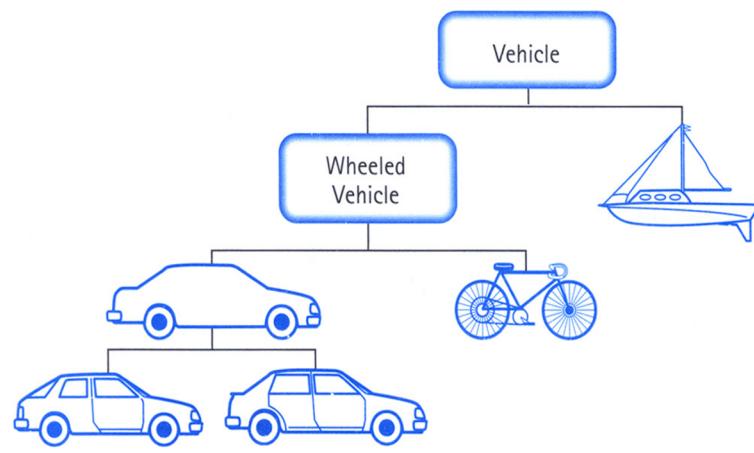


Objekt-Orientierte Programmierung

VORLESUNG 1





Dominik Knoll

Geboren in Oberösterreich

Schule, Handwerksausbildung und Studium in Salzburg

2007 Magister der Angewandten Informatik

“Reinforcement Learning in Neurocontrollers of Simulated and Real Robots”

Arbeit als Informatiker in Italien (2 j.), Schweiz (1 j.),
Russland (3 j.)

2020 Doktorat in Engineering Systems

“Model-based processes and tools for concurrent conceptual design of space systems”

www.linkedin.com/in/djknoll/

djknull.github.io



Wer sind “Echte”-
Programmierer?

Was machen sie?

