Programmieren / Algorithmen & Datenstrukturen 1

Grundlagen (i), Teil 1

Prof. Dr. Skroch

Universitatea
BABEŞ-BOLYAI

Grundlagen (i)

Inhalt.

- Hallo C++
- Objekte, Typen, Werte, und Steuerungsprimitive
- Berechnungen und Anweisungen
- Fehler
- Fallstudie: Taschenrechner
- Funktionen und Programmstruktur
- Klassen

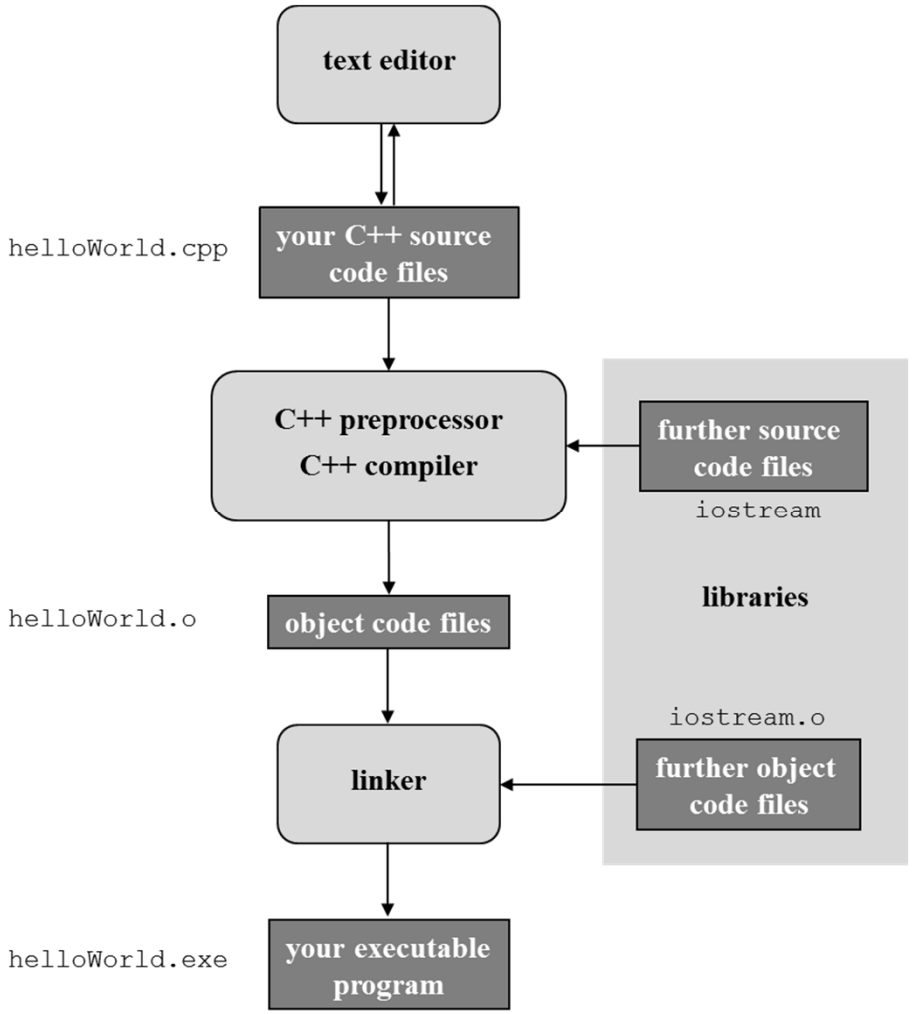
Vom Quellcode zum Programm

Eine Tool Chain vom Quellcode zum ausführbaren Programm: eingeben, übersetzen und binden.

- Editor
  - Ein- und Ausgabe, und Speicherung, des Quellcodes
  - Ergebnis: Quellcodedateien (etwa *fastTSP.cpp*, *allNP2P.h*)
- Präprozessor
  - Ergänzungen und Ersetzungen innerhalb des C++ Quellcodes (in der Realität fast immer integrierter Teil des Compilers)
  - Ergebnis: Quellcode als Input für die Übersetzung durch den Compiler
- Compiler
  - Übersetzung der präprozessierten C++ Quellcode-Datei(en) in binären Objektcode
  - Ergebnis: Objektcodedateien (etwa *fastTSP.o*, *allNP2P.o*) als Input für den Linker
- Linker
  - Zusammenführung der unterschiedlichen Objektcodes (etwa aus mehreren eigenen Objektcodedateien, dazu Objektcodedateien aus den verwendeten Bibliotheken)
  - Ergebnis: ausführbare Datei (etwa *fastTSP.exe*)

Vom Quellcode zum Programm

Eine Tool Chain vom Quellcode zum ausführbaren Programm: eingeben, übersetzen und binden.

