

Programmieren 1 – Informatik C, der Einstieg

Prof. Dr.-Ing. Maike Stern | 17.10.2023

- Organisatorisches
- Was ist Programmieren?
- Programmieren mit Scratch

- Einführung in C
- Hallo, Welt!
- Ein- und Ausgabe
- Variablen, Datentypen und der &-Operator
- Entwicklungsumgebungen

- Scratch ist eine grafische Programmiersprache mit Online-Editor
 - Keine Installation notwendig
 - Sehr simple Syntax, kaum fehleranfällig
 - Code-Editor und gleichzeitig grafische Ausgabe
 - Die Eingaben werden direkt visualisiert
- C ist eine hardwarenahe Programmiersprache
 - Installation von Compiler, Texteditor und Konsole bzw. integrierter Entwicklungsumgebung (IDE)
 - Umfangreiche Syntax, die Fehler nicht verzeiht
 - Keine native grafische Ausgabe
 - Visualisierung nur mit speziellen externen Bibliotheken möglich

Einschub Programmierparadigmen

maschinenorientiert

```
graph TD; A[maschinenorientiert] --> B[Maschinensprache]; A --> C[Assembler];
```

Maschinensprache

- elementare Befehle der CPU
- ist abhängig von der Maschinenarchitektur
- sehr eingeschränkter Befehlssatz
- jeder Befehl ist eine Folge von 0en und 1en

Assembler

- wie Maschinensprache, aber Befehle sind lesbare Abkürzungen (MOV, MULT, ADD)
- Register haben feste Namen
- wird z.B. für Gerätetreiber genutzt, da sehr effizient und es können architekturspezifische Befehle genutzt werden

maschinenorientiert

Maschinensprache

Assembler

Speicher:

Adresse:

0x00B813A0	55	8b	ec	81	ec	c0	00	00	00	53	56	57	8d	bd	40	ff	ff	ff	b9	30	00	00
0x00B813B0	00	b8	cc	cc	cc	cc	f3	ab	8b	f4	68	3c	57	b8	00	ff	15	bc	82	b8	00	83
0x00B813C0	c4	04	3b	f4	e8	66	fd	ff	ff	33	c0	5f	5e	5b	81	c4	c0	00	00	00	3b	ec
0x00B813E2	88	54	fd	ff	ff	8b	e5	5d	c3	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x00B813F8	cc	cc	cc	cc	cc	cc	ff	25	bc	82	b8	00	cc	cc	cc	cc	cc	cc	cc	cc	cc	cc
0x00B8140E	cc	cc	75	01	c3	55	8b	ec	83	ec	00	50	52	53	56	57	8b	45	04	6a	00	50
0x00B81424	e8	80	fd	ff	ff	83	c4	08	5f	5e	5b	5a	58	8b	e5	5d	c3	cc	cc	cc	cc	cc
0x00B8143A	cc	cc	cc	cc	cc	cc	8b	ff	55	8b	ec	51	53	56	57	33	ff	8b	f2	39	3e	8b

Bedeutung:
(Prozessortyp abhängig)

00B813A0	push	ebp
00B813A1	mov	ebp, esp
00B813A5	sub	esp, 0C0h
00B813A9	push	ebx
00B813AA	push	esi
00B813AB	push	edi