Programmer



What my friends think I do



What my mom thinks I do



What society thinks I do



What my boss thinks I do



What I think I do



What I actually do

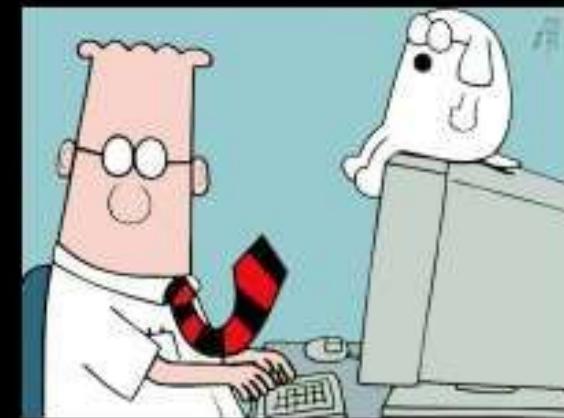
Computer Programmer



what my mom think I do



what my friends think
I do



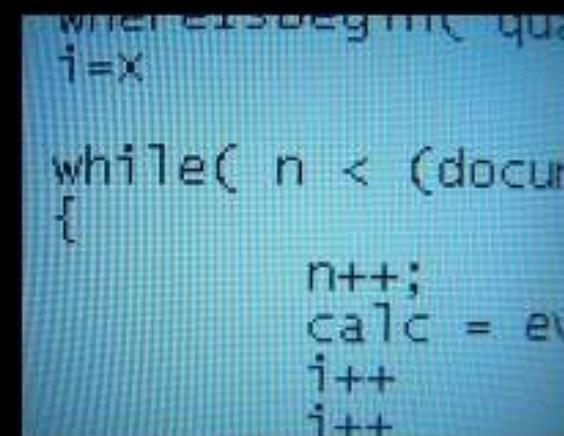
what society thinks I
do



what my boss thinks I
do



what I think I do



what I really do

COMPUTER PROGRAMMING



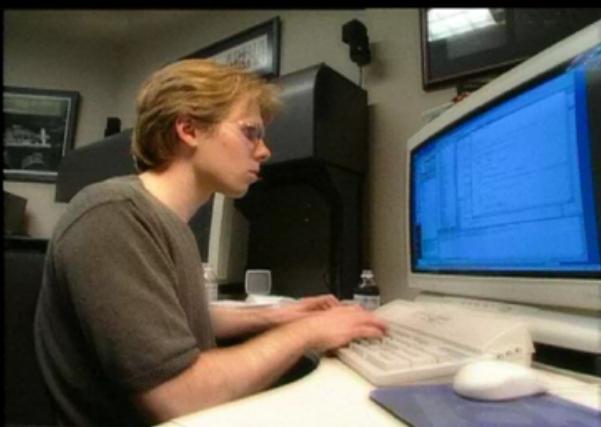
**What my Family
Thinks I Do**



**What my Friends
Think I Do**



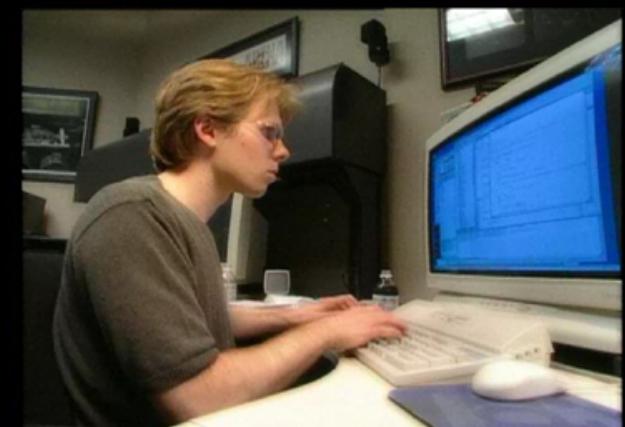
**What Society
Thinks I Do**



**What Hard Sciences
Majors Think I Do**



What I Think I Do



What I Actually Do





Warum C++ ?

- Programme können auf verschiedenen (allen) Betriebssystem kompiliert und ausgeführt werden
- Erlaubt die Hardware effizient zu nutzen
- Industrie-Standard: etabliert, zuverlässig, breit unterstützt



wichtige Information über die Lehrveranstaltung



Profs: Dominik Knoll, Iulian Bența

Arbeitsumfang (in Stunden)

Vorlesung: 2

Seminar/Labor: 1 + 2

Email: dominik.johannes.knoll@scsubbcluj.ro



Ziele

- Der Unterschied zwischen der traditionellen Programmierung und der objektorientierter Programmierung
- Verstehen der Klassen als Grundstrukturen der Programmierung
- OOP Säulen und Prinzipien
- Benutzerschnittstellen in C++
- Entwurfsmuster



Kursinhalt

1. Hello C++
2. Objektorientierte Paradigma
3. Objektorientierte Programmierung in C++
4. Generische Programmierung
5. Ereignisgesteuerte Programmierung



Prüfungsform

- **Klausur, schriftlich (30%)**
- **Praktische Prüfung, mündlich (30%)**
- **Laboraufgaben, je 2 Wochen (40%)**

Minimale Leistungsstandards

K, L, P >= 5 (inkl. aller Labors)