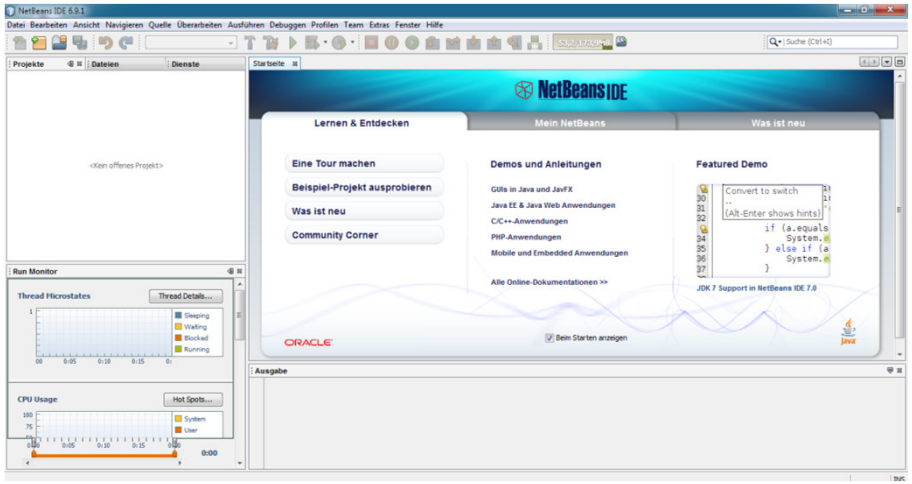


Integrierte Entwicklungsumgebungen

IDEs (Integrated Development Environments) integrieren die Komponenten einer Tool Chain.

- Häufige Bestandteile einer modernen IDE:
  - Editor, Compiler, Linker.
  - Bibliotheken, *Debugger*, Versionierung, …

- Beispiel: die NetBeans IDE.



- Dauer der Erstinstallation und -anpassung: falls keine Probleme auftreten, ca. 30 Minuten. Andernfalls kann es beliebig länger dauern…

Der kleinste C++ Programm

Was macht dieses Programm?

```
01  /*
02     main.cpp
03     v0.1 111020-OSk
04  */
05
06  int main( )                          // C++ Programme starten bei int main( )
07  {
08
09  }
10
```

Menschen, Quellcode und Compiler

Struktur und Whitespace, Kommentare.

- Grundsatz: Quellcode soll für *Menschen* gut lesbar sein.
  - Allgemein gilt, dass für Dritte schwer verständlicher Quellcode kein Zeichen für besonders gewitzte Programmierer ist, sondern oft eher für das Gegenteil…
  - Quellcode entspricht nicht nur vollständig den formalen Regeln der Programmiersprache.
  - Quellcode wird darüber hinaus – auch durch den Einsatz von sog. *Whitespace* (dazu gehören Leerzeichen, Tabulatoren, Zeilenumbrüche) – zur möglichst guten Lesbarkeit für *Menschen* strukturiert.
    - Guter Quellcode ist einheitlich und übersichtlich strukturiert.
    - Gliederungszeilen können die Lesbarkeit weiter verbessern (vgl. Zeile 05).
  - Quellcode wird in ausreichender Menge und sinnvoll kommentiert.
    - Für Einsteiger ist es ganz am Anfang noch schwer, zu beurteilen was ausreichend und sinnvoll ist… mit der Zeit werden Sie aber ein Gefühl dafür entwickeln.
- `/*  */`
  - Blockkommentar, beginnt bei `/*` und endet bei `*/`, alles zwischen den beiden Zeichen wird von C++ ignoriert.
- `//`
  - Zeilenkommentar, die Quellcodezeile wird ab `//` bis zu ihrem Ende von C++ ignoriert.

Ein C++ Programm enthält Funktionen

Jedes C++ Programm muss als eindeutigen Einstiegspunkt genau eine globale Funktion `int main()` enthalten.

