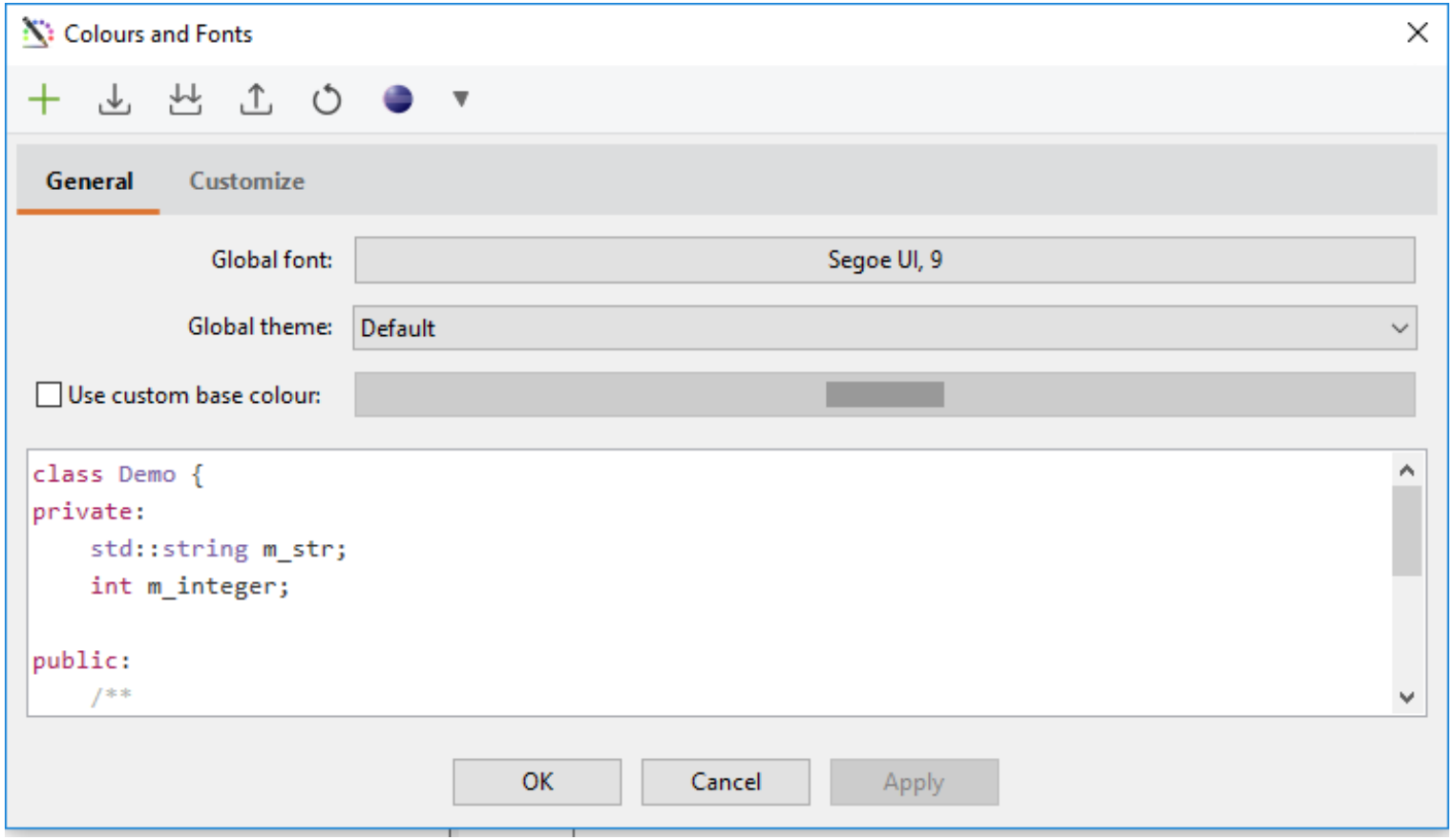
Settings - Preferences



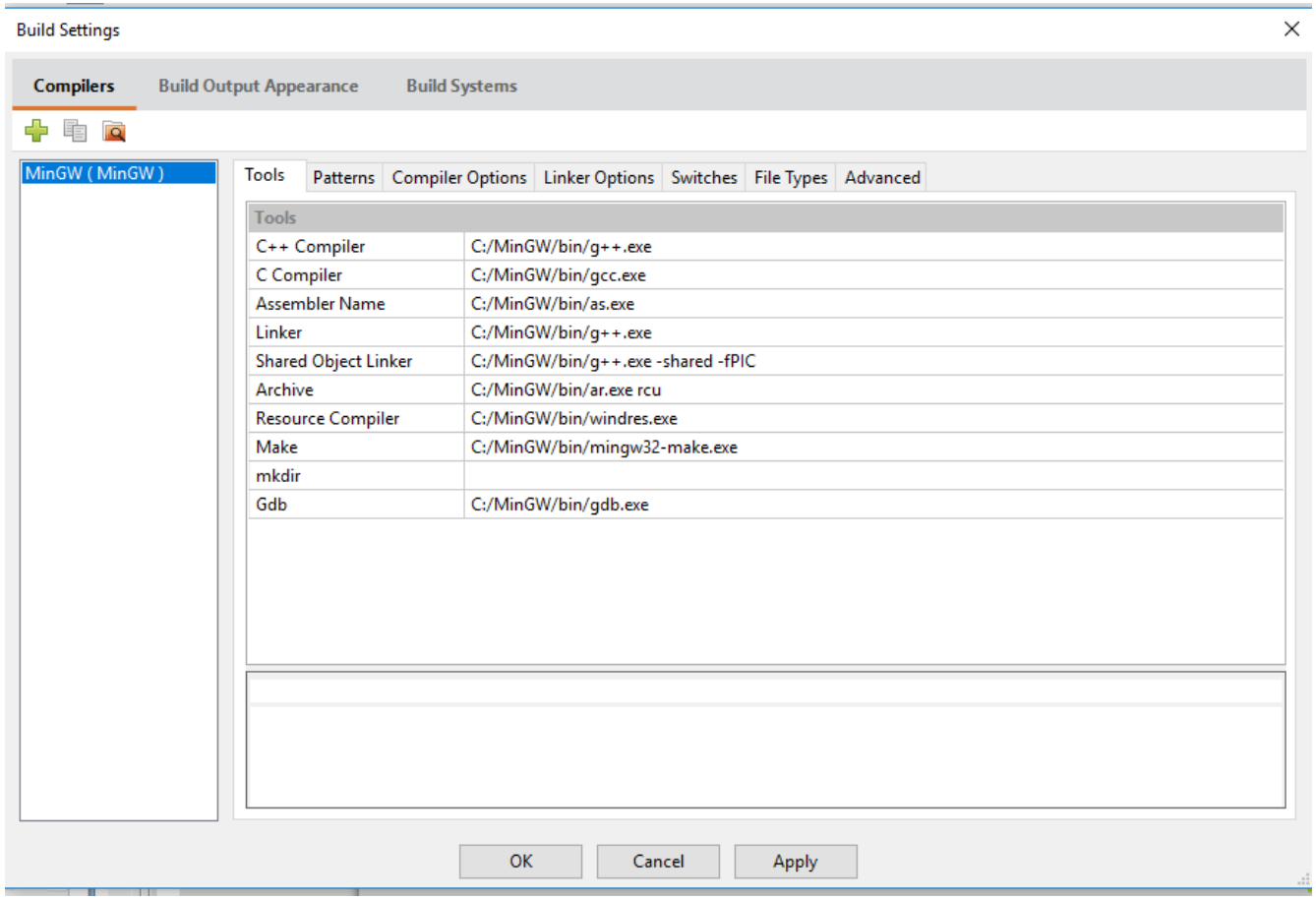
Einstellungsmöglichkeiten

- Ordner *Guides*: Anzeige von Zeilennummern.
- Ordner *Indentation*: Einstellung von Einzügen mit Hilfe von Tabulatoren.
- Ordner *Terminal*: Welche Konsole wird als Standard-Ausgabe genutzt?
- Ordner *Misc*: Welche Eingabe-Zeichenkodierung wird genutzt? Standardmäßig wird Unicode 8 genutzt.

Settings – Color and Fonts



Settings – Build Settings