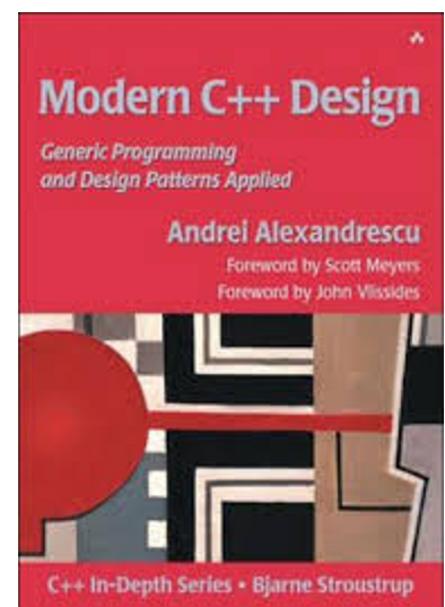
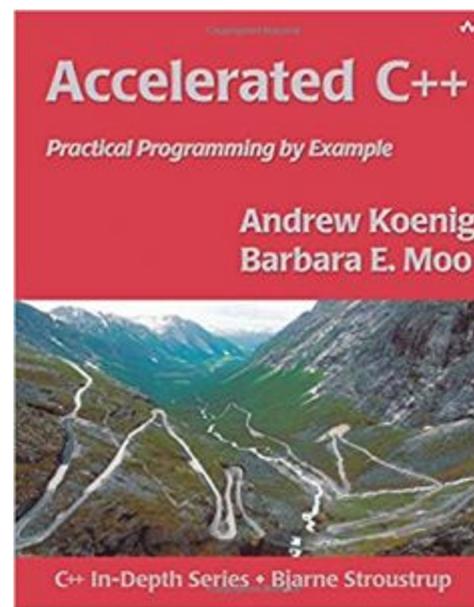
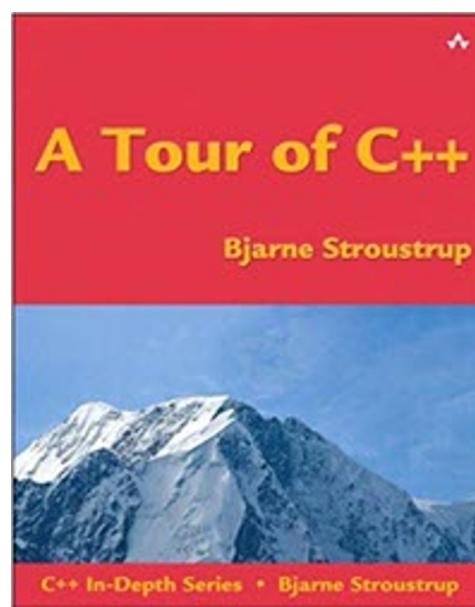
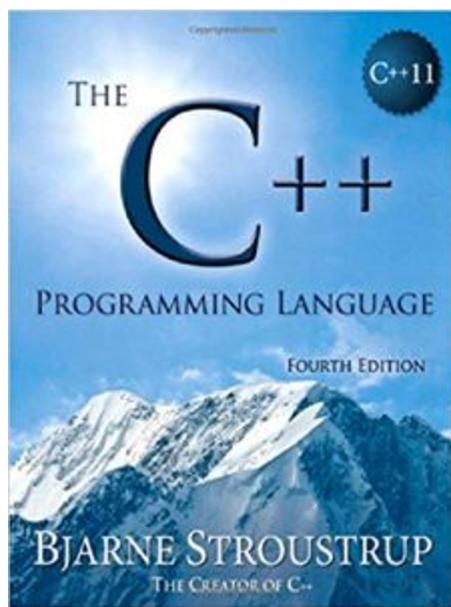
Anwesenheit

- **Kurs: optional**
- **Seminar: 5/7**
- **Labor: 12/14**

los_gehts (C++)



Literatur





Vom Quellcode zum Programm 1

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

1. Editor

- Ein- und Ausgabe, und Speicherung, des Quellcodes
- Ergebnis: Quellcodedateien (etwa myProg.cpp, library.h)

2. Präprozessor (oft integrierter Teil des Compilers)

- Ergänzungen und Ersetzungen innerhalb des C++ Quellcodes
- Ergebnis: Quellcode als Input für die Übersetzung durch den Compiler