maschinenorientiert

Maschinensprache

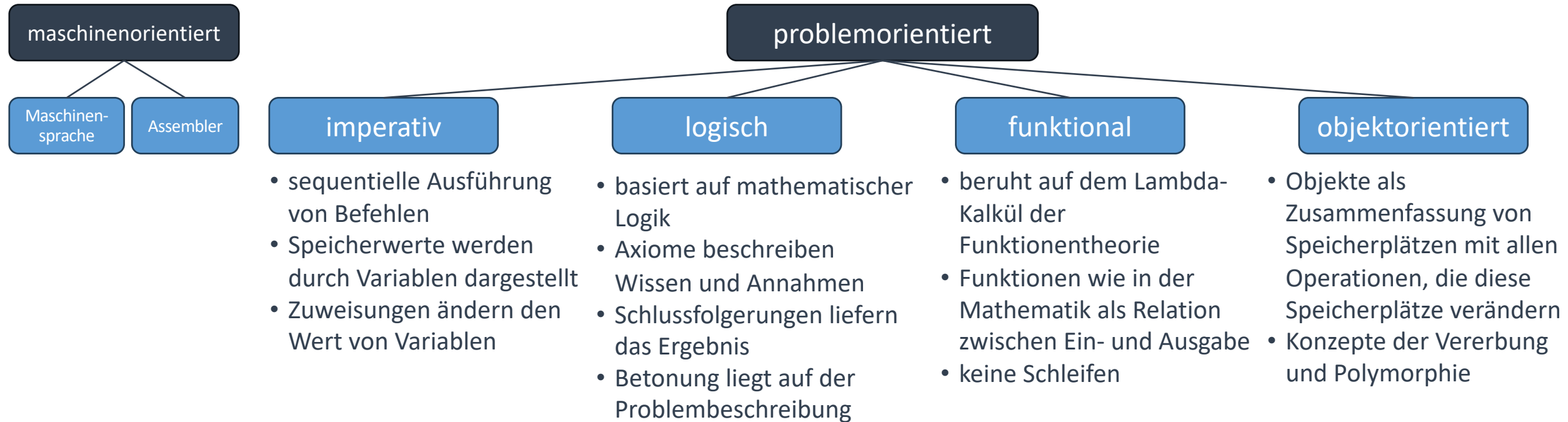
Assembler

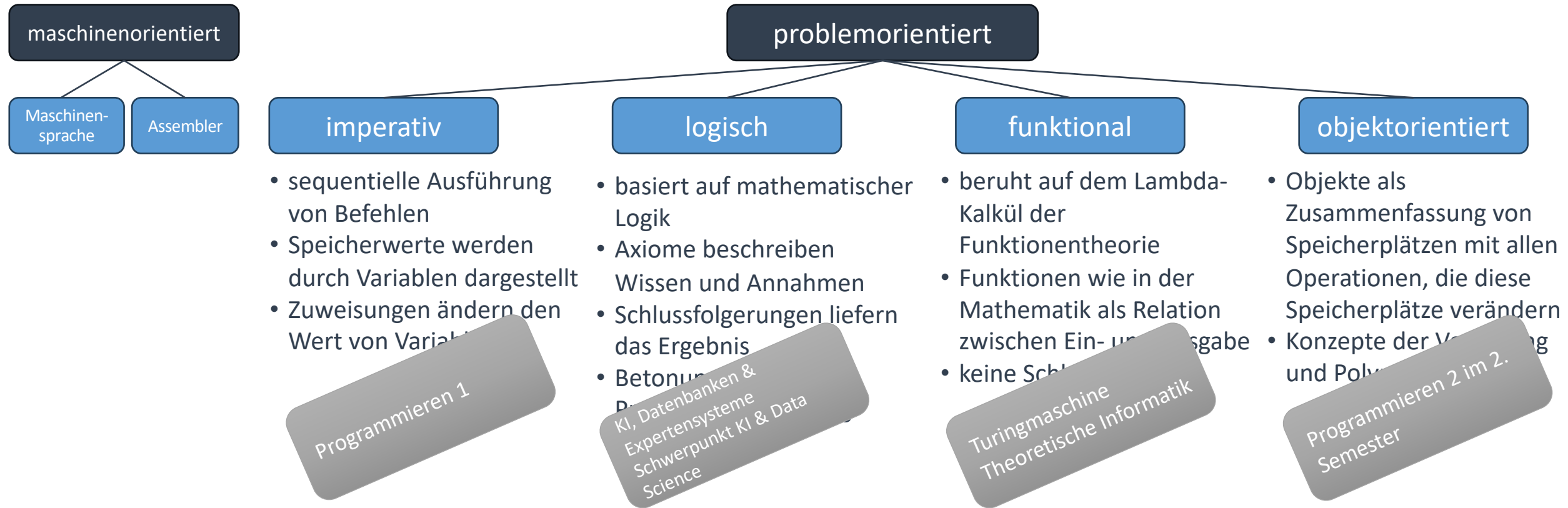
```
SECTION .data          ; DATEN
msg:  db "Hello World",10 ; Diesen Text wollen wir ausgeben
                                ; Die 10 am Ende bedeutet "naechste Zeile".
                                ; Es handelt sich um einen ASCII-Code (Details
                                ; siehe http://www.asciitable.com/)
len:   equ $-msg          ; Wir berechnen die Laenge des Text, indem
                                ; wir die Speicheradresse von msg von der
                                ; aktuelle Speicheradresse ("$$") subtrahieren

SECTION .text          ; PROGRAMMCODE
global main            ; Das Programm startet bei "main"
main:
    mov edx, len        ; In edx tragen wir die Laenge ein.
                                ; edx ist ein sogenanntes "Register" (Details siehe
                                ; https://de.wikipedia.org/wiki/Register\_\(Computer\))

    mov ecx, msg         ; In ecx die Adresse des Textes
    mov ebx, 1           ; 1 steht fuer "stdout" = Bildschirm
    mov eax, 4           ; 4 steht fuer "Ausgabe"
    int 0x80             ; Mit Interrupt 80 hex rufen wir den
                                ; Linux Kernel auf

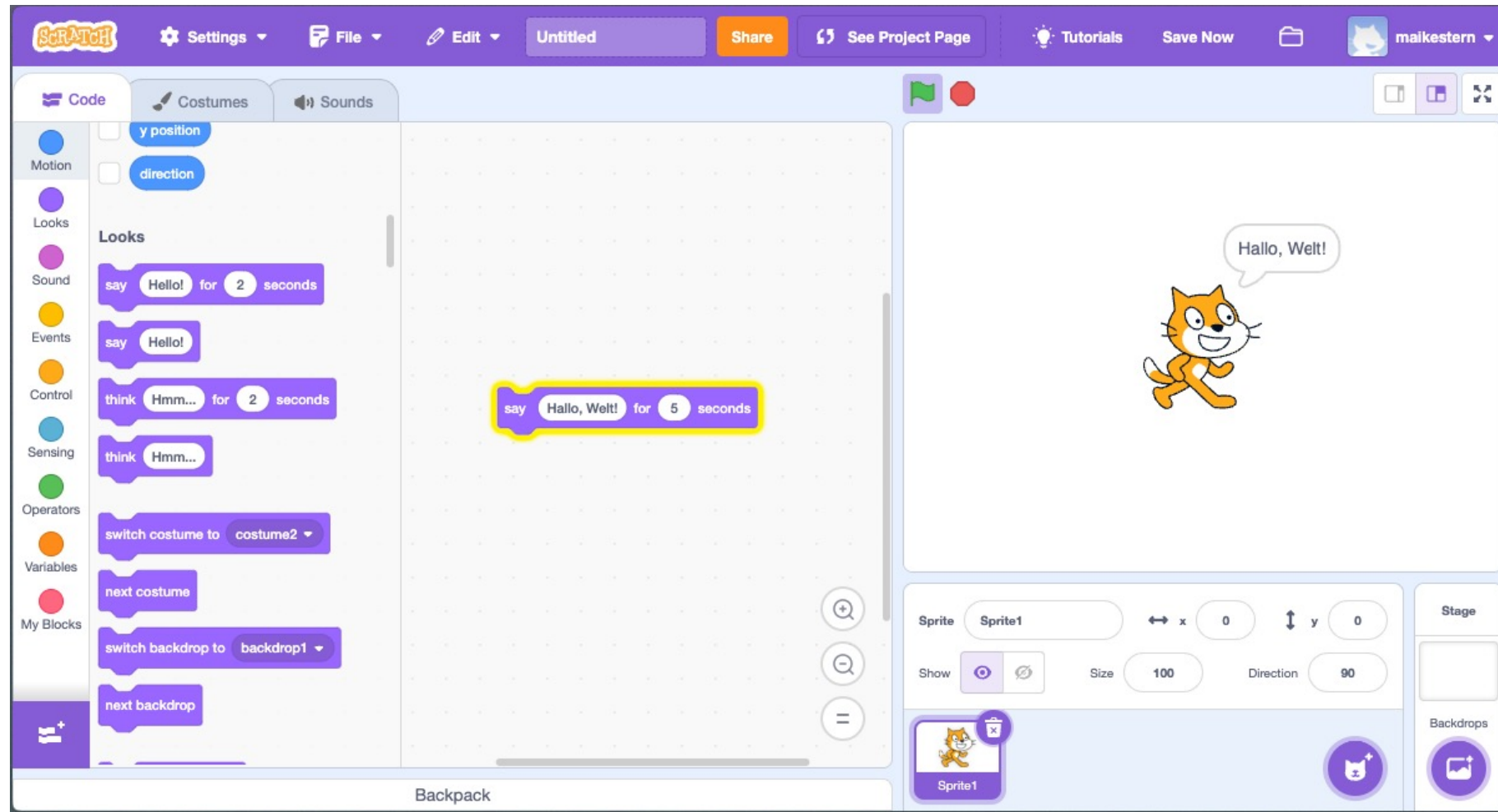
    mov ebx, 0           ; 0 steht fuer "normal beendet"
    mov eax, 1           ; 1 steht fuer "programm beenden"
    int 0x80
```

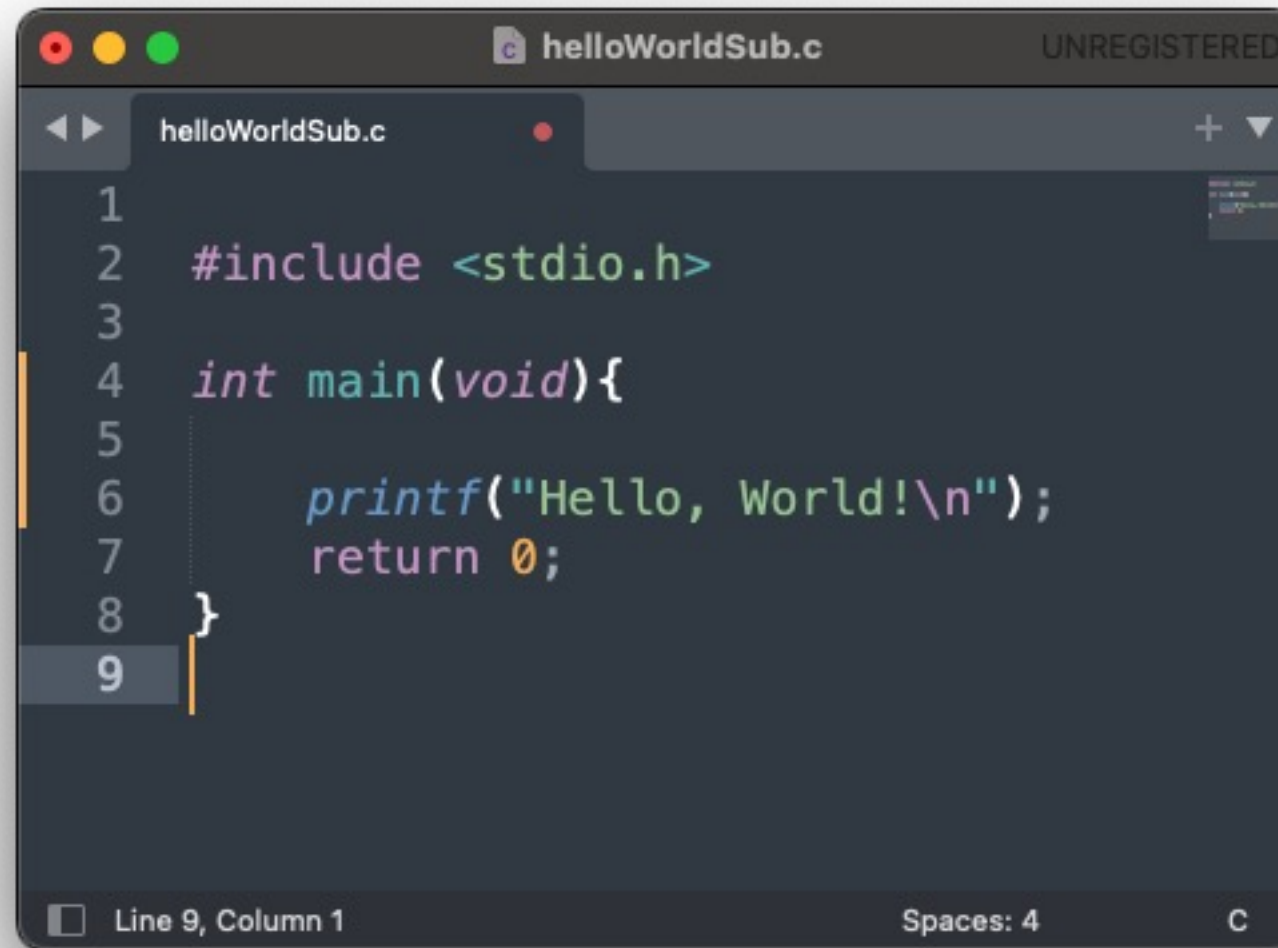


Hello, World!

Das erste Programm in C



„Hallo Welt“ – das erste Programm

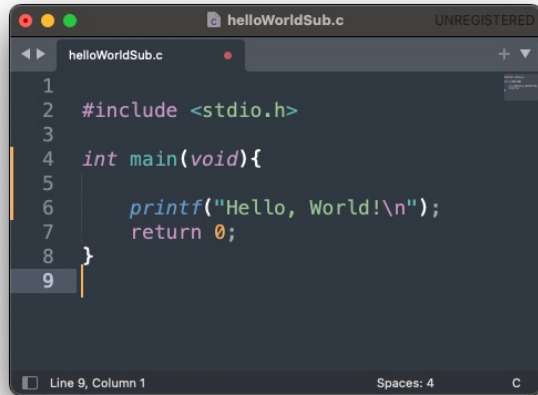


```
1
2  #include <stdio.h>
3
4  int main(void){
5
6      printf("Hello, World!\n");
7      return 0;
8  }
9
```

The screenshot shows a code editor window with a dark theme. The title bar at the top reads 'helloWorldSub.c' and 'UNREGISTERED'. The code is written in C and is as follows:

```
1
2  #include <stdio.h>
3
4  int main(void){
5
6      printf("Hello, World!\n");
7      return 0;
8  }
9
```