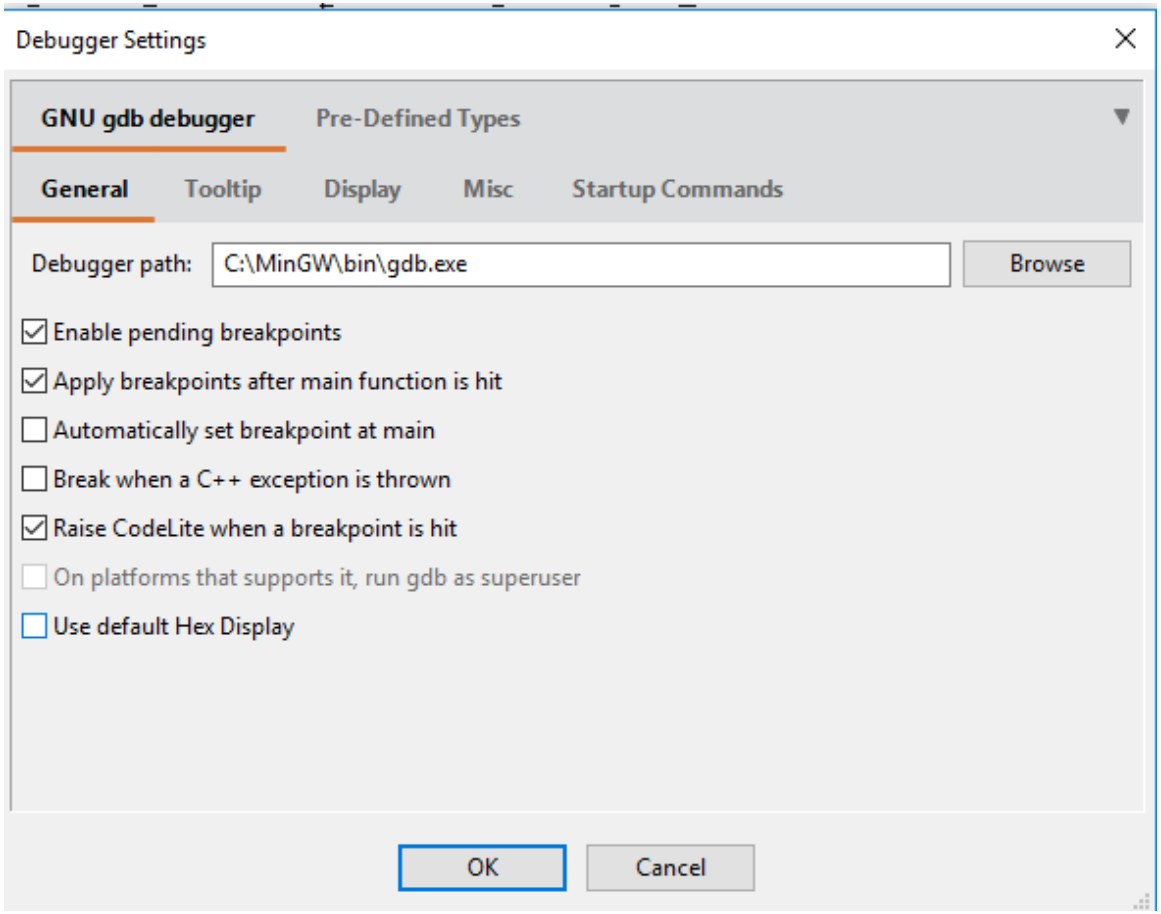
Compilers

- Zeile C++ *Compiler*. Speicherort des C++- Compilers.
- Zeile *Linker*. Mit welchen Tool werden Bibliotheken und so weiter mit dem Code verbunden?
- Zeile *Make*. Liest eine Datei mit Instruktionen, wie aus dem Code automatisiert eine ausführbare Datei erstellt wird.

Hinweise

- Der Compiler für C++ muss installiert sein. Andernfalls kann kein lauffähiges Programm erzeugt werden.
- Es können mehrere Compiler auf dem Rechner installiert werden.

