

C++ Kurs
TU Dresden
Fakultät für Informatik

Maximilian Starke
Student der TU Dresden

5. April 2017

Inhaltsverzeichnis

1	Einrichtung	2
1.1	ISO C++	2
1.1.1	Allgemeines	2
1.1.2	Versionen	2
1.2	Dateien in einem C++ Projekt	3
1.3	Compiler	4
1.4	IDEs	5
1.4.1	JA oder NEIN	5
1.4.2	IDEs im Überblick	5
1.5	Referenzen	5
1.6	The Hello World	10
1.6.1	Das erste kleine Programm	10
1.6.2	Ein paar Werkzeuge	10
1.6.3	Programmierstil	11
2	Datentypen in C++	12
2.1	primitive Datentypen	12
2.2	Einige Operatoren	12
2.3	Casts	12
2.4	Zusammengesetzte Datentypen	12
2.4.1	Arrays	12
2.4.2	Records und Klassen	12
2.4.3	Containerklassen	12
2.5	Klassen	12
2.5.1	Konstruktoren	12
2.5.2	Vererbung	12
2.5.3	Polymorphie	12
3	Strukturierte Programmierung	13
3.1	Kontrollstrukturen	13
3.2	Funktionen	13
3.3	Operatoren	13
3.4	Modularisierung	13
4	Zusätzliche Features	14
4.1	Templates	14
4.2	Exceptions	14
4.3	Multithreading	14

Kapitel 1

Einrichtung

1.1 ISO C++

1.1.1 Allgemeines

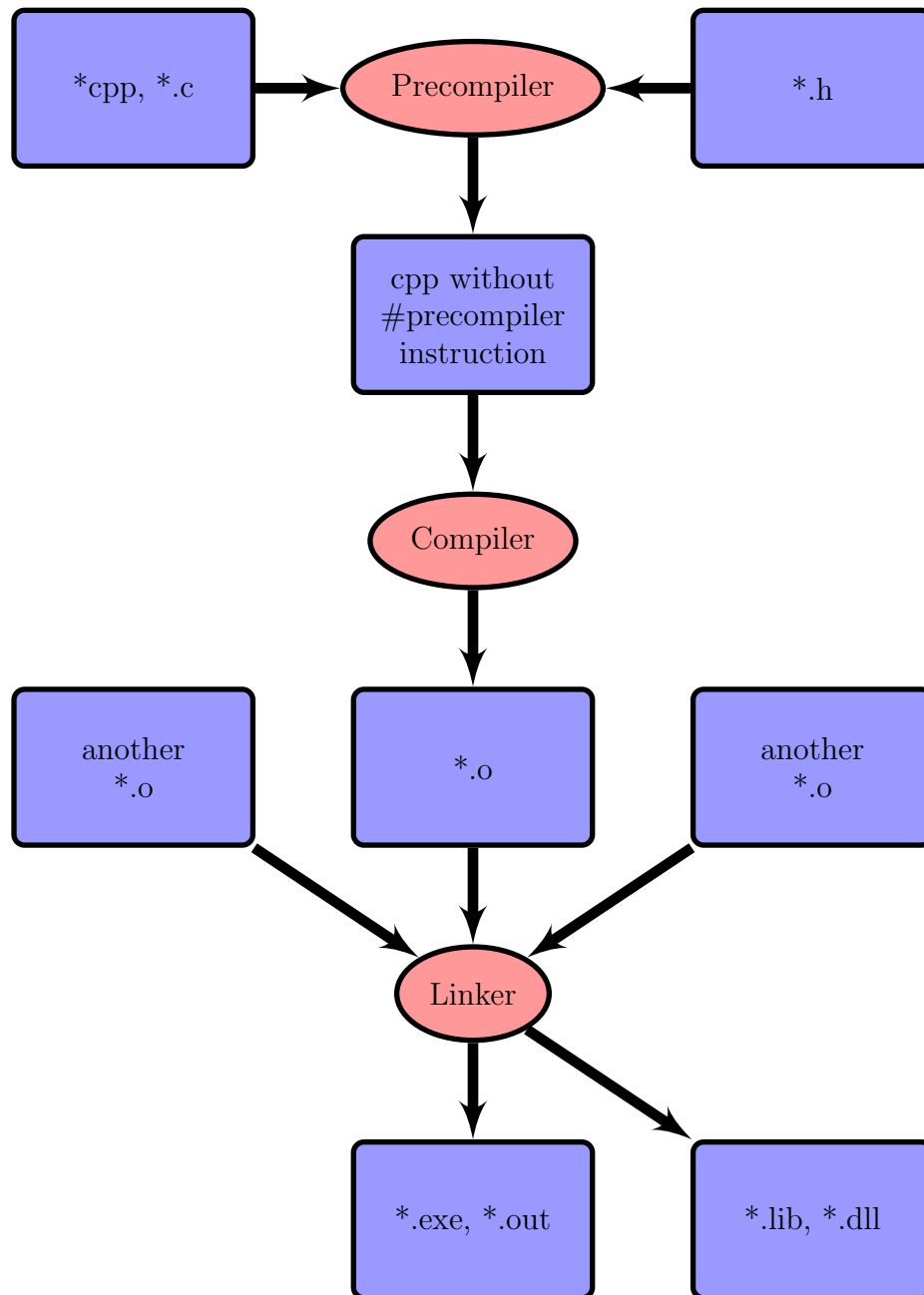
- ab 1979 von Bjarne Stroustrup bei AT&T entwickelt als Erweiterung der Programmiersprache C
- später von ISO genormt
- effizient und schnell - Schnelligkeit eines der wichtigsten Designprinzipien von C++
- hohes Abstraktionsniveau u.a. durch Unterstützung von OOP
- ISO Standard beschreibt auch eine Standardbibliothek
- C++ ist **kein** echtes Superset von C (siehe stackoverflow.com, ...)
- C++ ist (wie C) **case sensitive**
- Paradigmen:
 - **generisch** (durch Benutzung von Templates, automatische Erstellung multipler Funktionen für verschiedene Datentypen)
 - **imperativ** (Programm als Folge von Anweisungen, Gegenteil von deklarativ siehe Haskell und Logikprogrammierung)
 - **objektorientiert** (Klassen, Objekte, Vererbung, Polymorphie, Idee: Anlehnung an Realität)
 - **prozedural** (Begriff mit verschiedenen Bedeutungsauffassungen, Unterteilung des Programms in logische Teilstücke (Prozeduren), die bestimmte Aufgaben / Funktionen übernehmen)
 - **strukturiert** (prozedural und Teilung in Sequenz, Verzweigung, Wiederholung, ...)
 - **funktional** (ab C++11, Definitionenkleinkram, siehe Wikipedia, Programm als verschachtelter [...] Funktionsaufruf organisierbar, eher typisch für Haskell o.ä.)