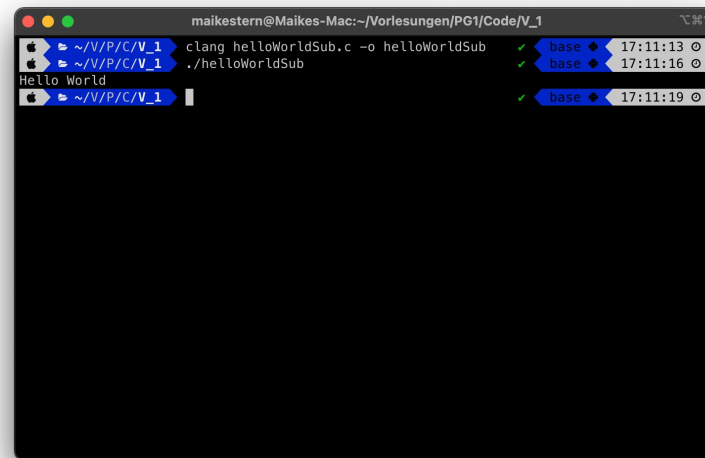
The cursor is positioned at the end of line 9. The status bar at the bottom indicates 'Line 9, Column 1', 'Spaces: 4', and the language is set to 'C'.



```
1 #include <stdio.h>
2
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

Texteditor mit C-Quellcode
HelloWorld.c

compiler



```
maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1
$ clang helloWorldSub.c -o helloWorldSub
$ ./helloWorldSub
Hello World
```


Konsole

1. ruft den Compiler auf, der den C-Quellcode zu Maschinencode kompiliert
und mit externen Header-Dateien linkt

Maschinencode
HalloWorld

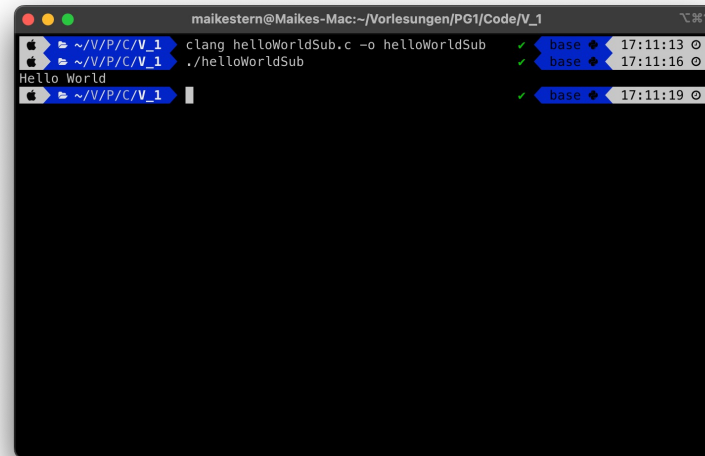
```
01101010001
11010100101
01010001011
11101010111
00011101010
01010101011
01010100101
11010100011
```

(für das Betriebssystem
ausführbare Datei)



```
helloWorldSub.c
1
2 #include <stdio.h>
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

Texteditor mit C-Quellcode
HelloWorld.c



```
maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1
$ clang helloWorldSub.c -o helloWorldSub
$ ./helloWorldSub
Hello World
```

Konsole

1. ruft den Compiler auf, der den C-Quellcode zu Maschinencode kompiliert und mit externen Header-Dateien linkt
2. führt den Maschinencode aus und schreibt „Hello World“ in die Konsole

Maschinencode
HalloWorld

01101010001
11010100101
01010001011
11101010111
00011101010
01010101011
01010100101
11010100011

(für das Betriebssystem
ausführbare Datei)

Hello, World!

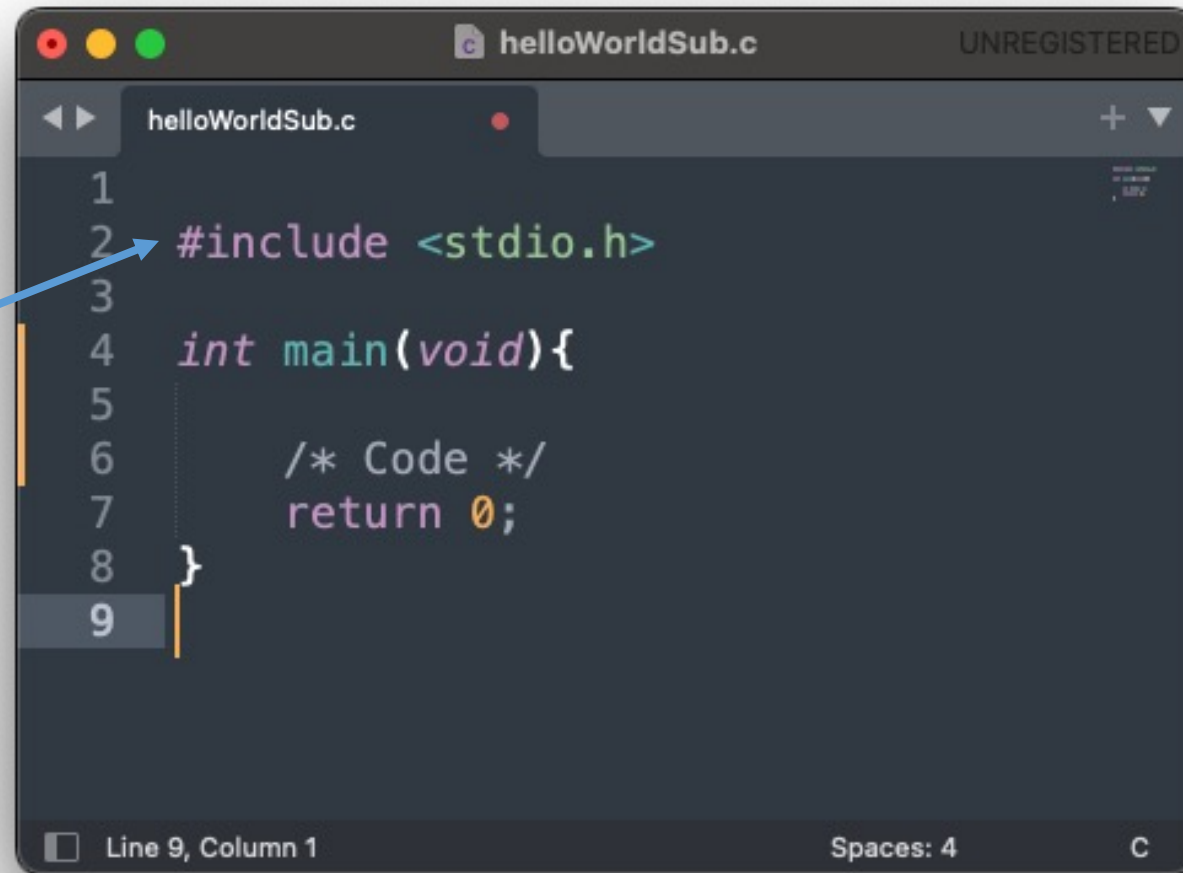
Die Syntax

„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax

Einbinden von header-Dateien mittels Include-Anweisung: Fügt weitere Funktionen hinzu, hier Standard-Eingabe- und Ausgabefunktionen

#include ist eine Präprozessoranweisung und nicht Bestandteil der Sprache C



```
1
2 #include <stdio.h>
3
4 int main(void){
5
6     /* Code */
7     return 0;
8 }
9
```

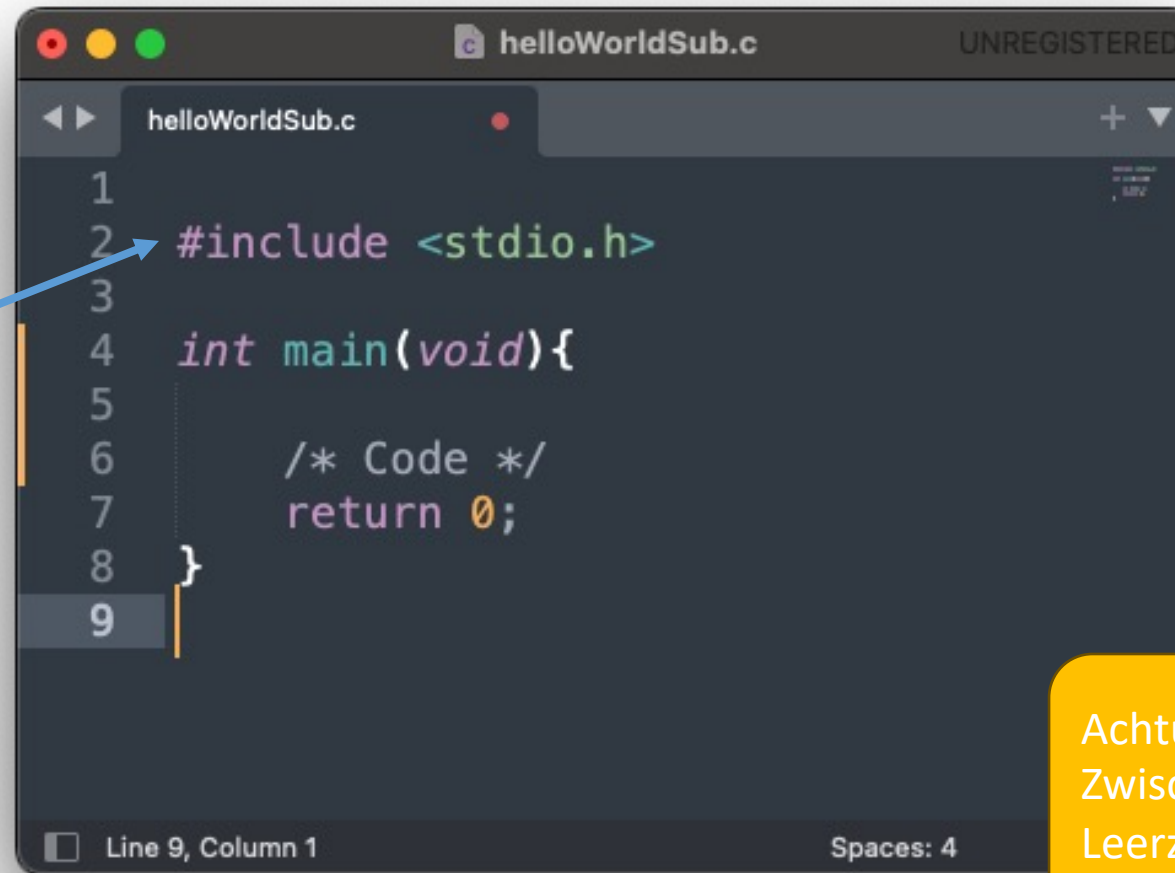
Line 9, Column 1 Spaces: 4 C

„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax

Einbinden von header-Dateien mittels Include-Anweisung: Fügt weitere Funktionen hinzu, hier Standard-Eingabe- und Ausgabefunktionen