Vom Quellcode zum Programm 2

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

3. Compiler

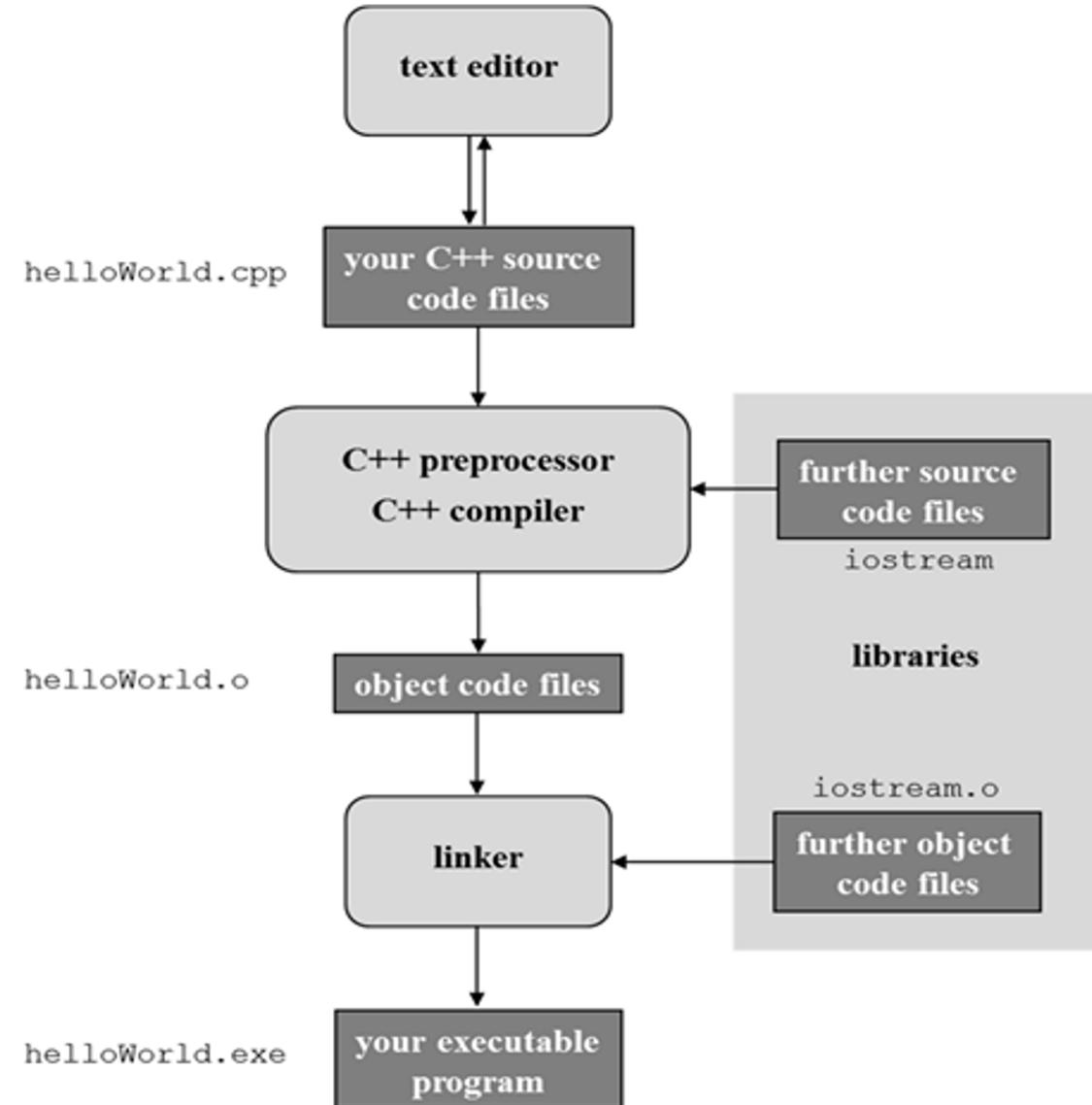
- Übersetzung der präprozessierten C++ Quellcode-Datei(en) in binären Objektcode
- Ergebnis: Objektcodedateien (etwa myProg.o, library.o) als Input für den Linker

4. Linker

- Zusammenführung der unterschiedlichen Objektcodes (etwa aus mehreren Objektcodedateien, insbesondere aus den verwendeten Bibliotheken)
- Ergebnis: ausführbare Datei (etwa myProg.exe)

Vom Quellcode zum Programm

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





Integrierte Entwicklungsumgebungen

IDEs (Integrated Development Environments) integrieren die Komponenten unter einer Oberfläche.

Häufige Bestandteile einer modernen IDE:

- Editor, Compiler, Linker,
- Debugger,
- Bibliotheken,
- Code Versionierung, ...