Settings - GDB-Settings



GDB (Gnu Debugger)

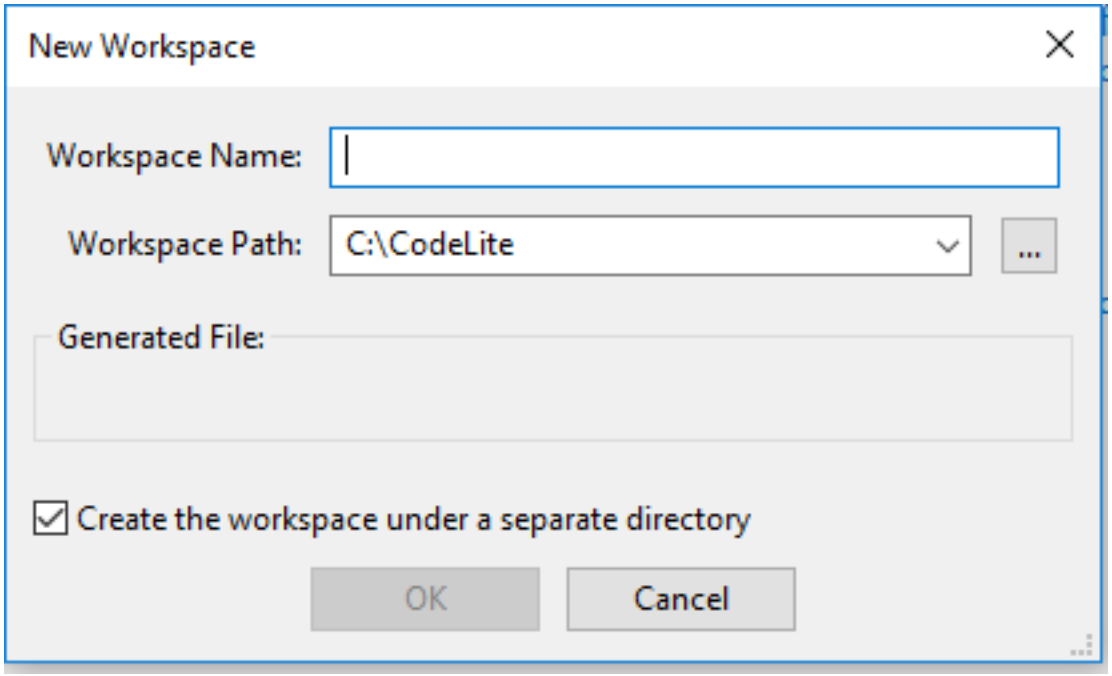
- Debuggen eines Programms. Fehlersuche in einem Programm.
- Einstellungen zum Tool (Debugger).

Schreiben von Code

- Im ersten Schritt wird ein Arbeitsbereich angelegt (*File* – *New* – *New Workspace*)
- In dem Arbeitsbereich wird ein Projekt angelegt (*File* – *New* – *New Project*). In diesem Projekt werden alle Dateien gesammelt, die zur Abbildung einer Aufgabe in C++ gehören.
- Jedes Projekt hat eine Datei *main.cpp* in dem Ordner *src*. Diese Datei stellt die Steuerungszentrale des Projekts dar. Im ersten Teil des Kurses wird der Code nur in diese Datei geschrieben.
- Sobald die Aufgabe in viele kleine Aktivitäten unterteilt wird, werden dem Projekt Codedateien hinzugefügt. In diesem Codedateien wird mit Hilfe von C++ die Aktion beschrieben.

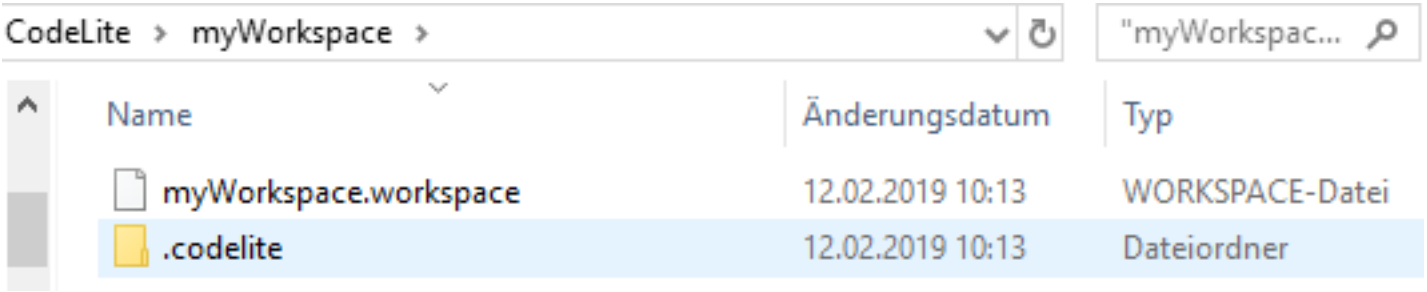
Neuanlage des Arbeitsbereichs

- *New Workspace* auf der Startseite.
- *File – New – New Workspace*.



Workspace (Arbeitsbereich)

- Darstellung eines Schreibtisches.
- „Ordner“ für große Programme.
- Informationen zum Erstellen eines ausführbaren Programms.



Hinweise zum Speicherort

- Der Pfad sollte keine Leerzeichen, Umlaute etc. enthalten.
- Die Speicherung eines Arbeitsbereichs unter User / … / Dokumente kann zu einem Fehler bei der Ausführung des Programms führen.

… schließen

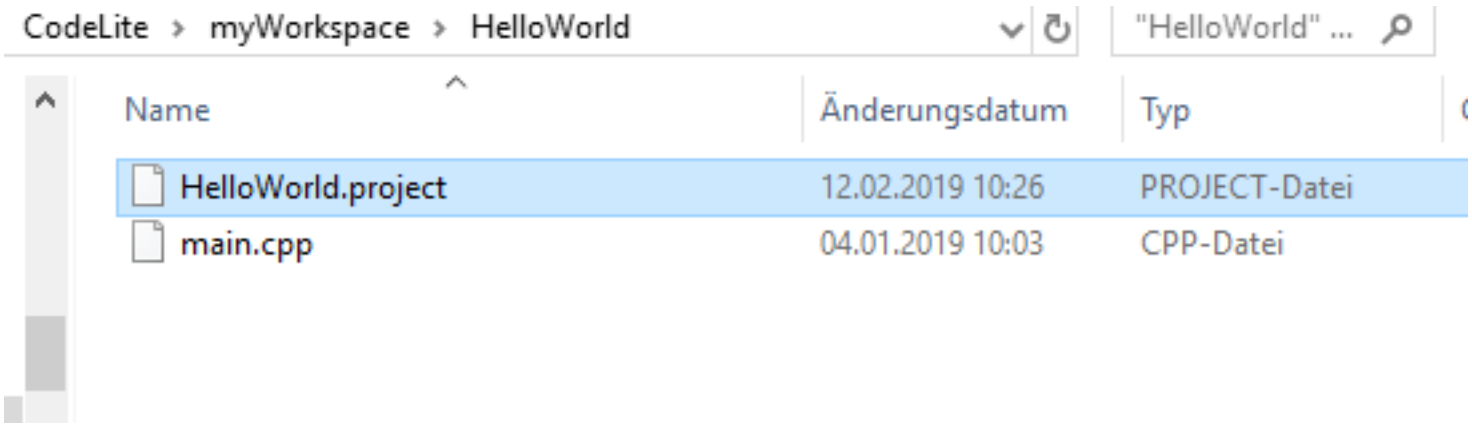
- *Workspace – Close Workspace*.
- *File – Close Workspace*.