#include ist eine Präprozessoranweisung und nicht Bestandteil der Sprache C

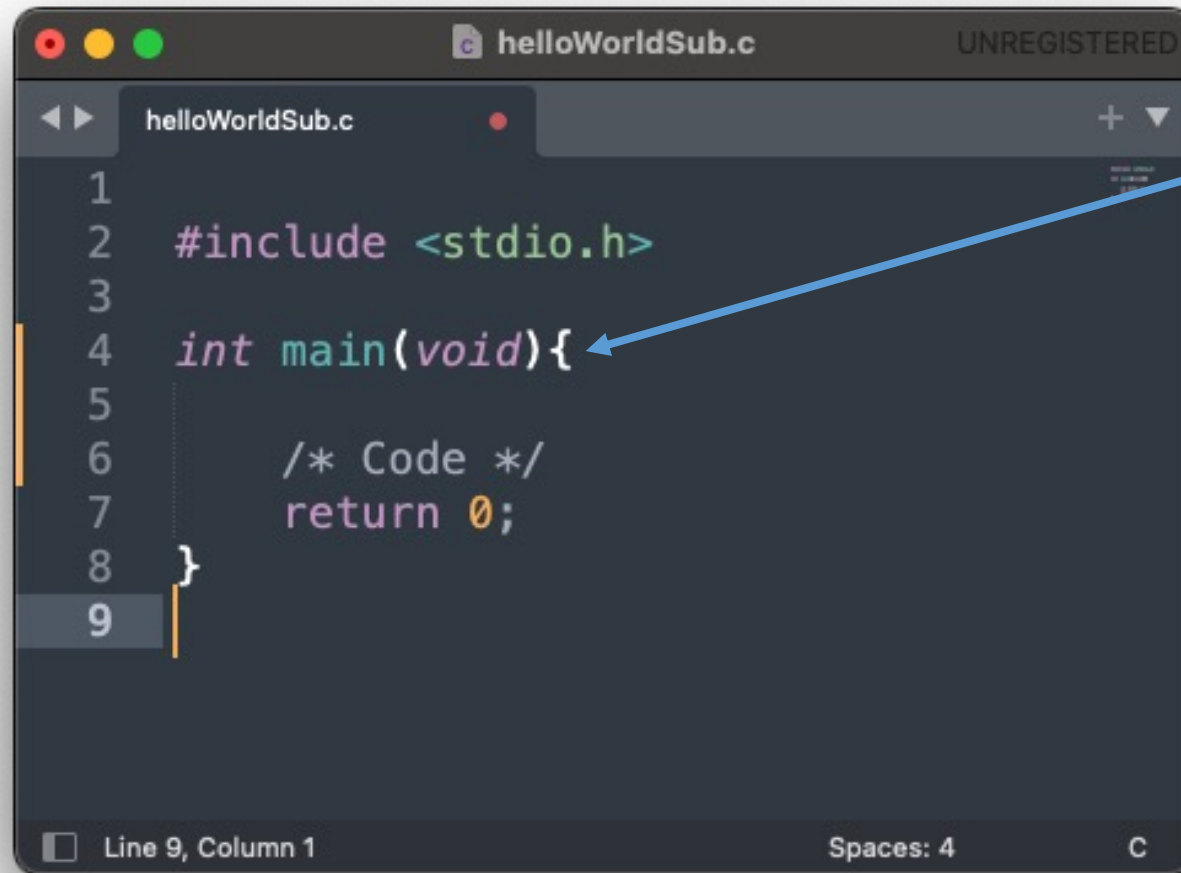


```
1
2 #include <stdio.h>
3
4 int main(void){
5
6     /* Code */
7     return 0;
8 }
9
```

Achtung!
Zwischen # und include ist kein Leerzeichen!

„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax



```
1
2  #include <stdio.h>
3
4  int main(void){
5
6      /* Code */
7      return 0;
8  }
9
```

Line 9, Column 1 Spaces: 4 C

Die Main-Funktion legt
den Startpunkt des
Programms fest

Programmablauf



```
1
2 #include <stdio.h>
3
4 int main(void){
5
6     /* Code */
7     return 0;
8 }
9
```

Das Programm startet bei der Main-Funktion.

In der Main-Funktion werden die Code-Anweisungen Schritt für Schritt ausgeführt.

Return beendet die Main-Funktion und gibt 0 aus – das heißt, das Programm wurde fehlerfrei beendet.

„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax

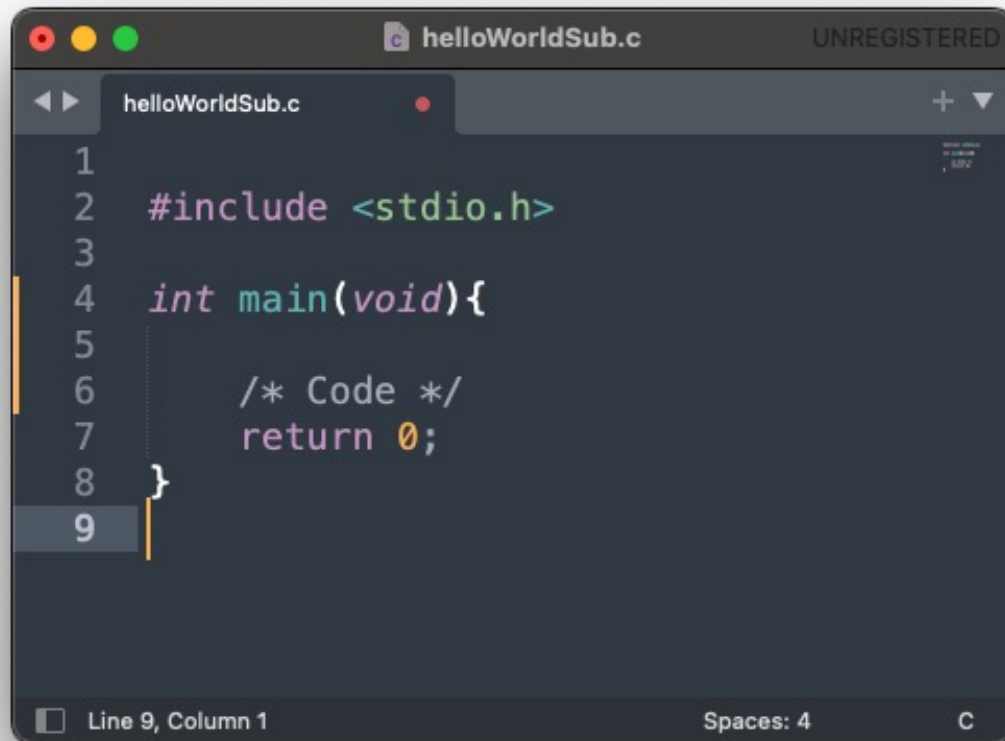
Syntax der Main-Funktion:

```
Rückgabety main(Parameter){  
    // code  
    return Rückgabewert;  
}
```

Der **Rückgabety** gibt an, welcher Datentyp von der Funktion mittels **return**-Statement zurückgegeben wird.

Hier eine Integer (int), also eine Ganzzahl, was obligatorisch ist bei der Main-Funktion. Die Zahl 0 steht dafür, dass das Programm korrekt beendet wurde.

Andere Funktionen können auch andere Datentypen (float, double, ...) bzw. nichts (void) zurückgeben. Im letzteren Fall kann das return-Statement weggelassen werden.



The screenshot shows a code editor window titled 'helloWorldSub.c' with a 'UNREGISTERED' label. The code is as follows:

```
1  
2  #include <stdio.h>  
3  
4  int main(void){  
5  
6      /* Code */  
7      return 0;  
8  }  
9
```

The status bar at the bottom indicates 'Line 9, Column 1', 'Spaces: 4', and a 'C' icon.