*Bereite deine
Arbeitsumgebung
vor!*

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders: test1.cpp, date.cpp, CPP-TRIAL, .vscode, date.dSYM, komplex.dSYM, test1.dSYM, date, date.cpp, komplex, komplex.cpp, test1, and test1.cpp. The Editor pane on the right displays the code for date.cpp:

```
date.cpp — cpp-trial
test1.cpp date.cpp
date.cpp > ...
1 #include <iostream>
2
3 struct date
4 {
5     int day;
6     int month;
7     int year;
8 };
9
10 int main()
11 {
12     date d;
13
14     d.day = 21;
15     d.month = 02;
16     d.year = 2020;
17
18     std::cout << d.day << "." << d.month << "." << d.year;
19
20 }
```

The Terminal pane at the bottom shows the command line: d.knoll@dk-mb-15-2019 cpp-trial %



Das kleinste C++ 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
```

High School / Junior High

```
10 PRINT "HELLO WORLD"
20 END
```

First Year in College

```
program Hello(input, output)
begin
  writeln('Hello World')
end.
```

Senior Year in College

```
(defun hello
  (print
    (cons 'Hello (list 'World))))
```

New professional

```
#include
void main(void)
{
    char *message[] = {"Hello ", "World"};
    int i;

    for(i = 0; i < 2; ++i)
        printf("%s", message[i]);
    printf("\n");
}
```

Seasoned professional

```
#include
#include
class string
{
private:
    int size;
    char *ptr;
public:
    string() : size(0), ptr(new char['\0']) {}
    string(const string &) : size(s.size())
    {
        ptr = new char[size];
        strcpy(ptr, s.ptr);
    }
    ~string()
    {
        delete [] ptr;
    }
    friend ostream &operator <<(ostream &, const string &);
    string &operator=(const char *);
};
ostream &operator<<(ostream &stream, const string &s)
{
    return(stream << s.ptr);
}
string &string::operator=(const char *chrs)
{
    if (this != &chrs)
    {
        delete [] ptr;
        size = strlen(chrs);
        ptr = new char[size + 1];
        strcpy(ptr, chrs);
    }
    return(*this);
}
int main()
{
    string str;
    str = "Hello World";
    cout << str << endl;
    return(0);
}
```

false

Apprentice Hacker

```
#!/usr/local/bin/perl
$msg="Hello, world.\n";
if ($#ARGV >= 0) {
    while(defined($arg=shift(@ARGV))) {
        $outf = $arg;
        open(FILE, ">" . $outf) || die "Can't write $arg: $!\n";
        print (FILE $msg);
        close(FILE) || die "Can't close $arg: $!\n";
    }
}
else {
    print ($msg);
}
1;
```

Experienced Hacker

```
#include
#define S "Hello, World\n"
main(){exit(sprintf(S) == strlen(S) ? 0 : 1);}
```

Seasoned Hacker