Neuanlage eines Projekts

- *File – New – New Project* oder *Workspace – New Project*.
- Im ersten Schritt wird eine Vorlage für das Projekt gewählt. In diesem Kurs: Auswahl der Vorlage *Console – Simple executable (g++)*.
- Im zweiten Schritt wird der Name des Projekt eingegeben und ein Speicherort gewählt.
- Im dritten Schritt wird der gewünschte Compiler und Debugger für das Projekt ausgewählt.

Projekt

- Abbildung der, in C++ zu lösenden Aufgabe.
- Sammlung von allen Dateien, die zu einem ausführbaren Programm gehören.
- Steuerungszentrale *main.cpp*. Start des ausführbaren Programms.
- Ein Arbeitsbereich kann mehrere Projekte enthalten.



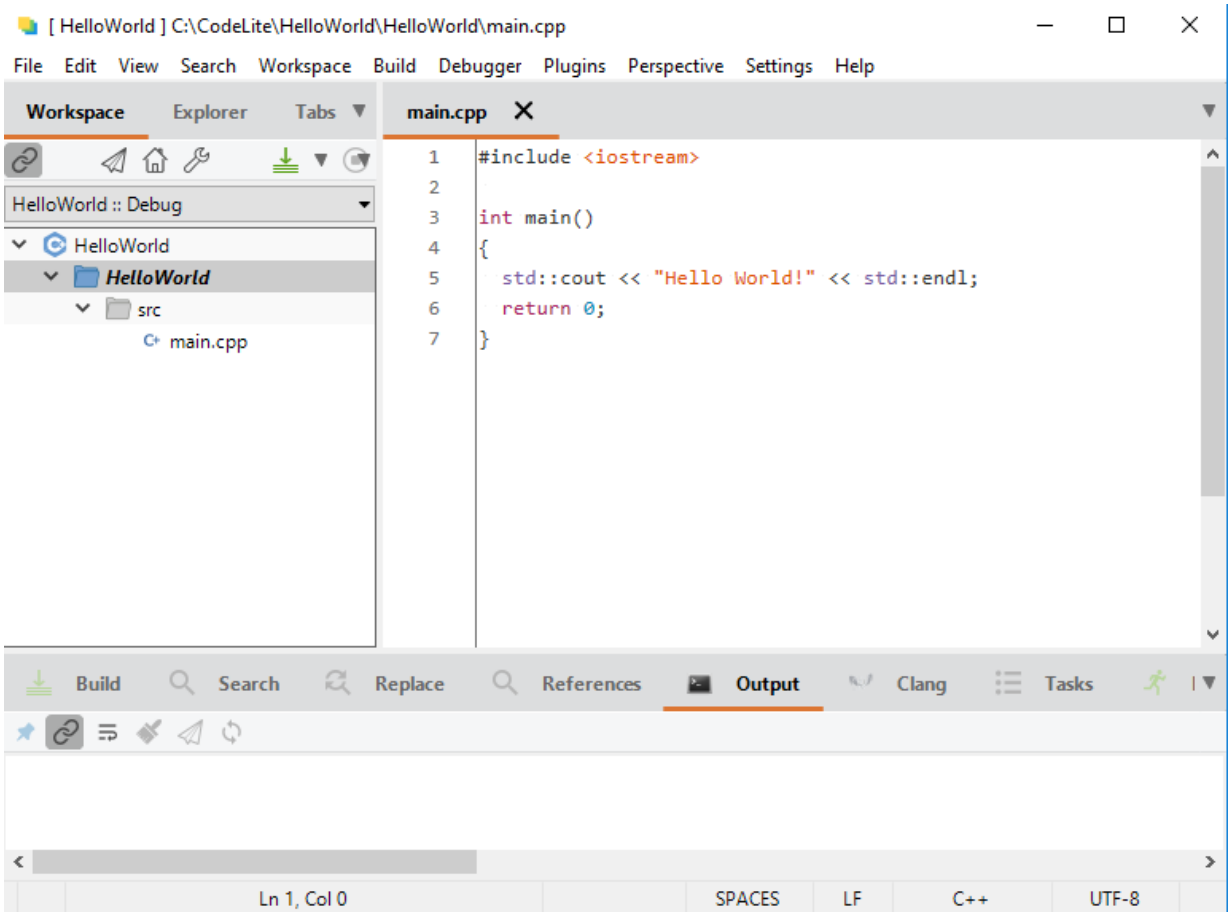
Vorlage: *Console – Simple executable (g++)*

- Einfache maschinennahe Programme.
- Ausgabe von Werten auf die Konsole oder in einem Terminal.
- Keine grafische Gestaltung des Programms.

Auswahl eines Projektnamens

- Der Name darf nur die Groß- und Kleinbuchstaben a…z, die Zahlen 0...9 und den Unterstrich enthalten.
- Der Name sollte die Aufgabe des C++-Programms widerspiegeln.
- Nutzung der Kamel-Notation: Projektnamen, die aus verschiedenen Wörtern zusammengesetzt werden. Jedes Wort in dem Projektnamen beginnt mit einem Großbuchstaben.