1.1.2 Versionen

- C++98
- C++03
- C++11
 - wesentliche Neuerungen. Einführung von `constexpr`, Elementinitialisierer, ... Neue Bedeutung des Schlüsselworts `auto` `#` Referenzen ergänzen
- C++14
 - aufweichung der `constexpr` Bedingungen.
- C++17
 - soll 2017 vollendet werden.

1.2 Dateien in einem C++ Projekt

Dateiendung	Bezeichnung	Inhalt
(*.cpp) (*.cc)	Quelldatei	Funktionsimplementation, Klassenimplementation, Berechnungen bzw. eigentliche Arbeit erledigen
(*.h)	Headerdatei	Funktionsdeklaration, Klassendefinition, Bezeichner öffentlich bekannt machen
(*.o)	Objektdatei	Objektcode (Maschinencode) einer Übersetzungseinheit
(*.exe) (*.out)	ausführbare Datei	fertiges Programm
(*.sln) (*.pro) (*.vcxproj)	"Projektdatei"	IDE Einstellungen (oder ähnliches) IDE-spezifische Namen und Verwendung
(*.res)	Ressourcendatei	multimediale Inhalte



1.3 Compiler

Compiler	Plattform
GCC/g++	Windows, Linux, Mac, Unix-like
Clang	Unix-like, Mac, Windows, Linux
Intel-C++	Linux, Windows, Mac
VC++	Windows

Das nun folgende Listing zeigt, wie ein C++ Quellcode, der als Datei vorliegt, „per Hand“ mit Kommandozeile unter Nutzung des Compilers (hier g++) übersetzt werden kann. Beim Aufruf des Compilers wurden hier noch einige Flags gesetzt, nämlich `-Wall`, um **sinnvolle Warnungen** ausgeben zu lassen, und `-pedantic`, um **vom C++ Standard geforderte Warnungen** erscheinen zu lassen. Außerdem wurde der C++ Standard (Version) gesetzt.

Nutzung von g++ mittels Powershell

```
PS A:\> cd .\example\
PS A:\example> ls

Verzeichnis: A:\example


Mode                LastWriteTime         Length Name
----                -
-a----            02.04.2017    08:38             87 hello_world.cpp

PS A:\example> type .\hello_world.cpp
#include <iostream>

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

PS A:\example> g++ -o programm hello_world.cpp -Wall -pedantic -std=c++11
PS A:\example> ls

Verzeichnis: A:\example


Mode                LastWriteTime         Length Name
----                -
-a----            02.04.2017    08:38             87 hello_world.cpp
-a----            02.04.2017    09:12          48650 programm.exe

PS A:\example> .\programm.exe
Hello World
PS A:\example>
```

Eine kleine Anmerkung zu Bezeichnungen, die mit Compilern zu tun haben, möchte ich an dieser Stelle noch loswerden, da hier immer eine kleine Verwechslungsgefahr besteht. Die **GCC** (*GNU Compiler Collection*) ist eine Compilersammlung mit Compilern zu C, C++ und einigen weiteren. Dagegen ist der **gcc** (klein geschrieben) der C-Compiler der Sammlung und **g++** der C++-Compiler.