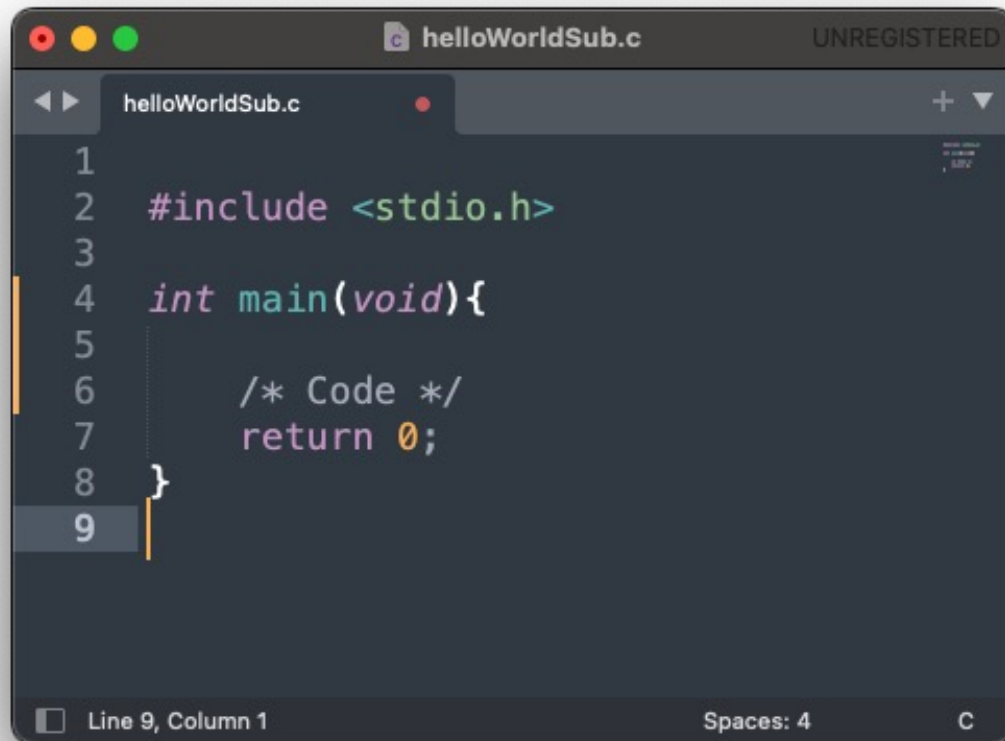
„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax

Syntax der Main-Funktion:

```
Rückgabetyp main(Parameter){  
    // code  
    return Rückgabewert;  
}
```

- Der **Rückgabetyp** gibt an, welcher Datentyp von der Funktion mittels **return**-Statement zurückgegeben wird.
- **Funktionsparameter** speichern die Eingabewerte. Hier ist der Parametertyp jedoch *void*. Das heißt, es wird kein Eingabewert an die Funktion übergeben
- Der **Anweisungsblock** der Funktion steht immer in geschweiften Klammern
- **Return** (optional): Der Rückgabewert gibt bei der Main-Funktion an, ob das Programm korrekt beendet wurde. Typisch: 0 = korrekt, ≥ 1 Fehler
- Das Semikolon beendet Anweisungen




The screenshot shows a code editor window titled 'helloWorldSub.c' with a dark theme. The code is as follows:

```
1  
2  #include <stdio.h>  
3  
4  int main(void){  
5  
6      /* Code */  
7      return 0;  
8  }  
9
```

The status bar at the bottom indicates 'Line 9, Column 1', 'Spaces: 4', and a 'C' icon.

„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax



```
1
2 #include <stdio.h>
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

Funktionsaufruf

- Die printf-Funktion schreibt auf die Konsole, hier: *Hello, World!*
- Die printf-Funktion ist im stdio.h-Header (Standard Input Output) beschrieben

Syntax Funktionsaufruf:

Funktionsname(Argument(e));

„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax



```
1
2 #include <stdio.h>
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

Funktionsaufruf

Syntax:

Funktionsname(Argument(e));

Zum Vergleich, die Funktions**deklaration**:

Rückgabewerttyp Funktionsname(Parameter){...}

```
int printf(const char *format, ...){
```

...

```
}
```


„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax



```
1
2 #include <stdio.h>
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

Line 9, Column 1 Spaces: 4 C

Funktionsaufruf

Syntax:

Funktionsname(Argument(e));

Zum Vergleich, die Funktions**deklaration**:

Rückgabewerttyp Funktionsname(Parameter){...}

Parameter: Variable, die in der Signatur der Funktion vorkommt. Parameter werden definiert, sind also Platzhalter.

Argument: der tatsächliche Wert, der einer Funktion zugeführt wird

„Hallo Welt“ – das erste Programm

Die grundlegende C-Syntax

Hier kommt der eigentliche Code hin, aktuell steht hier nur ein Kommentar, also Text, der vom Compiler ignoriert wird.

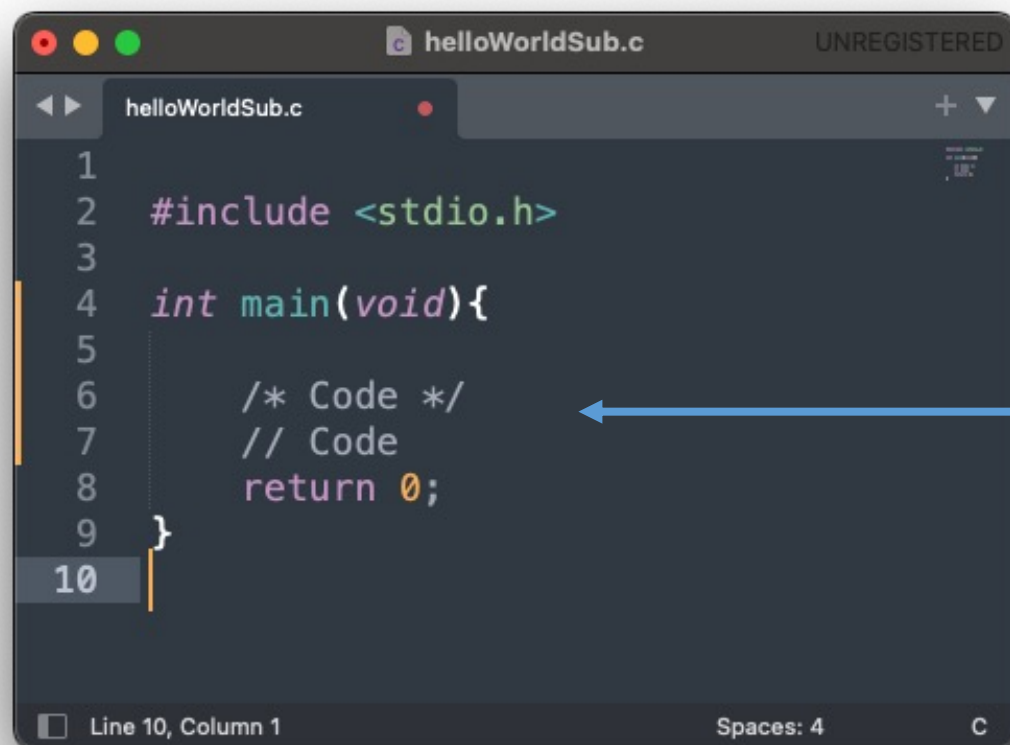
Für Kommentare gibt es zwei Varianten:

Insbesondere für mehrzeilige Kommentare:

`/* Code */`

Für einzeilige Kommentare:

`// Code`



```
1
2 #include <stdio.h>
3
4 int main(void){
5     /* Code */
6     // Code
7     return 0;
8 }
9
10
```

Line 10, Column 1 Spaces: 4 C

C-Code kompilieren und ausführen

Mit Texteditor und Konsole (IDE kommt später)

Das erste Programm – wie ausführen?

Option 1: Texteditor + Konsole



```
1  #include <stdio.h>
2
3
4  int main(void){
5
6      printf("Hello, World!\n");
7      return 0;
8  }
9
```

Line 9, Column 1 Spaces: 4 C

Textdatei mit dem Programm



```
maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1
~/V/P/C/V_1 clang helloWorldSub.c -o helloWorldSub
~/V/P/C/V_1 ./helloWorldSub
Hello World
~/V/P/C/V_1
```

base 17:11:13
base 17:11:16
base 17:11:19

Konsole

Das erste Programm – wie ausführen?

Option 1: Texteditor + Konsole

Compiler-Befehle:

- `compiler Programmname.c // speichert das kompilierte Programm als a.out ab`
`./a.out // führt das kompilierte Programm aus`



```
maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1
~/V/P/C/V_1 clang helloWorldSub.c -o helloWorldSub
~/V/P/C/V_1 ./helloWorldSub
Hello World
~/V/P/C/V_1
```

Konsole

Das erste Programm – wie ausführen?

Option 1: Texteditor + Konsole

Compiler-Befehle:

- `compiler Programmname.c` // speichert das kompilierte Programm als `a.out` ab
- `./a.out` // führt das kompilierte Programm aus

Weitere Optionen:

- `compiler Programmname.c -o neuerName` // speichert das kompilierte Programm unter dem angegebenen Namen
- `compiler -Wall Programmname.c` // zeigt alle Compilerwarnungen beim kompilieren an



```
maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1
~/V/P/C/V_1 clang helloWorldSub.c -o helloWorldSub
~/V/P/C/V_1 ./helloWorldSub
Hello World
~/V/P/C/V_1
```

Konsole

- Die Programmiersprache C hat im Vergleich zu Scratch eine umfassende, fehleranfällige Syntax
- Code muss so geschrieben werden, dass der Compiler und man selbst / andere ihn verstehen