Workspace und Projekt in CodeLite

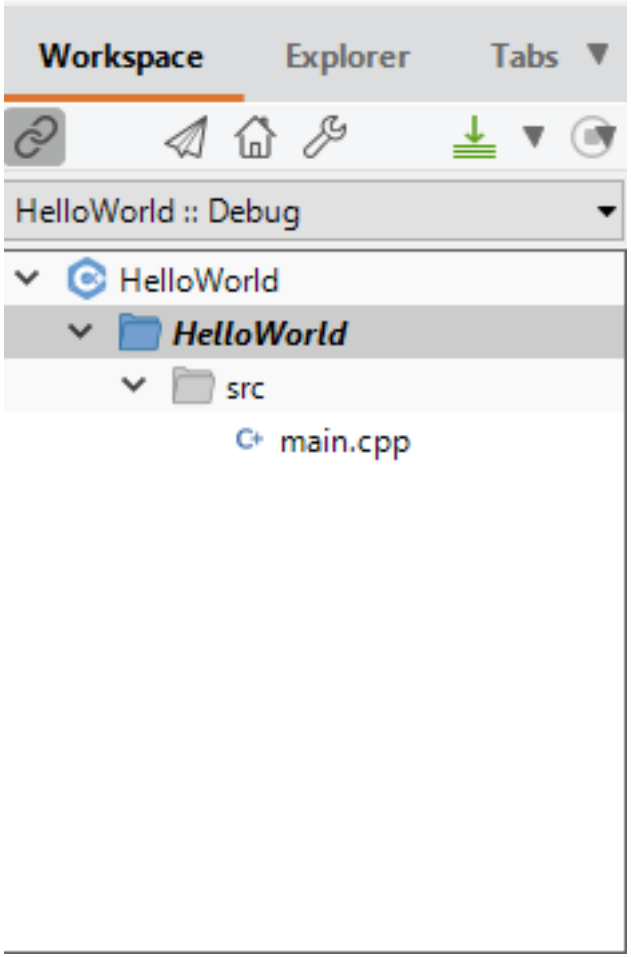


Elemente der IDE

- Titelleiste. Anzeige des Namens des gewählten Arbeitsbereiches.
- Menüleiste. Das Menü *Workspace* enthält Befehle zum Arbeiten mit dem Arbeitsbereich.
- *Workspace pane*. Hierarchische Abbildung der Dateien und Ordner in einem Arbeitsbereich.
- *Output pane*. Dieser Abschnitt wird automatisch beim Erstellen eines C++-Programms eingeblendet.
- Statuszeile.

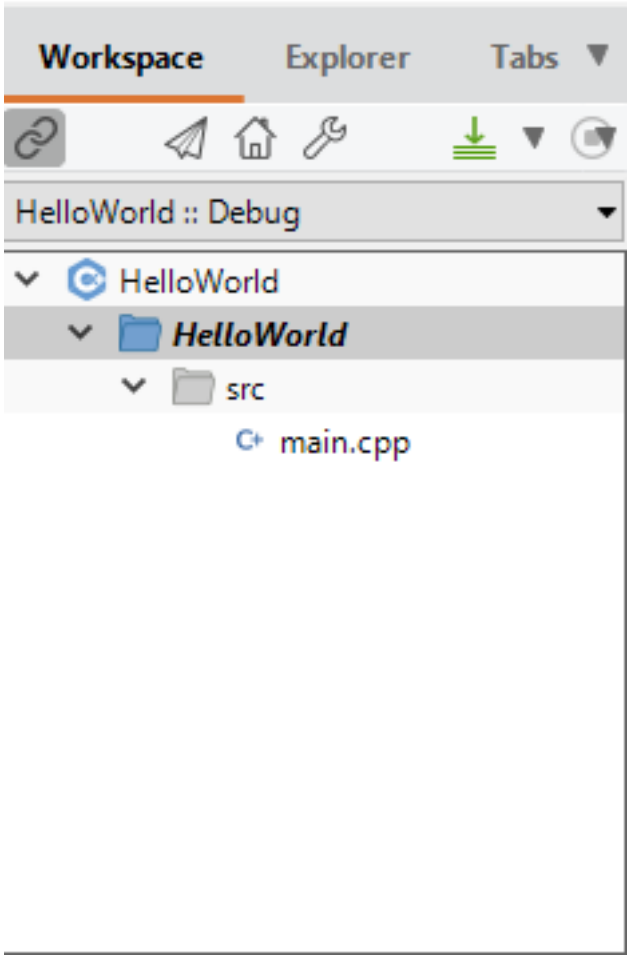
Workspace pane

- Ordner in der obersten Ebene: Name des Arbeitsbereichs. In dem Fenster kann immer nur ein Arbeitsbereich abgebildet werden.
- Ordner in der zweiten Ebene: Projektordner, zugeordnet zu einem Arbeitsbereich. Ein Arbeitsbereich kann beliebig viele Projektordner enthalten.
- Ordner in der dritten Ebene: Ordner und Dateien des Projekts in Abhängigkeit der gewählten Vorlage.



Ordner in einem Projekt

- Jedes Projekt hat verschiedene Unterordner in Abhängigkeit der Vorlage.
- Der Unterordner *src* (Source) enthält immer die Datei main.cpp. Der Quellcode des Projekts wird in diesem Ordner gespeichert.
- Sammlung von Quellcode-Dateien (*.cpp) oder Header-Dateien (*.h).



Einstellungen

- Rechtsklick auf den Namen des Arbeitsbereiches oder eines Projekts.
- Menüeintrag *Settings* am unteren Rand des Kontextmenüs.
- Mit Hilfe der angebotenen Möglichkeiten können Optionen, die nur diesen Arbeitsbereich oder Projekt betreffen, eingestellt werden.

Dateien im Konsolenprogramm

- Quelldateien. Endung „.cpp“. Anweisungen in der Programmiersprache C++. Definitionen von Funktionen und so weiter.
- Header-Dateien. Endung „.h“. „Präambel“ einer Quelldatei. Sie Deklarationen von Funktionen und Klassen.

… im Code-Fenster anzeigen

- Doppelklick auf eine Datei mit der Endung „.cpp“ oder „.h“.
- Das Code-Fenster wird automatisiert rechts vom *workspace pane* geöffnet.
- Der Name Codedatei wird am oberen Rand des Fensters angezeigt. Die aktuell geöffnete Datei ist rot unterstrichen.

Code

- Anweisungen in einer bestimmten Programmiersprache. In diesem Kurs C++.
- Text in einer bestimmten Syntax.
- Text, der von Menschen, die die Sprache können, gelesen werden kann.

Schlüsselworte in C++

- Reservierte Wörter in C++.
- Schlüsselwörter bilden den Sprachkern der Programmiersprache.
- Hinweis: IDEs färben den Code entsprechend eines Schemas ein.