Um auf Ihrem Betriebssystem einen C++ Compiler zum Laufen zu bringen, haben Sie meist verschiedenste Möglichkeiten.

Um unter **Linux** GCC zu nutzen, müssen lediglich die entsprechenden Pakete installiert werden, meist ist die GCC sogar schon vorinstalliert.

Unter **Windows** kann man den von Microsoft bereitgestellten Visual C++ Compiler verwenden, i.d.R. in Verbindung mit einer Installation von Visual Studio (eine IDE für u.a. C++). Die zu installierenden Kompo-

nenten bei Visual Studio kann man selbst beim Installationsprozess auswählen, i.A. ist der Speicherverbrauch jedoch relativ groß. Wer auf eine speicherschonende Variante zurückgreifen will oder muss, dem empfehle ich MinGW - eine Portierung der GCC aus dem Hause Linux für Windows. Planen Sie früher oder später Qt-Creator als eine C++-IDE zu installieren, dann können Sie sich eine extra Installation von MinGW im Vorhinein sparen, da QT-Creator MinGW bereits mitinstalliert. Sofern mit der Kommandozeile direkt auf g++ zugegriffen werden soll, ist es unter Windows i.d.R. erforderlich den Pfad der MinGW-Binarys der Systemvariablen PATH hinzuzufügen.

1.4 IDEs

1.4.1 JA oder NEIN

ohne IDE	mit IDE
Compiler, Linker über Shell bedienen Texteditor evtl. make + makefile Dokumentationen	Projekteinstellungen & Buttons in IDE integriert automatisch generiertes makefile geordneter Menübaum
Einarbeitungszeit (??) für kleine und mittelgroße Projekte	Einarbeitungszeit (??) kleine, mittlere und große Projekte

1.4.2 IDEs im Überblick

IDE	Plattform	Anmerkungen
Eclipse, Netbeans Qt SDK Code::Blocks	Java (JVM) WIN, Linux, Mac WIN, Linux, Mac	in und für Java geschrieben, unterstützt auch C++ bringt umfangreiches Qt-Framework mit für GUIs u.v.m.
Visual Studio	Windows	kostenfreie BVersion für den Hausgebrauch: VS Community 2017, sehr umfangreich (Refactoring Tools, Debugger, Laufzeitanalyse, Frameworks wie MFC, ATL, WTL) und damit auch speicherintensiv, zu installierende Features wählbar, benutzt eigenen MS VC++ Compiler
Orwell DEV-C++	Windows	
Geany	Linux, WIN	schlichter Texteditor mit Syntaxhighlighting und diversen Buttons für Compilerausführung, Logausgabe
KDevelop	Linux, WIN	#
Anjuta	Linux	#
XCode	MacOS	„hauseigene“ IDE von Apple

Auf den Seiten 6 bis 9 finden sich Screenshots einiger IDEs.

1.5 Referenzen

- Buch:
 - Wolf, Jürgen: C++ - Das umfassende Handbuch. Rheinwerk Computing
- Websites:
 - <http://en.cppreference.com/w/>
 - <http://www.cplusplus.com/reference/>

Es gibt auch offline Versionen der Referenzen.