- Jede Funktion besteht aus vier Teilen
  - *Name* der Funktion, hier `main`.
    - Vorsicht: in C++ wird bei allen Namen Groß- und Kleinschreibung unterschieden.
  - *Rückgabetyp*, hier `int`.
    - Der Rückgabetyp gibt an, von welchem Typ das Ergebnis der Funktion ist.
    - `int` (eine ganze Zahl, Integer) ist einer der eingebauten Typen von C++.
    - "`int`" ist auch ein von C++ reservierter Name.
  - *Parameterliste* (auch *Argumentliste* genannt), von runden Klammern `( )` eingeschlossen.
    - Die Typen der Parameter in der Reihenfolge, in der die Funktion mit ihnen aufgerufen wird.
    - Hier gibt es keine Parameter, die Liste ist demnach leer.
  - *Rumpf*, der als Block in geschweiften Klammern `{ }` eingeschlossen ist.
    - Blöcke sind eine lokale Gliederungsebene in C++.
- Das kleinstmögliche C++ Programm lautet

```
int main(){}
```

und besteht aus genau einer Funktion, die nichts* tut.

* Wenn Sie schon Vorkenntnisse in C++ haben, dann wissen Sie womöglich, dass das *nicht ganz* stimmt... es soll aber hier noch nicht vertieft werden.

Das erste Programm

Version 0.1.

```
01  /*
02     001-Variablen.cpp
03     v0.1 141217-OSk
04     Zwei Variablen definieren
05  */
06
07  #include <string>
08
09  int main( )
10  {
11    // Definition einer Variablen vom Typ int namens yob (year of birth):
12    int yob { 1997 };
13
14    // Definition einer Variablen vom Typ std::string namens nick (nickname):
15     std::string nick { "capitalQ" };
16
17    return 0;    // return Anweisung, 0 kennzeichnet erfolgreiches Ende
18  }
19
```

Das erste Programm

Der Kommentar am Anfang (Zeilen 1-5).

- Jede Programmdatei beginnt per Konvention mit einem kurzen Kommentar, der i. Allg. zumindest folgende Informationen enthält:
  - *Wer* hat die Programmdatei bearbeitet, und *wann* ?
  - *Was* macht die Programmdatei?
- Viele IDEs bieten die Möglichkeit, solche Kommentare weitgehend automatisch erstellen zu lassen.
  - Oft auch zusammen mit einer konfigurierbaren, automatischen Quellcode-Formatierung.
- Fast immer gibt es Richtlinien für die Formatierung des Quellcodes, die auch die Art der einleitenden Kommentare vorschreiben.
  - Für ein Projekt, für eine ganze Firma.
  - Wird unterschiedlich streng gehandhabt.

Das erste Programm

Die `include`-Direktive.

- `#include <string>`
- Jede Zeile im Quellcode, die mit einem # Zeichen beginnt, ist eine sog. Präprozessor-Direktive (wie z.B. die Zeile 7).
  - Der Quellcode wird an jeder dieser Zeilen verändert (vom C++ Präprozessor), bevor der geänderte Quellcode von den nachfolgenden Schritten der *C++ Tool Chain* weiter verarbeitet wird (Compiler, Linker).
- Die `#include`-Direktive lässt den Präprozessor den Inhalt einer Datei an den Punkt im Quellcode einkopieren, an dem die Direktive steht.
  - `#include <dateiname>` bzw. `#include "dateiname"` , man spricht vom Einkopieren sog. *Headerdateien*.
  - Hier wird der Quellcode einer Headerdatei namens `string` aus der *StdLib* (C++ Standardbibliothek) in Ihren Quellcode einkopiert.
    - Die StdLib stellt Ihnen im `string` Header standardisierte Möglichkeiten und Hilfsmittel für die Verarbeitung von Zeichenketten (engl. "strings") zur Verfügung.
    - Suchen Sie die Datei `string` auf Ihrem System, sehen Sie sich den Quellcode einmal an.

Das erste Programm

Anweisung (engl. *statement*).