```
#include <stdio.h>
int main(){printf("Hi");return 0;}
```
- Bestimmte Programmfunktionen werden mittels header-Dateien eingebunden

```
#include <stdio.h>
```
- Die Main-Funktion ist der Startpunkt des Programms
- Der Compiler wandelt den C-Quellcode in Maschinencode bzw. eine ausführbare Datei um, welche über die Konsole aufgerufen werden kann
- Außerdem findet der Compiler Syntaxfehler und gibt (mit der entsprechenden Einstellung) Warnungen aus
Typisch: Semikolon vergessen, so dass zwei Statements als eins gelesen werden

Ein- & Ausgabe


```
#include <stdio.h>

int main(){
    printf("Hello, World!\n");
    return 0;
}
```

```
>> ./helloWorld
Hello, World!
```

- printf ist eine Funktion, die als Argument übergebenen Text auf die Konsole schreibt
- Dient der Kommunikation mit dem/der Programmierer:in bzw. dem/der Nutzer:in
- Syntax Funktionsaufruf:

Argument

Funktionsname

`printf("Hello, World!\n");` ← *Semikolon beendet das statement*

```
#include <stdio.h>

int main(){
    printf("Hello, World!\n");
    return 0;
}
```

```
>> ./helloWorld
Hello, World!
```

- printf ist eine Funktion, die als Argument übergebenen Text auf die Konsole schreibt
- Dient der Kommunikation mit dem/der Programmierer:in bzw. dem/der Nutzer:in
- Syntax:

Text in Anführungszeichen: ein sogenanntes String-Literal. Einfache Anführungszeichen wie z.B 'c' sind für einzelne Zeichen (character) reserviert

printf("Hello, World!\n");

auszugebender Text

\n ist ein
Zeilenumbruch

Möglichkeiten, Textausgaben mit printf zu formatieren

```
int main(){
    int i = 0, j = 1, k = 2;
    float m = 0.1;

    printf("\n\nThis outputs plain text.\n");
    printf("Let us print the first variable i: %d\n", i);
    printf("Let us print all the variables: %d, %d, and %d\n", i,j,k);
    printf("Let us calculate %d squared: %d\n", k, k*k);
    printf("Finally, a floating point: %f\n\n\n", m);

    return 0;
}
```

```
#include <stdio.h>

int main(){
    int i;
    printf("Please enter a \
          number: \n");
    scanf("%d", &i);
    printf("Your number is: %d\n", i)

    return 0;
}
```

```
>> ./read
Please enter a number: 42
Your number is: 42
```

- Die scanf-Funktion liest Text über die Standardeingabe (Tastatur) ein
- Syntax:

Argument

Funktionsname

Semikolon beendet das statement

scanf("%d", &i);

```
#include <stdio.h>
```

```
int main(){  
    int i;  
    printf("Please enter a \\  
           number: \n");  
    scanf("%d", &i);  
    printf("Your number is: %d\n", i)  
  
    return 0;  
}
```

```
>> ./read  
Please enter a number: 42  
Your number is: 42
```

- Die scanf-Funktion liest Text über die Standardeingabe (Tastatur) ein

- Syntax:

schreibt den eingelesenen Wert
in die Variable *i*

→ wohin

scanf("%d", &i);

Formatiertes Einlesen der Eingabe

%d steht für eine Ganzzahl mit mindestens 16 Bit

→ was

Eingabe von Text über die Konsole Ablauf Texteingabe

```
#include <stdio.h>

int main(){
    int i;
    printf(" Enter an integer: \n");
    scanf("%d", &i);
    printf("Your number is: %d\n");
    return 0;
}
```



```
>> ./letsTalk
Please enter a number:
```

```
>> ./letsTalk
Please enter a number: 1337
```



```
#include <stdio.h>

int main(){
    int i;
    printf(" Enter an integer: \n");
    scanf("%d", &i);
    printf("Your number is: %d\n");
    return 0;
}
```



Die *scanf*-Funktion sendet eine Eingabeaufforderung an die Konsole

Die Konsole erwartet eine Eingabe

Der/die Nutzer:in tippt Text ein und schließt diesen mit der Eingabetaste ab

Der eingegebene Text wird in einer Variable im Programm gespeichert

```
#include <stdio.h>
```

```
int main(){  
    int i;  
    printf("Please enter a \\  
           number: \n");  
    scanf("%d", &i);  
    printf("Your number is: %d\n", i)  
  
    return 0;  
}
```

```
>> ./read  
Please enter a number: 42  
Your number is: 42
```

- Die scanf-Funktion liest Text über die Standardeingabe (Tastatur) ein
- Syntax:

scanf("%d", &i);

↙ schreibt den eingelesenen Wert in die Variable *i*

↑

Formatiertes Einlesen der Eingabe
%d steht für eine Ganzzahl mit mindestens 16 Bit

- Was sind Variablen?
- Was sind Datentypen?
- Was bedeutet das &?

Variablen, Datentypen & Adressoperator

Die *scanf*-Funktion liest formatierte Werte ein und speichert den Wert in einer Variablen. Das heißt, man gibt mit einem entsprechenden Typbezeichner vor, was für eine Art Wert in die Variable geschrieben wird.

```
scanf("%d", &i);
```



der Variablen *i* wird der im Format *%d* eingelesene Wert zugewiesen

Die *scanf*-Funktion liest formatierte Werte ein und speichert den Wert in einer Variablen. Das heißt, man gibt mit einem entsprechenden Typbezeichner vor, was für eine Art Wert in die Variable geschrieben wird.

```
int i; // Deklaration und Definition der Variablen i
scanf("%d", &i);
```

↑
der Variablen *i* wird der im Format *%d* eingelesene Wert zugewiesen

Variablen in der Programmierung sind ähnlich wie Variablen in der Mathematik. Eine Bezeichnung steht stellvertretend für einen bestimmten Wert.

Im Gegensatz zur Mathematik muss die Variable aber zunächst deklariert und definiert werden.

Deklaration

- Die Deklaration macht den Compiler mit dem **Namen** der Variablen bekannt (z.B. *i*) und verknüpft diesen Namen mit einem **Datentyp** (z.B. Integer: int):

`int i`

↑ ↑
Datentyp Variablenname

Deklaration

- Die Deklaration macht den Compiler mit dem **Namen** der Variablen bekannt (z.B. *i*) und verknüpft diesen Namen mit einem **Datentyp** (z.B. Integer: int):

`int i`
↖ ↗
Datentyp Variablenname

Definition

- Die Definition reserviert den Speicherplatz für die Variable

`int i;` deklariert also eine Variable mit dem Namen *i* und dem Datentyp Integer (int) & reserviert gleichzeitig den für eine Integer notwendigen Speicherplatz (Definition). Bei C vermischen sich (bis auf spezielle Fälle) also Definition und Deklaration

Deklaration

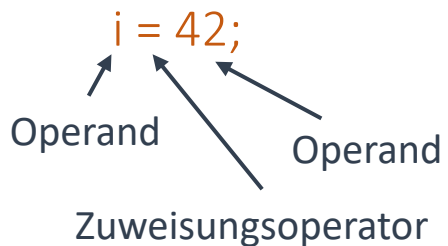
- Die Deklaration macht den Compiler mit dem Namen der Variablen bekannt (z.B. *i*) und verknüpft diesen Namen mit einem Datentyp (z.B. Integer: `int`).

Definition

- Die Definition reserviert den Speicherplatz für die Variable

Initialisierung

- Bei der Initialisierung wird der Variablen ein Anfangswert zugewiesen



Achtung!

Deklariert und definiert man eine Variable, sollte man sie immer auch gleich initialisieren!

Also: `int i = 42;`

Andernfalls ist der Wert der Variablen nicht definiert (in gewissem Sinne zufällig) → Fehlerquelle!

```
#include <stdio.h>

int main(){

    int i = 42;
    printf("The answer is: %d\n", i);

    i = 0;
    printf("Now it is: %d\n", i);