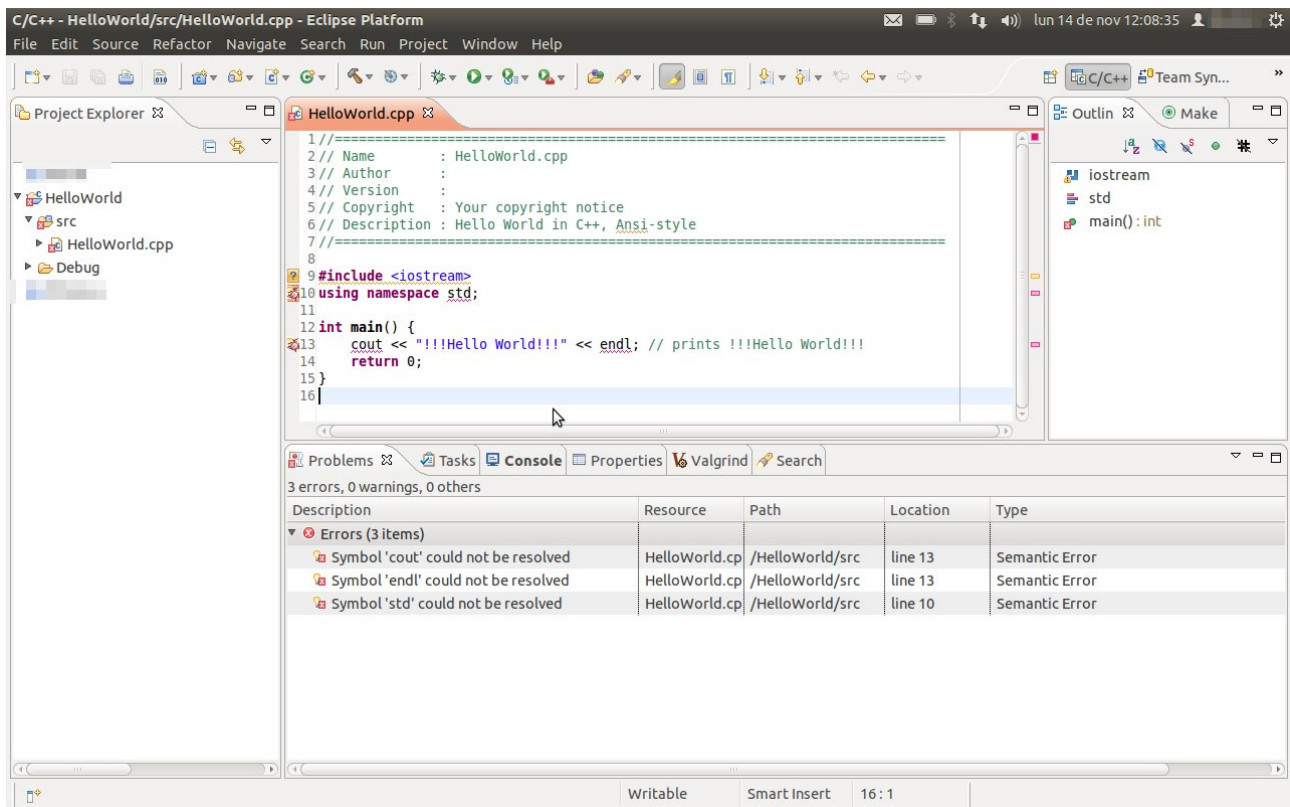


Abbildung 1.1: Eclipse mit einem C++ Projekt

<https://www.eclipse.org/forums/index.php/fa/6135/0/>

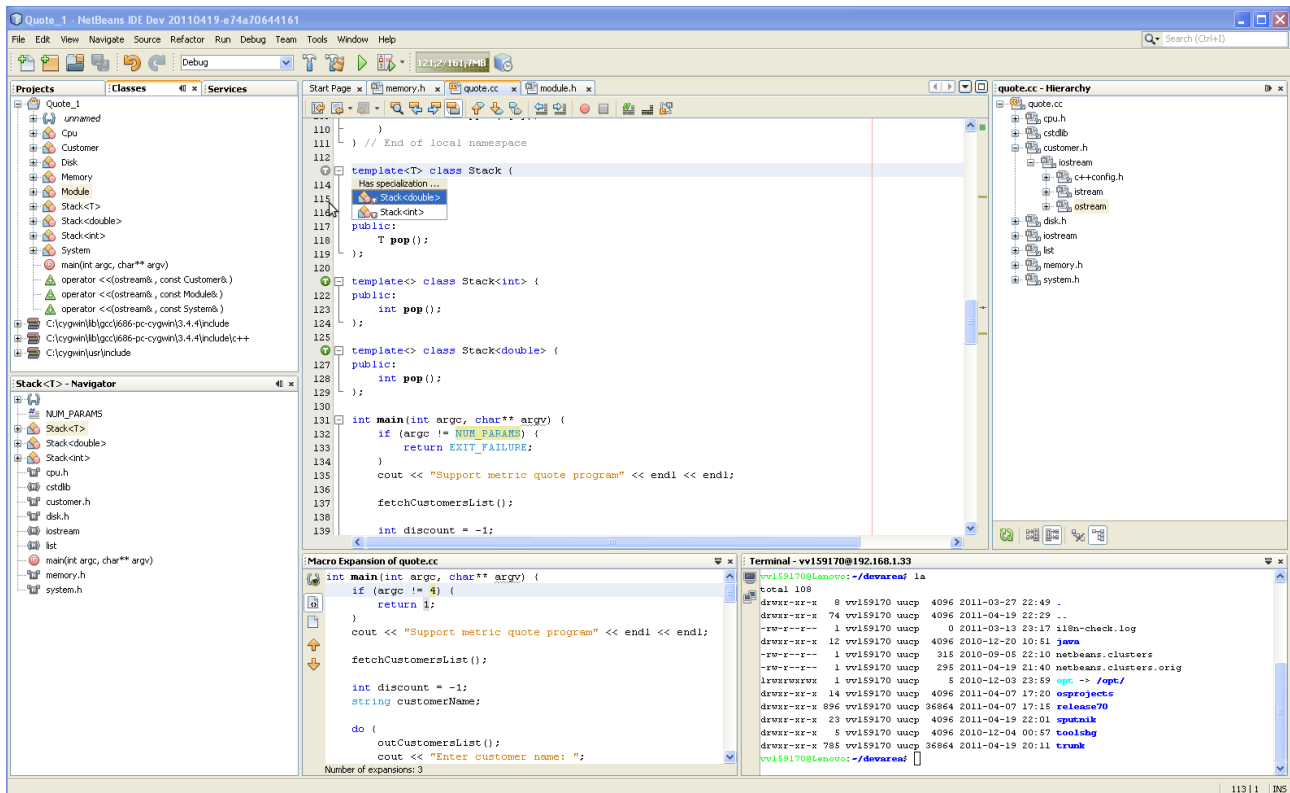


Abbildung 1.2: NetBeans und die Verwendung von C++