Liste von Schlüsselwörtern

- http://de.cppreference.com/w/cpp/keyword
- https://en.wikibooks.org/wiki/C%2B%2B_Programming/Programming_Languages/C%2B%2B/Code/Keywords

Beispiel „main.cpp“ im Ordner src

```
#include <iostream>

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

Aufbau

Präprozessor-Anweisungen	`#include <iostream>`

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

Präprozessor-Anweisungen

```
#include <iostream>
```

- Beginn mit dem Hash-Zeichen (#).
- Beendigung mit der Zeile und nicht mit einem Semikolon.
- Positionierung am Anfang einer Programmdatei.
- Vor der Kompilierung des Programms werden die Zeilen durch den entsprechenden Text ersetzt.

Beispiel für die Nutzung

```
#include <iostream>
```

- In diesem Beispiel wird vorgefertigter Programmcode eingebunden (`#include`).
- Die Datei `iostream` wird in diesem Beispiel eingebunden. Die Anweisung wird durch den Inhalt der Datei ersetzt.

Dateien zum Einbinden

```
#include <iostream>
```

- Die eingebundene Datei erweitert den Sprachumfang von C++. In diesem Beispiel werden Funktionen zur Ein- und Ausgabe auf die Konsole bereitgestellt.
- Dateien in spitzen Klammer. Bereitstellung durch den Compiler. Dateien aus dem Verzeichnis include. Angabe ohne Dateiendung.

Funktionen in C++

```
#include <iostream>

int main()
{

}
```

- Funktionen beschreiben eine Handlung mit Hilfe von Anweisungen.
- Quelldateien enthalten mindestens eine Funktion.

Funktion main

```
#include <iostream>

int main()
{

}
```

- Speicherung in *main.cpp*.
- Pro Projekt nur einmal vorhanden.
- Steuerungszentrale eines großen Programms.

Aufbau


Parameterliste (hier: leer)
int main( )
Kopf der Funktion
{
std::cout << "Hello World \n";
return 0;
}
Codeblock zu der Funktion
Rückgabe-wert in Abhängigkeit des Funktionstyps

Kopf der Funktion main

```
int main()
```

- Definition einer Schnittstelle nach außen.
- Start des ausführbaren Programms (*.exe) → Aufruf der Funktion `main()`.

Hinweise

- Beachtung der Groß- und Kleinschreibung. Der Name `Main` bezeichnet eine andere Funktion als die Bezeichnung `main`.
- Fehlermeldung „undefined reference to winmain@16“, falls die die Funktion `main()` fehlt.

Parameterliste der Funktion

```
int main()
```

- Übergabe von Werten beim Start an das Programm.
- Runde Klammern: Beginn und Ende der Parameterliste.
- Leere Parameterliste: Es werden keine Werte vom Starter zur Ausführung des Programms benötigt.

Rückgabewert der Funktion

```
int main()

```

- Funktionstyp: Ganzzahl (`int`).
- Lieferung an den Aufrufer der Funktion: eine Ganzzahl.
- Der Rückgabewert dieser Funktion enthält Informationen zum Status des Programms.

Funktionsrumpf

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

- Funktionsrumpf: Beginn und Ende mit den geschweiften Klammern.
- Die geschweiften Klammern fassen Code für die, in der Funktion zu beschreibenden Aktion zusammen.

Anweisungen

```
std::cout << "Hello World!" << std::endl;
return 0;
```

- Beschreibung eines Arbeitsschrittes mit Hilfe von C++.
- Beendigung mit einem Semikolon.
- Häufig steht pro Zeile eine Anweisung.

1. Anweisung im Beispiel

```
std::cout << "Hello World!" << std::endl;
```

- `std::cout`. Standardausgabe, definiert im Standardnamensraum. Deklaration in der Datei `iostream`.
- `std::endl`. Zeilenumbruch, definiert im Standardnamensraum. Deklaration in der Datei `iostream`.
- `"Hello World!"`. Zeichenkette, begrenzt durch die Anführungszeichen.
- Alle drei Elemente sind mit dem Umleitungsoperator `<<` verbunden.

Nutzung von cout und endl

```
std::cout << "Hello World!" << std::endl;
```

- Deklaration in der Datei `iostream`. Die Datei muss durch den Befehl `#include` eingebunden werden. Andernfalls können die Befehle nicht genutzt werden.
- Nutzung von `std`. Befehle werden in einem Namensraum zusammengefasst. `cout` und `endl` sind eindeutig in dem Standard-Namensraum `std` definiert. Die zwei direkt aufeinander folgenden Doppelpunkte verbinden einen Namensraum mit einem Befehl.

Zeichenkette (String)

```
std::cout << "Hello World!" << std::endl;
```

- Beginn und Ende mit den Anführungszeichen.
- Zusammenfassung von alphanumerischen und numerischen Zeichen.
- Beliebig lange Zeichenkette direkt im Code.

Ausgabe auf die Konsole

```
std::cout << "Hello World!" << std::endl;
```

- Der Operator << leitet die Zeichenkette und den Zeilenumbruch (`endl`) auf die Standardausgabe (`cout`) um.
- Als Standardausgabe wird in diesem Kurs eine Konsole oder ein Terminal genutzt. Windows: MS Dos Eingabeaufforderung.

2. Anweisung in dem Beispiel

```
return 0;
```

- Beendigung des Programms.
- Rückgabe des Wertes 0 an den Aufrufer des Programms.
- Die Ganzzahl Null in der Funktion `main()` symbolisiert ein fehlerfrei laufendes Programm.

Einrückungen in Codeblöcken