    return 0;
}
```

```
>> ./TheAnswer
The answer is: 42
Now it is: 0
```

- Variablen können (sollten!) zeitgleich deklariert, definiert und initialisiert werden
`int i = 42;`
- Variablen können im Programmverlauf verändert werden

Variablennamen sind eine Wissenschaft für sich, denn

- zum einen sind nicht alle Namen erlaubt. Zulässige Variablennamen:
 - beginnen mit einem Buchstaben oder Unterstrich (z.B. `_i`)
 - bestehen nur aus Buchstaben, Zahlen oder Unterstrichen
 - und sind keine reservierten Wörter in C

Was sind erlaubte Variablennamen?

x	i
42answer	WIDTH
Darth Vader	What's_that
noOrdinaryRabbit	else
lots_of_underscores	H_233_m_pp

Variablennamen sind eine Wissenschaft für sich, denn

- zum einen sind nicht alle Namen erlaubt. Zulässige Variablennamen:
 - beginnen mit einem Buchstaben oder Unterstrich (z.B. `_i`)
 - bestehen nur aus Buchstaben, Zahlen oder Unterstrichen
 - und sind keine reservierten Wörter in C

Was sind erlaubte Variablennamen?

<code>x</code>	<code>i</code>
<code>42answer</code>	<code>WIDTH</code>
<code>Darth Vader</code>	<code>What's _that</code>
<code>noOrdinaryRabbit</code>	<code>else</code>
<code>lots_of_underscores</code>	<code>H_233_m_pp</code>

Variablennamen sind eine Wissenschaft für sich, denn

- zum einen sind nicht alle Namen erlaubt. Zulässige Variablennamen:
 - beginnen mit einem Buchstaben oder Unterstrich (z.B. `_i`)
 - bestehen nur aus Buchstaben, Zahlen oder Unterstrichen
 - und sind keine reservierten Wörter in C
- zum anderen gibt es verschiedene Arten, **wie** Variablennamen geschrieben werden
 - camel case: `numberOfBugs`
 - pascal case: `NumberOfBugs`
 - snake case: `number_of_bugs`
 - screaming snake case: `NUMBER_OF_BUGS`
 - ungarische Notation: `iCounter`, `fResult`, `strName` (nicht empfehlenswert)
(https://de.wikipedia.org/wiki/Ungarische_Notation#Einw%C3%A4nde_gegen_die_ungarische_Notation)
 - und viele andere: [https://en.wikipedia.org/wiki/Naming_convention_\(programming\)](https://en.wikipedia.org/wiki/Naming_convention_(programming))

Variablennamen sind eine Wissenschaft für sich, denn

- zum einen sind nicht alle Namen erlaubt. Zulässige Variablennamen:
 - beginnen mit einem Buchstaben oder Unterstrich (z.B. `_i`)
 - bestehen nur aus Buchstaben, Zahlen oder Unterstrichen
 - und sind keine reservierten Wörter in C
- zum anderen gibt es verschiedene Arten, wie Variablennamen geschrieben werden
 - camel case: `numberOfBugs`
 - pascal case: `NumberOfBugs`
 - snake case: `number_of_bugs`
 - screaming snake case: `NUMBER_OF_BUGS`
 - und viele andere:
[https://en.wikipedia.org/wiki/Naming_convention_\(programming\)](https://en.wikipedia.org/wiki/Naming_convention_(programming))

Nach welcher Konvention Namen geschrieben werden, hängt auch davon ab wofür der Name steht (und welche Sprache man verwendet):

- Variablen: `variableName`
- Konstanten: `ALL_CAPS`
- Funktionen: `FunctionName`
- ...

Wichtig ist: konstant sein! Keine Glaubensfrage draus machen und sich bei gemeinsamen Projekten auf eine Konvention einigen.

Variablennamen sind eine Wissenschaft für sich, denn

- zum einen sind nicht alle Namen erlaubt
- zum anderen gibt es verschiedene Arten, wie Variablennamen geschrieben werden
- und außerdem sind nicht alle Variablen- und Funktionsnamen gleich gut!
 - Variablennamen sollten selbsterklärend sein und die Funktion des Code kommentieren

Worum geht's in diesem
Codeausschnitt?

(das ist natürlich ein absichtlich
schlechtes Beispiel...)

```
class X:

    def __init__(self, y=None):
        """Inits the X."""
        self.y = y

    def g_l_d(self, b, n_c_c, n_r, **kwargs):

        dic = ds.l_ds(b, n_c_c, n_r, **kwargs)
        self.d = dic['df']
        self._dot_str = dic['d_g']
        self.ate = dic['ate']

    def disp_g(self):
        g = pydot.g_f_dd(self._dot_str)
        self._dot = g[0]
        self._png = self._dot.c_p(prog='dot')
        print('True causal graph:\n')
        display(Image(self._png))
```



coding horror

- Variablennamen sind unverständlich
- Keine Kommentare
- Der Code ist nicht selbsterklärend

Achtung, Python-Code!

Worum geht's in diesem Codeausschnitt?

```
class DataGenerator:

    def __init__(self, paths=None):
        self.paths = paths

    def generate_linear_dataset(self, beta, n_common_causes, n_rows, **kwargs):
        dic = datasets.linear_dataset(beta, n_common_causes, n_rows, **kwargs)
        self.data = dic['df']
        self._dot_str = dic['dot_graph']
        self.ate = dic['ate']

    def display_graph(self):
        graphs = pydot.graph_from_dot_data(self._dot_str)
        self._dot = graphs[0]
        self._png = self._dot.create_png(prog='dot')
        print('True causal graph:\n')
        display(Image(self._png))
```

Achtung, Python-Code!



clean code

- für alle verständliche Variablennamen, die den Code kommentieren
- sinnvolle, ergänzende Kommentare
- es wird schon beim ersten Blick klar, worum es in dem Code geht

```
class DataGenerator:
    """Main class for generating synthetic data.

    Attributes:
        paths: A cause2e.PathManager managing paths and file names.
        data: A pandas.DataFrame containing the data.
        ate: A float indicating the average treatment effect.

        This is what we want to estimate in the subsequent causal analysis.
    """
    def __init__(self, paths=None):
        """Inits the DataGenerator."""
        self.paths = paths

    def generate_linear_dataset(self, beta, n_common_causes, n_rows, **kwargs):
        """Generates a linear dataset.

        Args:
            beta: A float describing the strength of the causal effect.
            n_common_causes: An integer indicating the number of common causes that treatment
                variable and outcome variable share.
            n_rows: An integer indicating the number of samples to be generated.
        """
        dic = datasets.linear_dataset(beta, n_common_causes, n_rows, **kwargs)
        self.data = dic['df']
        self._dot_str = dic['dot_graph']
        self.ate = dic['ate']

    def display_graph(self):
        """Displays the causal graph that was used to generate the data."""
        # TODO: should be handled by graph module
        graphs = pydot.graph_from_dot_data(self._dot_str)
        self._dot = graphs[0]
        self._png = self._dot.create_png(prog='dot')
        print('True causal graph:\n')
        display(Image(self._png))
```

Achtung, Python-Code!

Variablennamen sind eine Wissenschaft für sich, denn

- zum einen sind nicht alle Namen erlaubt
- zum anderen gibt es verschiedene Arten, wie Variablennamen geschrieben werden
- und außerdem sind nicht alle Variablen- und Funktionsnamen gleich gut!
 - Variablennamen sollten selbsterklärend sein und die Funktion des Code kommentieren
 - Man sollte keine bzw. nur allgemein anerkannte Abkürzungen nutzen (z.B. num für number of...)
 - einzelne Buchstaben nur für Schleifenvariablen bzw. in mathematischen Formeln
for(int i = 0; i<20; i++)...
 - Englische Variablen- und Funktionsnamen verwenden

Variablennamen sind eine Wissenschaft für sich, denn

- zum einen sind nicht alle Namen erlaubt
- zum anderen gibt es verschiedene Arten, wie Variablennamen geschrieben werden
- und außerdem sind nicht alle Variablen- und Funktionsnamen gleich gut!
 - Variablennamen sollten selbsterklärend sein und die Funktion des Code kommentieren
 - Man sollte keine bzw. nur allgemein anerkannte Abkürzungen nutzen (z.B. num für number of...)
 - einzelne Buchstaben nur für Schleifenvariablen bzw. in mathematischen Formeln
for(int i; i<20; i++)...
 - Englische Variablen- und Funktionsnamen verwenden

Natürlich fällt einem beim Programmieren nicht immer sofort ein guter Name ein. In diesem Fall: Immer denselben Platzhalter verwenden, z.B. *bla*, *blub*, ... Oder noch besser, international bekannte Platzhalter, z.B. *foo*, *bar*, *baz*, ... Aber später nicht das Ersetzen vergessen!

Zur Geschichte von foo:

<https://datatracker.ietf.org/doc/html/rfc3092>

- **Deklaration:** Variablen müssen mittels Deklaration dem Compiler bekannt gemacht werden, das heißt Datentyp und Name werden festgelegt
- **Definition:** Zeitgleich mit der Deklaration wird auch der dem Datentyp entsprechende Speicherplatz reserviert
- **Initialisierung:** Variablen dienen als Platzhalter für Werte, die mit einem Anfangswert initialisiert werden. Im Verlauf des Programms kann dieser Wert durch Zuschreibung geändert werden:

`int i = 1337;`

- **Variablennamen** (genau wie Funktionsnamen) sollten mit Bedacht gewählt werden

```
#include <stdio.h>

int main(){
    int i;
    printf("Please enter a \
           number: \n");
    scanf("%d", &i);
    printf("Your number is: %d\n", i)

    return 0;
}
```

Variablen, Datentypen & Adressoperator

Eingabe von Text über die Konsole