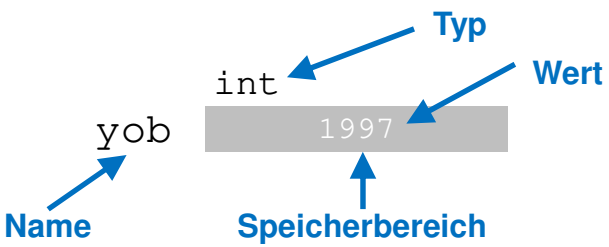
- `int yob { 1997 };`
- Diese Quellcodezeile ist eine C++ Anweisung.
- Sie können sich eine Anweisung als einen Schritt im Programmablauf vorstellen.
  - Viele C++ Anweisungen müssen mit einem Semikolon `;` beendet werden.
  - Faustregel: nach jedem Ausdruck
    - der, mit Ausnahme der Initialisierung einer Variablen, *nicht* mit einer schließenden geschweiften Klammer `}` endet
    - und der *keine* Präprozessor-Direktive (Zeile, die mit # beginnt) ist,

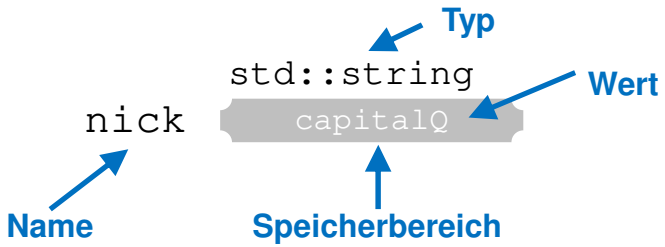
    ist ein Semikolon zu setzen.
  - Genaue Regel: leider für Einsteiger wenig verständlich, wird hier nicht vertieft.
- Schreiben Sie in Ihren Quellcode i. Allg. nicht mehr als eine Anweisung pro Zeile.

Das erste Programm

Initialisierung zweier Variablen.

- `int yob { 1997 };` `std::string nick { "capitalQ" };`





- Die beiden Anweisungen in den Zeilen 12 und 15 initialisieren Variablen, eine vom Typ `std::string` namens `nick` und eine vom Typ `int` namens `yob`.
- Diese beiden Anweisungen sind
  - *Deklarationen,* da sie *neue Namen* in das Programm einführen,
  - *Definitionen*, da sie auch *Speicher* für diese Variablen reservieren,
  - *Initialisierungen*, da sie die neuen Speicherbereiche direkt mit passenden Werten füllen: `"capitalQ"` bzw. `1997`.
- Die beiden Variablen sind innerhalb der `main()`-Funktion des Programms definiert und (nur) dort gültig.

Das erste Programm

Namen, Gültigkeit (engl. *scope*) von Namen.

- `std::string nick { "capitalQ" };`
- Alle Namen in einem C++ Programm (wie `std`, `main`, `nick`, usw.) sind in bestimmten Bereichen des Programms gültig.
  - Ggf. in allen Bereichen…
- Identische Namen dürfen in C++ Programmen nicht mehrfach mit unterschiedlichen Bedeutungen vorkommen.
  - Es könnte aber z.B. bei einem großen Programm, das von vielen Programmierern geschrieben wird, mehr als ein Programmierer auf den Namen "`string`" für jeweils unterschiedliche Dinge gekommen sein.
  - Das darf nicht zu Namenskonflikten führen, jeder Name *muss letztlich eindeutig* sein.
- Namen können daher innerhalb eines, wiederum benannten, *Namensraums* (`namespace`) deklariert werden.
  - Mit Angabe ihres Namensraums sind Namen dann immer eindeutig, denn innerhalb eines `namespace` darf es jeden Namen nur einmal geben.
  - Mit der Syntax in Zeile 15 ist `string` aus dem `namespace` namens `std` gemeint.

Das erste Programm

Auflösung des Gültigkeitsbereichs von Namen im Programm durch den Operator :: (engl. *scope resolution*).

- `std::string nick { "capitalQ" };`
- Gleichnamige Bezeichner können durch ihre Deklaration in unterschiedlich benannten Namensräumen eindeutig gemacht und dann konfliktfrei in C++ verwendet werden.
  - Das `namespace` Konzept ähnelt dem Konzept der Vor- und Familiennamen bei Menschen: Tom Petty und Tom Waits sind unterschiedliche Namen.
- Um den Gültigkeitsbereich bzw. Scope von `string` anzugeben (den `namespace std`) wird der Operator `::` (sog. Gültigkeitsbereichsauflösungsoperator) verwendet.
- Im Namensraum `std` befindet sich vollständig die gesamte C++ Standardbibliothek (StdLib).
  - `string` aus der StdLib liegt daher auch im Namensraum `std`.
- Die `namespace`-Namen selbst sind eindeutig.
  - Es gibt nur einen `namespace` namens `std`.