```
{
   std::cout << "Hello World!" << std::endl;
   return 0;
}
```

```
{std::cout << "Hello World!" << std::endl;return 0;}
```

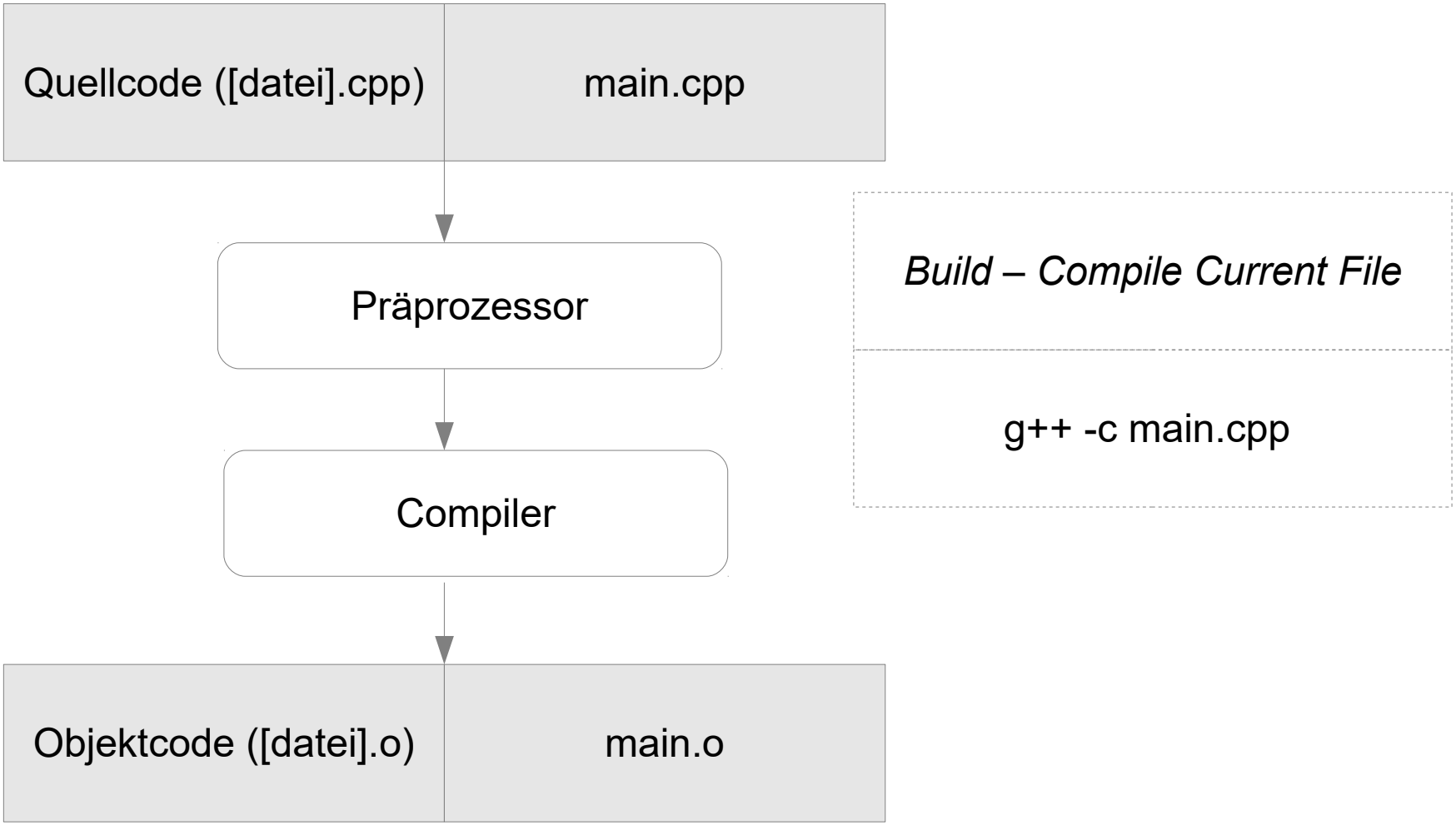
- Beide Codeblöcke erzeugen keinen Fehler.
- Einrückungen erhöhen die Lesbarkeit. Anweisungen, die zu einem Block gehören, werden optisch gekennzeichnet.
- „Eine Zeile = eine Anweisung“ erhöht die Lesbarkeit von Code.

Nutzung von Leerzeichen

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

- Zwischen Schlüsselwörtern, Operatoren und Literalen wie zum Beispiel der Text „Hello World!“ stehen Leerzeichen.
- Zwischen zusammengesetzten Operatoren wie `<<` oder `::` dürfen keine Leerzeichen stehen.
- Zwischen dem Funktionsnamen `main` und der Parameterliste `()` dürfen keine Leerzeichen gesetzt werden.

Kompilierung eines Programms



Präprozessor

- Der Quellcode wird für den Compiler aufbereitet.
- „Suchen und ersetzen“ von Präprozessor-Anweisungen durch die entsprechende Textabschnitte.
- Beispiel: Die Anweisungen `#include <iostream>` wird durch den entsprechenden Inhalt der Bibliothek ersetzt.
- Steuerung der eigentlichen Kompilierung.

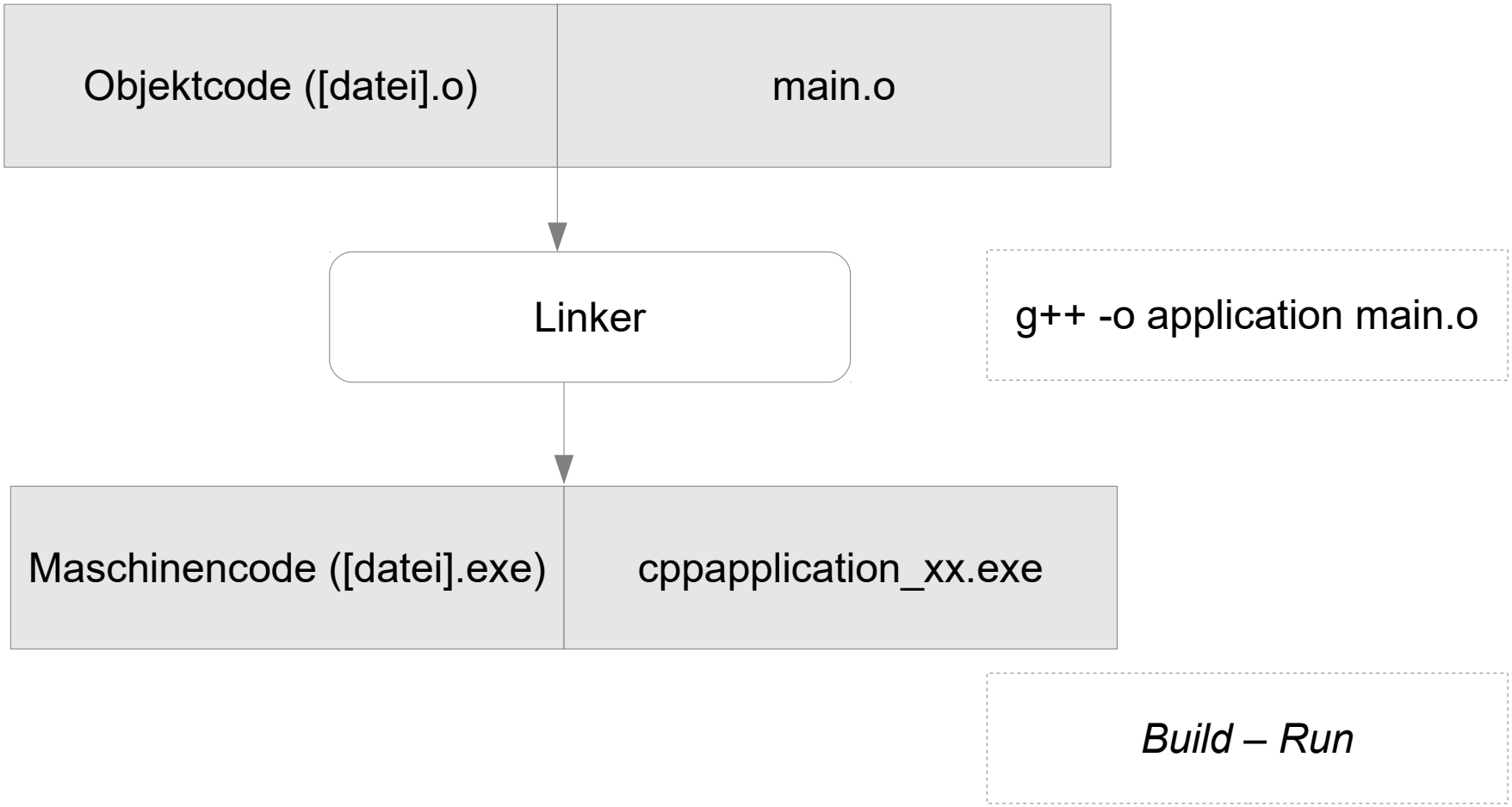
Compiler

- Der Compiler übersetzt den, für einen Menschen lesbaren Programmtext in Maschinencode.
- Falls der Code gegen die Syntax der Programmiersprache verstößt, wird die Kompilierung abgebrochen.

… in CodeLite einstellen

- *Settings – Build Settings*. Diese werden automatisch durch das Icon *Scans* auf der Registerkarte *Compilers* gesetzt.
- Der Pfad zum Compiler wird in der Zeile *C++-Compiler* beschrieben. Mit Hilfe der Schaltfläche am rechten Rand kann der Pfad verändert werden.

Ausführbares Programm erstellen



Linker

- Alle, zu einem Projekt gehörenden Objektdateien werden zu einem ausführbaren Programm verbunden.
- Fehlende Objektdateien verursachen einen Fehler.

… in CodeLite einstellen

- *Settings – Build Settings*. Diese werden automatisch durch das Icon *Scans* auf der Registerkarte *Compilers* gesetzt.