https://netbeans.org/images_www/v7/screenshots/cnd.png

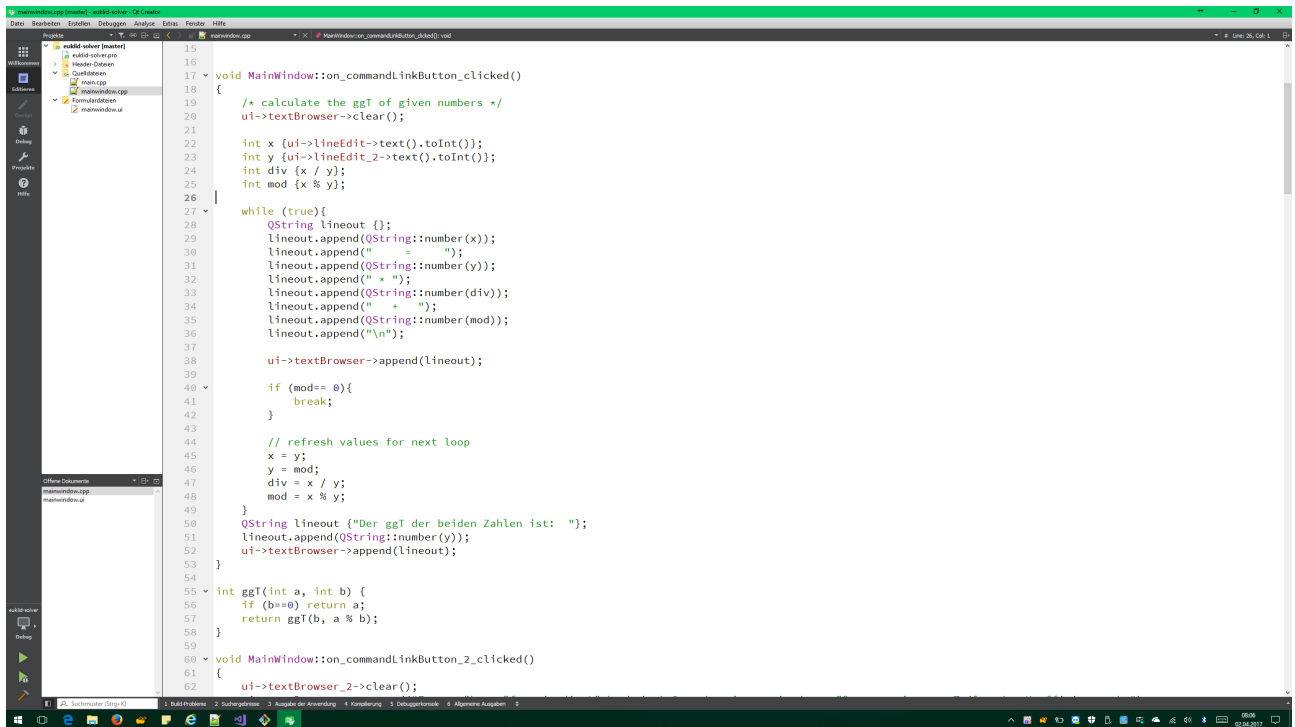


Abbildung 1.3: C++ Code in der QT Creator IDE

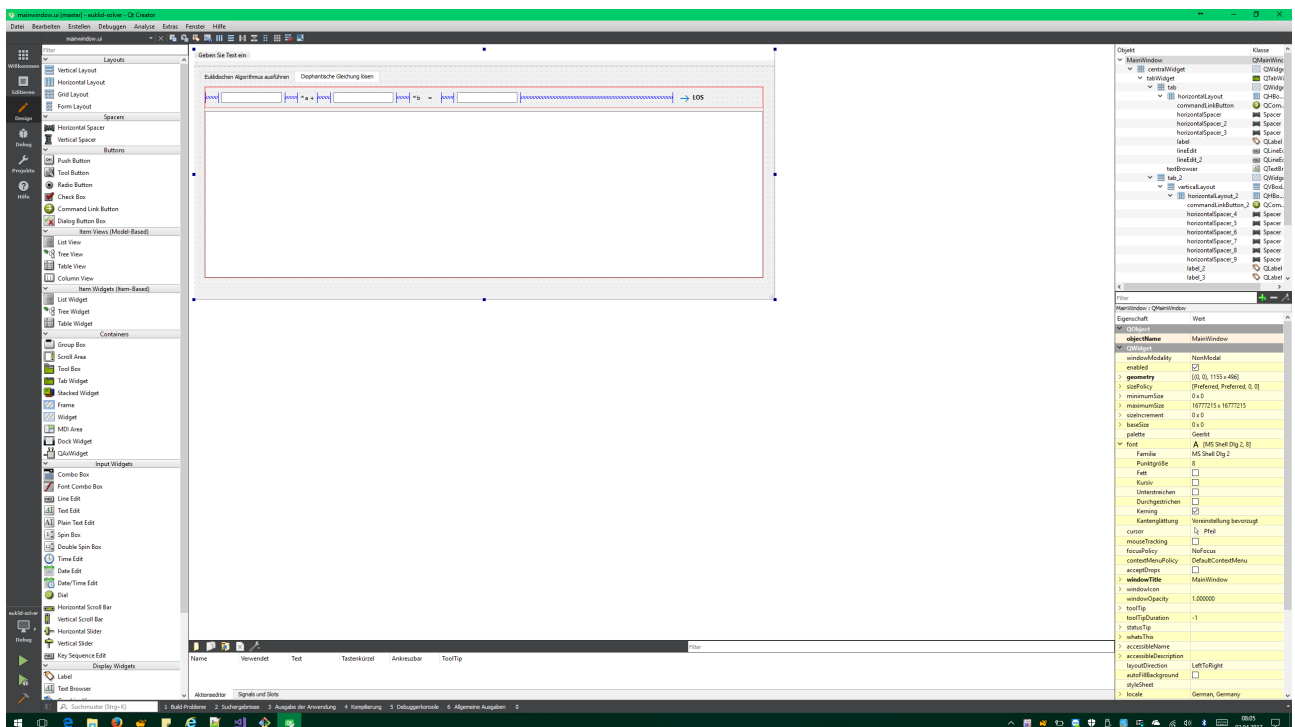
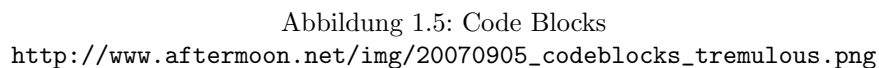