```
* cc -o a.out ~/src/misc/hw/hw.c
* a.out
```

Guru Hacker

```
* cat
Hello, world.
^D
```

Junior Manager

```
10 PRINT "HELLO WORLD"
```

```
Middle Manager
mail -s "Hello, world." bob@bob.com
could you send me a "Hello, world." program that prints
Hello, world. tomorrow.
^D
Senior Manager
mail -s "Hello, world." bob@bob.com
I am sending you a "Hello, world." program by this afternoon.
```

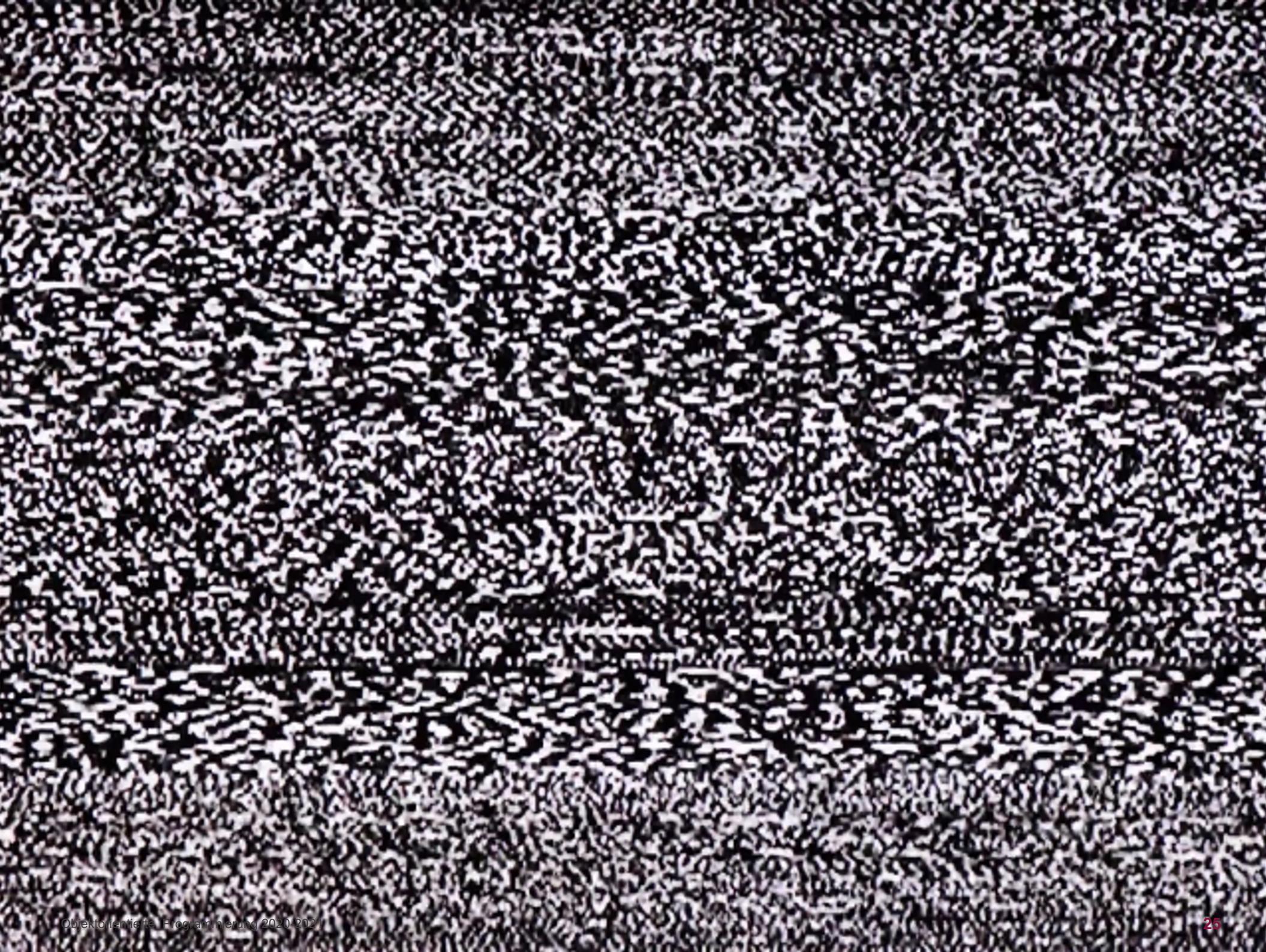
Chief Executive

```
* letter
letter: Command not found.
* mail
To: ^X ^F ^C
* help mail
help: Command not found.
* damn!
!: Event unrecognized
* logout
```



Menschen, Quellcode und Compiler

- **Grundsatz:** Quellcode soll für Menschen gut lesbar sein.
 - *Es reicht nicht dass Quellcode den formalen Regeln der Programmiersprache entspricht!*
 - Guter Quellcode ist einheitlich und übersichtlich strukturiert.
 - **Kommentare** können helfen die Intention des Programmierers darzulegen.
 - Wieviel davon sinnvoll und notwendig ist ... dazu braucht es etwas Erfahrung.
- /* */
- Blockkommentar, beginnt bei /* und endet bei */, alles zwischen den beiden Zeichen wird vom Compiler ignoriert.
- //
- Zeilenkommentar, die Quellcodezeile wird ab // bis zu ihrem Ende vom Compiler ignoriert.



code-to-noise
ratio



Negativ Beispiel