- Der Pfad zum Linker wird in der Zeile *Linker* beschrieben. Mit Hilfe der Schaltfläche am rechten Rand kann der Pfad verändert werden.

Statisches Linken

- Dateien werden einmalig bei der Entwicklung eingebunden.
- Die Schalter *-static-libgcc* stellt eine Bibliothek zur Unterstützung bereit.
- Der Schalter *-static-libstdc*++ stellt Funktionen für die Standards C++11 und C++14 bereit.
- Beide Bibliotheken werden automatisch durch den Schalter *-static* eingebunden.

makefile

- Automatisierung Steuerung der Kompilierung und Verlinkung von C++-Programmdateien.
- Lesen durch das Tool *make*.
- Ablage im Arbeitsbereich.
- In diesem Kurs wird das automatisch von CodeLite erzeugte makefile genutzt.

Einstellungen in CodeLite

- *Settings – Build Settings*. Diese werden automatisch durch das Icon *Scans* auf der Registerkarte *Compilers* gesetzt.
- Der Pfad zu dem Tool make wird in der Zeile *make* beschrieben. Mit Hilfe der Schaltfläche am rechten Rand kann der Pfad verändert werden.

Beispiele

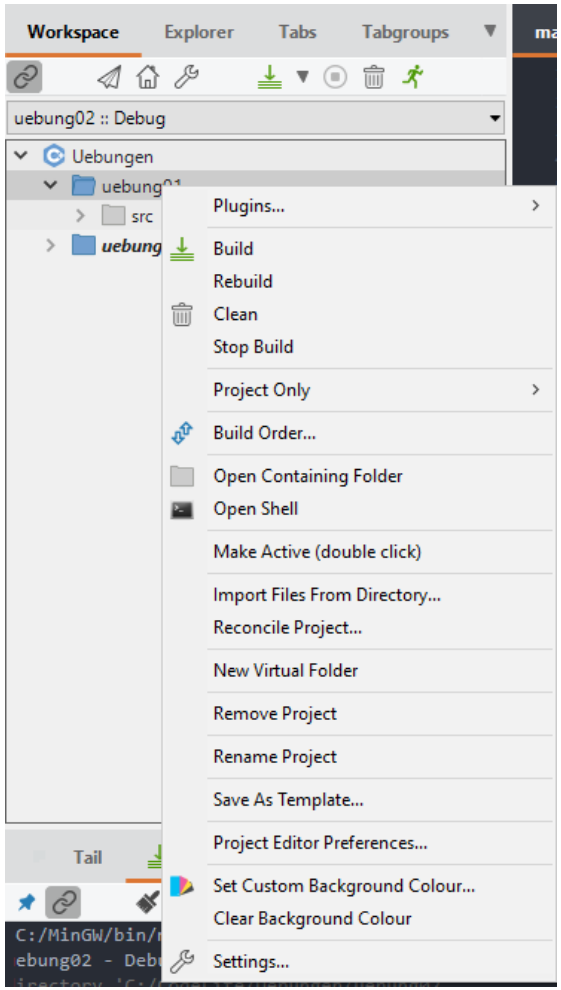
- *C:/mingw32/bin/make.exe* ist im Paket MSYS enthalten. Nutzung unter Unix / Linux.
- *C:/mingw32/bin/mingw32-make.exe* arbeitet mit der MS DOS Eingabeaufforderung zusammen.
- *C:/mingw32/bin/mingw32-make.exe -j4 SHELL=cmd.exe*. Als Konsole wird die MS DOS Eingabeaufforderung übergeben

Bau eines ausführbaren Programms

- *Build – Build Project.*
- Aus dem aktuell gewählten Projekt wird ein ausführbares Programm erstellt.
- Alle benötigten Schritte werden automatisiert durchgeführt.

Mehrere Projekte in einem Arbeitsbereich

- Mausklick mit rechts auf den Projektnamen.
- *Build* im Kontextmenü.
- Oder: *Project Only* – *Build* im Kontextmenü.



Ausführen des Programms

- *Build* – *Run* führt immer das aktive Projekt aus.
- Das aktive Projekt wird durch Fettschrift gekennzeichnet.

Setzen des aktiven Projekts

- Doppelklick auf den Projektnamen.
- Oder: Mausklick rechts auf den Projektnamen. *Make Active* im Kontextmenü.