Abbildung 1.4: Fensterdesign mit QT Creator



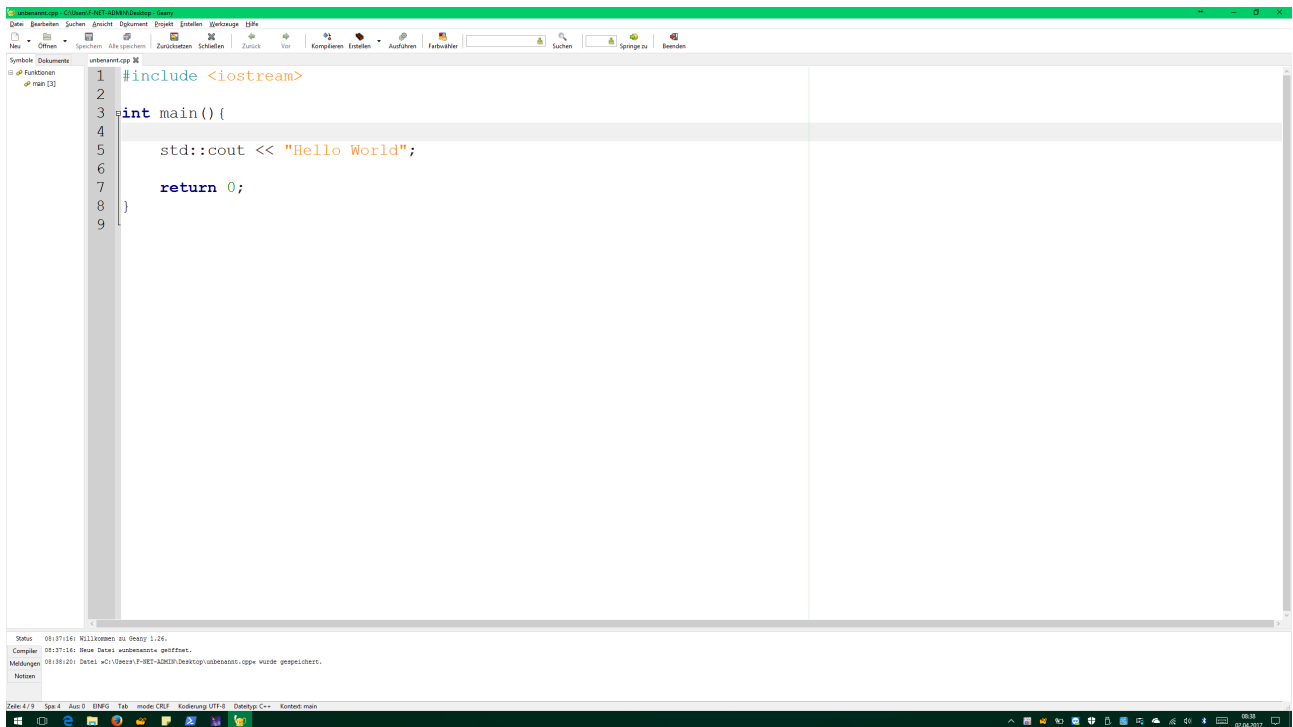


Abbildung 1.7: Geany

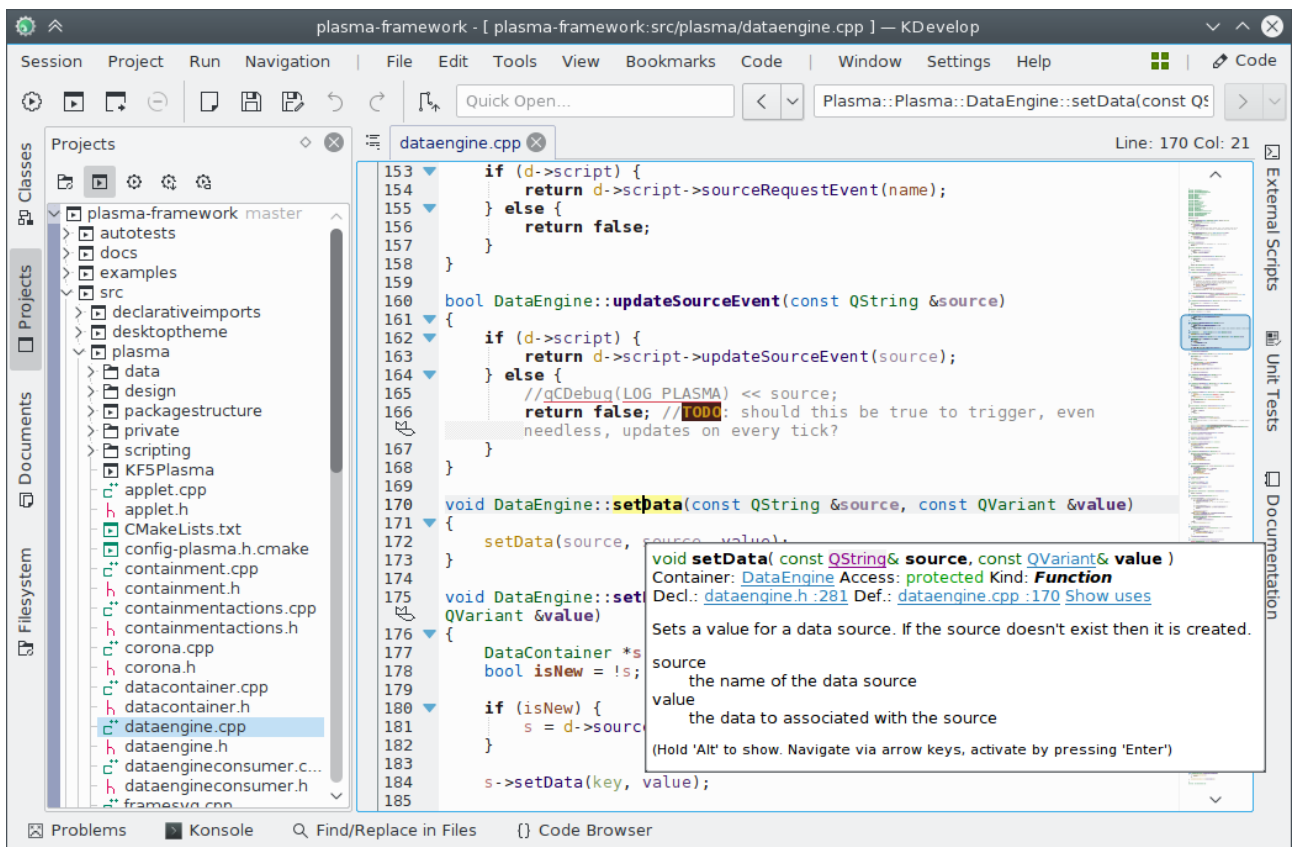


Abbildung 1.8: KDevelop

https://www.kdevelop.org/sites/www.kdevelop.org/files/inline-images/kdevelop5-breeze_2.png

1.6 The Hello World

1.6.1 Das erste kleine Programm

Unser erstes C++ Programm

```
#include <iostream>
// "Einbinden" d.h. 1-zu-1-Einfuegen des Headers iostream.h

int main(int argc, char* argv[])
// main-Funktion: Einstiegspunkt der Anwendung
// count: Anzahl der uebergebenen Parameter
// arg: Pointer auf ein Array von Pointern auf C-Style-Strings (die Parameter)
// Parameter der main-Funktion duerfen in der Signatur auch weggelassen werden.

// Parameter der main-Funktion
{ // Beginn vom Anweisungsblock der main-Funktion

    std::cout << "Hello World" << std::endl;
    // Ausgabe von "Hello World" und Zeilenumbruch
    // genauer:
    /*
    * implizite Klammerung:
    * ((std::cout) << "Hello World") << (std::endl);
    * std          ... ein Namensraum
    * ::           ... scope-Operator (Bereichsoperator)
    * cout:        ... gepufferter Standardausgabestream
    * <<           ... Ausgabeoperator (auch bitshift-Operator)
    * "Hello World" ... C-Style-String Literal
    * endl         ... Objekt aus dem std Namensraum, das einen Zeilenumbruch ('\n')
erzeugt.
    * ;           ... Abschluss einer einzelnen Anweisung
    */

    for(int i = 0; i < argc; ++i ){
        std::cout << i << ". Parameter: " << argv[i] << '\n';
    } // Beispiel fuer die Ausgabe der Komandozeilenargumente
    // argv[0] ist der Name der executable Datei

    return 0; // Rueckgabewert 0 "erfolgreich (ohne Fehler) beendet"
}
```