```
49
50     // Get the "Cancel" event link and store it in &cancelDescription
51     FMOD::Studio::EventDescription* cancelDescription = NULL;
52     ERRCHECK( system->getEvent("event:/UI/Okay", &cancelDescription) );
53
54     do // start of while (!Common_BtnPress(BTN_QUIT)) loop
55     {
56         // Check for button input
57         Common_Update();
58
59         // Check if button 1 was pressed
60         if (Common_BtnPress(BTN_ACTION1))
61         {
62             // Create an instance for triggering the Cancel event
63             FMOD::Studio::EventInstance* eventInstance = NULL;
64             ERRCHECK( cancelDescription->createInstance(&eventInstance) );
65
66             // Start playing the Cancel event from the instance
67             ERRCHECK( eventInstance->start() );
68
69             // Release will clean up the instance when it completes
70             ERRCHECK( eventInstance->release() );
71         }
72
73         // Update the audio system - needs to be done each game frame (usual
```



Bemerke den Unterschied

```
// diese Funktion extrahiert das grösste Element
// aus dem Array ams mit der Länge l
int whatever(int ams, int l) {
    // z ist das aktuelle maximum
    int z=INT_MIN;
    for(int i=0; i<l; i++) {
        if(ams[i]>z) z = ams[i];
    }
    return z;
}
-----int extractMaximum(int *array, int length) {
    int maximum=INT_MIN;
    for(int index=0; index<length; index++) {
        if(array[index]>maximum) maximum = array[index];
    }
    return maximum;
}
```

call things by
their name!

the name reflects
the purpose



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**

1. Name der Funktion, hier `main`.
 - Vorsicht: in C++ wird bei allen Namen Groß- und Kleinschreibung unterschieden.
2. 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.



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

3. 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.
4. Rumpf, der als Block in geschweiften Klammern {} eingeschlossen ist.
 - Blöcke sind eine lokale Gliederungsebene (Gültigkeitbereich) in C++.



Das erste Programm

```
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  
18 } Ende  
19
```



Der Kommentar am Anfang

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



Die include-Direktive.

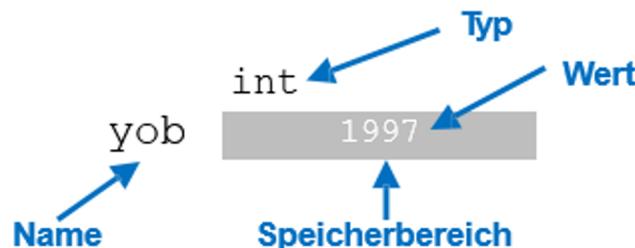
- `#include <string>`
- Jede Zeile im Quellcode, die mit einem # Zeichen beginnt, ist eine sog. Präprozessor-Direktive (wie z.B. die Zeile 07).
 - Der Quellcode wird an jeder Zeilen verändert (vom Präprozessor), bevor er (vom Compiler) verarbeitet wird.
- Die `#include`-Direktive weist den Präprozessor an, den Inhalt einer Datei an dem Punkt im Quellcode einzufügen, an dem die Direktive steht.
 - `#include <dateiname>` bzw. `#include "dateiname"` , man spricht vom Einfügen der sog. Headerdateien.
 - In Zeile 07 wird der Quellcode einer Headerdatei namens `string` aus der C++ Standardbibliothek in den Quellcode eingefügt.



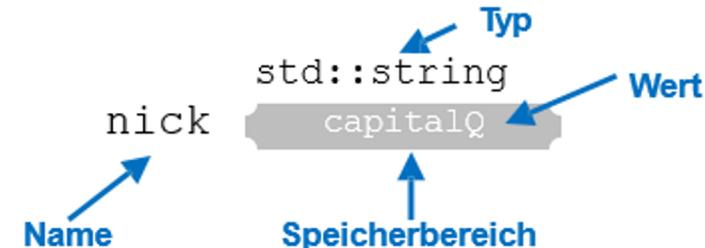
Anweisungen

- `int yob { 1997 };`
- Diese Quellcodezeile ist eine C++ Anweisung.
→ ein Schritt im Programmablauf
- Anweisungen enden stets mit einem Semikolon ;