Gültigkeitsbereiche

Gültigkeitsbereiche in C++ helfen dabei, Namen möglichst lokal zu halten und Namenskonflikte für mehrmals vergebene Namen aufzulösen.

- Globaler Bereich
  - Außerhalb jedes anderen Scopes – überall gültig.
- Benannter Namensraum
  - Gültig innerhalb eines benannten Bereichs (`namespace`).
  - Namensräume dienen dazu, Konflikte zu beheben, wenn gleiche Namen mehrfach vorkommen und unterschiedlich verwendet werden (z.B. Namen aus unterschiedlichen Bibliotheken).
- Klassenbereich
  - Gültigkeit innerhalb einer sog. Klasse.
- Lokaler Bereich
  - Vereinfacht ausgedrückt: gültig zwischen den geschweiften Klammern `{ }`, innerhalb derer der Name deklariert wurde.
- Anweisungsbereich
  - Gültig innerhalb einer (einzigen) Anweisung.

Gültigkeitsbereiche

Gültigkeitsbereiche in C++ helfen dabei, Namen möglichst lokal zu halten und Namenskonflikte für mehrmals vergebene Namen aufzulösen.

- Beispiel-Quellcode

```
01 int aNumber { 42 };              // diese Variable namens ::aNumber
02                                  // liegt im globalen Scope (der keinen Namen hat)
03
04 namespace pad1 {                // diese Variable namens pad1::aNumber
05   int aNumber { 27 };            // liegt im namespace namens pad1
06 }
07
08 int main( )
09 {
10   // die folgende Variable namens aNumber hat lokalen Scope in main():
11   int aNumber { pad1::aNumber - ::aNumber };
12
13   // Wir starten den Debugger und sehen uns an, welchen Wert aNumber enthaelt
14
15   return 0;
16 }
17
```

Auflösung des Gültigkeitsbereichs

Möglichkeiten der Scope-Auflösung für Namen.

- Der gewünschte Namensraum kann auf drei Arten angegeben werden.
  - **Explizit ("vollqualifizierter Name")**

```
std::string nick1 { "Sokrates" };
std::string nick2 { "Plato" };
std::string nick3 { "Aristoteles" };
// Man muss im Quellcode jedesmal std:: vor string
// schreiben, das kann laestig werden.
```

  - `using`-Direktive

```
using namespace std;
// Jetzt ist der direkte Zugriff auf ALLE Namen
// im Namensbereich std (C++ StdLib) moeglich.
string nick4 { "Kant" };
```

    - Verwenden Sie allgemein möglichst *keine* `using`-Direktiven.
    - In dieser Lehrveranstaltung ist `using namespace std` für die C++ Standardbibliothek (StdLib) die einzige erlaubte `using`-Direktive.

  - **`using`-Deklaration**
    - Siehe nächste Seite.

Das erste Programm

Version 0.2.

- `using`-Deklaration

```
using std::string;
// Der Name string bedeutet ab jetzt immer std::string,
// eine nette Abkuerzung fuer den Quellcode.
string nick5 { "Foucault" };
```

- Mit einer `using`-Deklaration sieht das voriges Programm etwa so aus:

```
01  #include <string>
02  using std::string;  // using Deklaration: string aus der StdLib
03
04  int main( )
05  {
06    int yob { 1997 };
07    string nick { "capitalQ" }; // ok wegen Zeile 2
08    return 0;
09  }
10
```

Das erste Programm

Rückgabewert.