Im Falle der `main`-Funktion ist es auch möglich das **return statement** (`return 0;`) wegzulassen. Dann wird implizit 0 als Funktionswert zurückgegeben. Die Funktionssignatur der `main`-Funktion darf auch in `int main(int argc, char** argv)` geändert werden. Der erste Arrayeintrag von `argv` enthält übrigens immer einen Zeiger auf den Namen (ohne Dateiendung), unter dem das Programm abgespeichert wurde. Damit ist `argc` stets mindestens 1.

1.6.2 Ein paar Werkzeuge

Bevor wir in Kapitel 2 einsteigen und das gesamte (naja *fast*) C++ von Grund auf kennenlernen wollen, sollten Sie noch einige nützliche Werkzeuge kennen, damit Sie neu gelernte Dinge auch ohne große Probleme ausprobieren können.

... und ein paar Hilfsmittel ...

```
#include <iostream>

#define debug // Benutzung bedingter Compilierung zum Debugging

int main(int argc, char* argv[]){

    int zahl = 0;
    std::cout << "Wie alt bist du?\n"; // eine simple Ausgabe
    std::cin >> zahl; // eine simple Eingabe
    std::cout << "Okay!\n\n";

    #ifndef debug
        //folgende Zeile compiliert nicht:
        std::cout << << "In 7 Jahren bist du " << 7 + zahl << " Jahre alt." << '\n';
    #endif //debug

    std::cout << "Tsch" << static_cast<char>(0x81) << "ss\n";
    //https://de.wikipedia.org/wiki/Codepage_850

    std::cin.sync();
    std::cin.get(); // wartet auf Enter zum fortfahren.

    /*
    Das ist
    ein mehrzeiliger
    Kommentar
    */

    // Das ist ein einzeliger Kommentar.

}
```

Objekt	Funktionalität
cin	Standardeingabe, standardmäßig Eingabe von Tastatur
cout	(gepufferte) Standardausgabe
cerr	ungepufferte Standardfehlerausgabe
clog	gepufferte Standardfehlerausgabe
Achtung: Diese Streamobjekte liegen alle im Namensraum std und werden nach einem #include <iostream> erst verfügbar	

1.6.3 Programmierstil

Bevor es richtig losgeht, möchte ich noch ein paar Worte über den Programmierstil loswerden. Im Grunde genommen dürfen Sie Ihren C++-Code schreiben, wie sie wollen, solange Sie die Spezifikationen von c++ einhalten. Es gibt auch nicht *den einen* Programmierstil, der sich durchgesetzt hat. Sie schreiben aber einen viel leserlicheren, einfacher wartbaren und für das Auge schöneren Code, wenn Sie beim programmieren **konsistent bleiben**, was einige Aspekte betrifft:

Einrückungen	tabs or spaces
Anweisungen pro Zeile	eine, ...
Bezeichner	snake_case, camelCase, PascalCase

Einige IDEs können Sie sogar mehr oder weniger dabei unterstützen, in dem Sie sich um die **Quelltextformatierung** kümmern. Dies ist gerade bei Projekten mit vielen Entwicklern hilfreich, da so ziemlich effizient für einheitliches Quelltextlayout gesorgt werden kann.

Kapitel 2

Datentypen in C++

2.1 primitive Datentypen

Zu aller erst ist es wichtig, dass Sie mit den **eingebauten Datentypen**, auch genannt **primitive Datentypen** vertraut sind. Aus diesen setzen sich dann alle höheren Datentypen wie zum Beispiel Klassen zusammen. Auch sämtliche (oftmals relativ komplexe) Klassen aus der C++ Standardbibliothek, welche Sie zunehmend immer häufiger nutzen werden, bauen im Grunde auf nichts anderem auf.

2.2 Einige Operatoren

2.3 Casts

2.4 Zusammengesetzte Datentypen

2.4.1 Arrays

2.4.2 Records und Klassen

2.4.3 Containerklassen

2.5 Klassen

2.5.1 Konstruktoren

2.5.2 Vererbung

2.5.3 Polymorphie

Kapitel 3

Strukturierte Programmierung

3.1 Kontrollstrukturen

3.2 Funktionen

3.3 Operatoren

3.4 Modularisierung

Kapitel 4

Zusätzliche Features

4.1 Templates

4.2 Exceptions

4.3 Multithreading