```
int yob { 1997 };
```



```
std::string nick { "capitalQ" };
```





Anweisungen

- Die beiden Anweisungen initialisieren **Variablen**
 - eine vom Typ `int` namens `yob`
 - eine vom Typ `std::string` namens `nick`
- 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.



Namen, Gültigkeit von Namen

- `std::string nick { "capitalQ" };`
- Alle Namen in einem C++ Programm (wie std, main, nick, usw.) sind in bestimmten Bereichen des Programms gültig.
- Jeder Name muss **eindeutig** sein
 - Identische Namen dürfen in C++ Programmen nicht mehrfach mit unterschiedlichen Bedeutungen vorkommen.
- Zum eingrenzen der Gültigkeit eines Namens können Namensräume (`namespace`) deklariert werden.
 - Unter Angabe eines Namensraums sind Namen dann immer eindeutig, denn innerhalb eines namespace darf es jeden Namen nur einmal geben.
 - Beispiel: `std::string` meint `string` aus dem namespace namens `std`.



Auflösung des Gültigkeitsbereichs (Operator ::)

- `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 von `string` anzugeben (den namespace `std`) wird der Operator `::` verwendet.
- Im Namensraum `std` befindet sich die gesamte C++ Standardbibliothek (StdLib).
 - `string` aus der StdLib liegt daher auch im Namensraum `std`.
- Die namespace-Namen müssen eindeutig sein.
 - Es gibt nur einen namespace namens `std`.



Gültigkeitsbereiche

- Globaler Bereich
 - Außerhalb jedes anderen Scopes – überall gültig.
- Benannter Namensraum
 - Gültig innerhalb eines benannten Bereichs (namespace).
 - Namensräume erlauben gleiche Namen mehrfach und unterschiedlichem Kontext zu verwenden
- 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 (einzig) Anweisung.



Beispiel für Gültigkeitsbereiche

```
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

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 lästig werden.
```

- using-Direktive

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

- using-Deklaration

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



Rückgabewert

- `return 0;`
- Mit der return-Anweisung gibt die main() Funktion den Wert 0 (Null) als ganze Zahl (Typ int) zurück.
 - Ähnlich wie Variablen haben Funktionen einen Wert.
- 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.
- Mit einem Rückgabewert 0 wird üblicherweise angezeigt, dass die Funktion erfolgreich beendet wurde.
- Rückgabewerte ungleich 0 sollen üblicherweise besondere Vorkommnisse zur Laufzeit anzeigen.



das klassische erste C++ Programm

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



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 Quellcode zwischen doppelten oberen Anführungszeichen ".
 - 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 '
 - Typ für einzelne Zeichen: `char`.



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



Ausgabe in C++

- `std::cout << "Hallo C++!\n";`
- Der Standard-Ausgabestrom `cout` und der Stromausgabe-Operator `<<` der StdLib

`cout << "Hallo C++!\n"`

Linker Operand:
ein Ausgabestrom.

Rechter Operand:
ein Zeichenketten-Literal.

- Unterscheidet den Typ seines rechten Operanden
 - `bool, char, int, double, std::string.`
- StdLib kennt vier Standard E/A-Ströme:

| | | |
|-------------------|-----------------------|---|
| <code>cin</code> | Zeicheneingabe | i.d.R. von der Tastatur, |
| <code>cout</code> | Zeichenausgabe | i.d.R. auf den Bildschirm, |
| <code>clog</code> | Zeichenausgabe | i.d.R. auf den Bildschirm, für Kontrollmeldungen, |
| <code>cerr</code> | Zeichenausgabe | i.d.R. auf den Bildschirm, für Fehlermeldungen. |



Funktionen und Rückgabewerte

```
1. #include <iostream>
2. using std::cout;
3.
4. double square( double x ) { // das Quadrat einer Gleitkommazahl berechnen
5.     return x * x;
6. }
7.
8. void print_square( double x ) {
9.     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. }
```



Finden Sie die Fehler?

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

1. include <iosteam>
2. int main
03 {
4.     std::string nickname { "Leibniz" };
5.     return o;
06 }

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



Ende gut, Alles gut ;-)

cppreference.com