- `return 0;`
- Mit der `return`-Anweisung gibt die `main()` Funktion den Wert `0` (null) als ganze Zahl (C++ Typ `int`) zurück.
  - Ähnlich wie Variablen haben C++ Funktionen i.Allg. einen Wert (den Rückgabewert).
  - Nach dem Ende einer Funktion tritt der Rückgabewert der Funktion an ihre Stelle, und kann von der aufrufenden Funktion ausgewertet und ggf. weiter verarbeitet werden.
  - `main()` wird nicht von einer anderen Funktion sondern vom Betriebssystem aus aufgerufen, und kann auch von dort abgefragt werden.
    - Etwa indem das Programm durch Shellskript / Batchdatei gestartet wird und der Rückgabewert in einer Variablen des Skripts gespeichert wird.
- Mit einem Rückgabewert `==0` (gleich null) möchte man üblicherweise anzeigen, dass die Funktion zur Laufzeit erfolgreich beendet wurde.
- Rückgabewerte `!=0` (ungleich null) sollen üblicherweise besondere Vorkommnisse zur Laufzeit anzeigen.

Hallo C++, das klassische erste C++ Programm

Hallo C++ auf dem Bildschirm ausgeben.

```
01  /*
02     001-Hallo.cpp
03     110821-OSk
04     Gibt Hallo C++! auf der Konsole aus
05  */
06
07  #include <iostream>
08  using std::cout;
09
10   int main( )
11  {
12     cout << "Hallo C++!\n";
12     return 0;
13  }
14
```

Hallo C++

Zeichenketten, Zeichen, und ihre Literale.

- `"Hallo C++!\n" // Zeichenketten-Literal`
- Literale im Quellcode sind Werte, die für sich selbst stehen und nicht (wie Namen) etwas anderes bezeichnen.
- Zeichenkette
  - Eine Zeichenkette ist eine Folge von einzelnen Zeichen.
  - Literale für Zeichenketten stehen im C++ Quellcode zwischen doppelten oberen Anführungszeichen `"` (ASCII-34, „Smart Quotes“ gehen nicht).
  - C++ Typ aus der StdLib zum Umgang mit Zeichenketten: `std::string`.
- Einzelnes Zeichen, wie z.B. `'a'` oder `'@'`
  - Literale für einzelne Zeichen steht im C++ Quellcode zwischen einfachen oberen Anführungszeichen `'` (ASCII-39, ‚Smart Quotes‘ gehen nicht).
  - Eingebauter C++ Typ zum Umgang mit einzelnen Zeichen: `char`.
- Informieren Sie sich selbstständig über die in C++ Quellcode erlaubten Zeichen (Bemerkung: deutsche Umlaute wie ö sind z.B. nicht erlaubt).

Hallo C++

Exkurs: Zeichencodierung.

- Eine Zeichencodierung ist eine Abbildung von Schriftzeichen auf Werte, die im Computer darstellbar sind.
  - Schriftzeichen werden dazu praktischerweise auf ganze Zahlen abgebildet.
- Schon lang bestehend und sehr verbreitet ist *ASCII* ("American Standard Code for Information Interchange").
  - Bildet die im US-Englisch vorkommenden Zeichen sowie einige Satz- und Sonderzeichen auf den Zahlenraum von 0 bis 127 ab.
  - Kann somit in 7 Bit dargestellt werden, $2^7 = 128$.
- Eine weitere Abbildung ist *ISO 8859-1* (benötigt 8 Bit).
  - Im Bereich von 0 bis 127 mit ASCII identisch.
  - Von 128 bis 255 sind weitere in westeuropäischen Sprachen vorkommende Zeichen und Sonderzeichen abgebildet (z.B. deutsche Umlaute).
- Eine weitere Abbildung ist *Unicode* (benötigt 16 Bit).
  - Im Bereich von 0 bis 255 mit ISO 8859-1 identisch.
  - Kann bis zu 65536 Schriftzeichen unterscheiden.

Hallo C++

Exkurs: Sonderzeichen und Escape-Sequenzen.

- `"Hallo C++!\n"`
- Für Sonderzeichen, die man nicht einfach so in den Quellcode eingeben kann, können sog. "Escape-Sequenzen" verwendet werden, z.B.:

Zeichen	*Escape-Sequenz*	*Hexadezimal ASCII*
Backslash	`\\`	`\x5C`
Single quotation mark	`\'`	`\x27`
Double quotation mark	`\"`	`\x22`
Question mark	`\?`	`\x3F`
Alarm	`\a`	`\x07`
Backspace	`\b`	`\x08`
Form feed	`\f`	`\x0C`
Line feed	`\n`	`\x0A`
Carriage Return	`\r`	`\x0D`
Tabulator	`\t`	`\x09`
Vertical tabulator	`\v`	`\x0B`

Ein (einziges) `char`

Ein (einziges) `char`

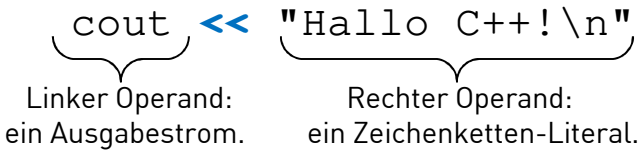
Hallo C++

Ausgabe mittels des binären Stromausgabe-Operators der C++ StdLib (*binär* bedeutet hier, dass der Operator *zwei Operanden* hat).

- `std::cout << "Hallo C++!\n";`
- Der Stromausgabe-Operator `<<` der StdLib und der dazu gehörende Standard-Ausgabestrom `cout` werden verwendet.
  - StdLib kennt vier Standard E/A-Ströme:

`cin`	Zeichen**eingabe**	i.d.R. von der Tastatur,
`cout`	Zeichenausgabe	i.d.R. auf den Bildschirm,
`clog`	Zeichenausgabe	i.d.R. auf den Bildschirm, für Kontrollmeldungen,
`cerr`	Zeichenausgabe	i.d.R. auf den Bildschirm, für Fehlermeldungen.

  - Der StdLib Stromausgabe-Operator <<
    - gibt seinen rechten Operanden auf den Ausgabestrom (linker Operand) aus.



    - Unterscheidet den *Typ* seines rechten Operanden, z.B. `bool`, `char`, `int`, `double`, `std::string`.
    - Die Ausgabe erfolgt typgerecht formatiert.

Funktionen und Rückgabewerte

Ausführbarer C++ Code ist in Funktionen enthalten und wird direkt oder indirekt aus `main()` aufgerufen.

```
01  #include <iostream>
02  using std::cout;
03
04  double square( double x ) { // das Quadrat einer Gleitkommazahl berechnen
05    return x * x;
06  }
07
08  void print_square( double x ) {
09    cout << "The square of " << x << " is " << square( x ) << '\n';
10  }
11
12  int main( ) {
13     print_square( 4.321 ); // 18.671
14     return 0;
15  }
16
```

Einige Beispielfragen

Das erste Programm.

- Was ist eine Tool Chain?
- Was macht der Compiler?
- Welche Funktion hat der Linker?
- Was ist der Unterschied zwischen Quellcode und Objektcode?
- Wo ist der Quellcode, der die Bibliotheksfunktionen implementiert?
- Was ist eine IDE, und warum sind IDEs so nützlich?
- Was macht das erste Programm?
- Wozu sind `#include`-Direktiven gut?
- Nennen Sie eine Funktion, die in jedem C++ Programm vorhanden sein muss.
- Können unterschiedliche Funktionen gleiche Namen haben?
- Welche vier Teile hat eine Funktion?

Einige Beispielfragen

Das erste Programm.

- Was ist eine Anweisung?
- Was gibt eine Funktion zurück?
- Nennen Sie die Gültigkeitsbereiche für Namen in C++.
- Was ist eine `using`-Deklaration? Warum setzt man sie ein?
- Welche I/O-Streams der C++ Standardbibliothek kennen Sie?
- Was ist eine Zeichenkette? Welcher Typ aus der C++ Standardbibliothek dient zum Umgang mit Zeichenketten?
- Welcher C++ Typ dient zum Umgang mit einzelnen Zeichen?
- Wozu dient die Zeile `return 0;` in den Programmen?
- Was ist ein Literal? Wie unterscheiden sich Literale von Namen?
- Erklären Sie die Funktionsaufrufe und Rückgabewerte bei der Ausgabe des Quadrats einer Gleitkommazahl im Beispielcode.
- Warum muss man, auch wenn man alles verstanden hat, trotzdem üben?

Einige Beispielfragen

Finden Sie die Fehler?

```
01  int Main( )
02  {
03    std::cout << 'Hallo C++!\n';
04    Return 0;
05  }
```

```
01  inc1ude <iosteam>
02  int main
03  {
04    std:string nickname { "Leibniz" };
05    return o;
06  }
```

```
01  #inculde <string>
02  using namespace std
03  int main( )
04  {
05    std::cout >> "Hallo C++!\n;
06    return 0;
07  )
```

Nächste Einheit:
Objekte, Typen, Werte, und Steuerungsprimitive