```
#include <stdio.h>
```

```
int main(){  
    int i;  
    printf("Please enter a \\  
           number: \n");  
    scanf("%d", &i);  
    printf("Your number is: %d\n", i)  
  
    return 0;  
}
```

```
>> ./read  
Please enter a number: 42  
Your number is: 42
```

- Die scanf-Funktion liest Text über die Standardeingabe (Tastatur) ein
- Syntax:

scanf("%d", &i);

↙ schreibt den eingelesenen Wert in die Variable *i*

↑

Formatiertes Einlesen der Eingabe
%d steht für eine Ganzzahl mit mindestens 16 Bit

- ~~Was sind Variablen?~~
- Was sind Datentypen?
- Was bedeutet das &?

- Wie wir gesehen haben, wird bei der
 - Deklaration von Variablen der Datentyp festgelegt
 - und bei der Definition Speicherplatz reserviert
- Das heißt, die Datentypen legen fest, wieviel Speicherplatz reserviert wird

Variablentyp- & name

Adresse

Speicherplatz

int i

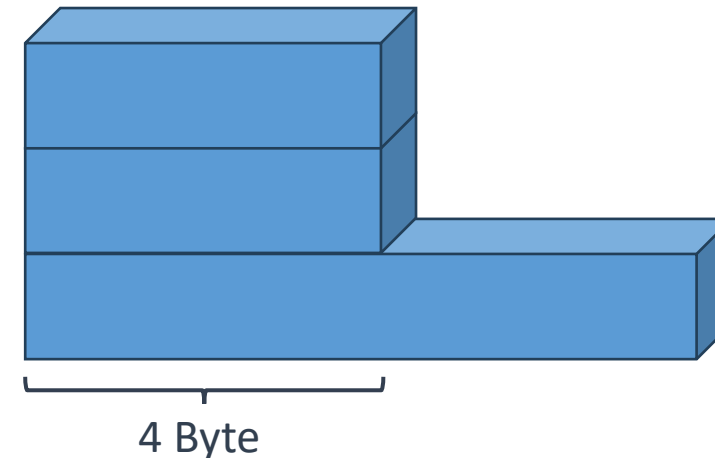
0x16ee1ae80

int j

0x16ee1ae7c

double k

0x16ee1ae78



- Wie wir gesehen haben, wird bei der
 - Deklaration von Variablen der Datentyp festgelegt
 - und bei der Definition Speicherplatz reserviert
- Das heißt, die Datentypen legen fest, wieviel Speicherplatz reserviert wird
- Bei der Initialisierung wird ein Wert in den Speicherbereich geschrieben

Variablentyp- & name

Adresse

Speicherplatz

int i = 1337

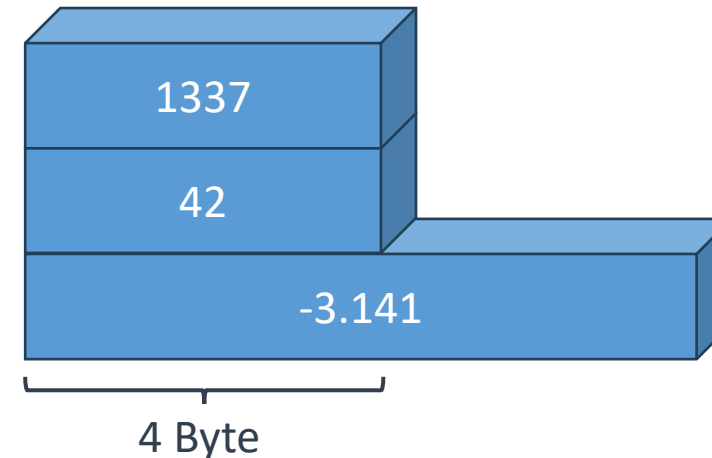
0x16ee1ae80

int j = 42

0x16ee1ae7c

double k = -3.141

0x16ee1ae78



- Wie wir gesehen haben, wird bei der
 - Deklaration von Variablen der Datentyp festgelegt
 - und bei der Definition Speicherplatz reserviert
- Das heißt, die Datentypen legen fest, wieviel Speicherplatz reserviert wird
- Bei der Initialisierung wird ein Wert in den Speicherbereich geschrieben

Variablentyp- & name	Adresse	Speicherplatz
int i = 1337	0x16ee1ae80	1337
int j = 42	0x16ee1ae7c	42
double k = -3.141	0x16ee1ae78	-3.141

Speicheradressen werden im Hexadezimalsystem angegeben, das den Wertebereich von 0 bis 15 umfasst:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f

Vier Bit können mit einer Hexadezimalzahl dargestellt werden:

0000 → 0

...

0100 → 4

...

1010 → A

...

1111 → F

<https://de.wikipedia.org/wiki/Hexadezimalsystem>

In C gibt es verschiedene Datentypen, die sich hinsichtlich des Speicherbedarfs unterscheiden

Name	Formelzeichen	Wertebereich	Größe (Minimum)
signed int	%d oder %i	−32.768 bis 32.767	2 Byte
signed long int	%ld oder %li	−2.147.483.648 bis 2.147.483.647	4 Byte
unsigned int	%u	0 bis 65.535	2 Byte
float	%f / %n.nf	1.2e-38 bis 3.4e38	4 Byte
double	%lf	2.3e-308 bis 1.7e308	8 Byte
char (Zeichen, Buchstaben, kleine Zahlen)	%c	0 bis 255 / -128 bis 127	1 Byte

- Bei Gleitpunktzahlen ist es nicht nur wichtig zu wissen, wie groß die Zahl werden kann, die gespeichert wird, sondern auch, wie hoch die Genauigkeit sein muss
- So haben float und double auch unterschiedliche Genauigkeiten:
 - float: 6-stellige Genauigkeit
 - double: 15-stellige Genauigkeit
- Eine float-Variable kann also insgesamt sechs Stellen unterscheiden:
 - 1234,12345 ist für den Compiler dasselbe wie
 - 1234,12399 da nur sechs Stellen unterschieden werden können, von links gezählt
- Gleitpunktzahlen sollten außerdem, aufgrund der Art wie sie gespeichert werden, nicht auf Gleichheit überprüft werden

- Der Datentyp legt fest, wieviel Speicherplatz für eine Variable reserviert wird
- Datentypen lassen sich außerdem in Ganzzahlen, Gleitpunktzahlen sowie Zeichen unterscheiden

Ganzzahl:
acht Kuchen



Gleitpunktzahl:
Kuchendurchmesser 20.3 cm

- Der Datentyp legt fest, wieviel Speicherplatz für eine Variable reserviert wird
- Datentypen lassen sich außerdem in Ganzzahlen, Gleitpunktzahlen sowie Zeigern unterteilen

Ganzzahl:
acht Kuchen



Gleitpunktzahl:
Kuchendurchmesser 20.3 cm

Warum GleitPUNKTzahl und nicht Gleitkommazahl?

C (und die meisten anderen Programmiersprachen) ist englisch – daher nicht vergessen:
Punkt für die Kommastelle:
20.3 cm
(20,3 → das Komma wird vom Compiler als Trennoperator gelesen und es wird ein Fehler ausgegeben)

- Der Datentyp character (*char*, Formelzeichen %c) wird verwendet, um einzelne Zeichen einzulesen sowie auszugeben
- Die einzelnen Zeichen werden anhand der ASCII-Tabelle kodiert

Variablen, Datentypen & Adressoperator

Character-Datentyp

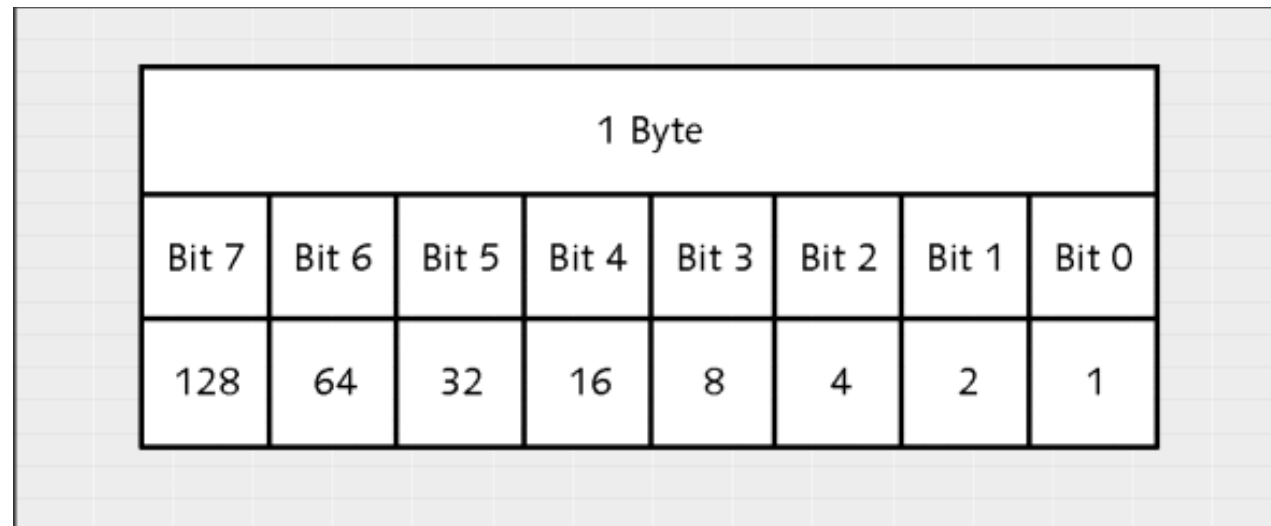
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

- Der Datentype character (*char*, Formelzeichen %c) wird verwendet, um einzelne Zeichen einzulesen sowie auszugeben
- Die einzelnen Zeichen werden anhand der ASCII-Tabelle kodiert

Nutzt man die scanf-Funktion um nacheinander Zeichen einzulesen, kann es zu unerwünschtem Verhalten kommen, da das Enterzeichen auch als Zeichen eingelesen wird und im Zwischenspeicher bleibt. Die zweite scanf-Abfrage wird daher nicht korrekt ausgeführt.

- Der Datentype character (*char*, Formelzeichen %c) wird verwendet, um einzelne Zeichen einzulesen sowie auszugeben
- Mit einer Standardgröße von 1 Byte kann er außerdem genutzt werden, um kleine Ganzzahlen zu verarbeiten (-128 ...127 bzw. 0...255) → In Fällen, in denen Speicherplatz sehr knapp ist und man sicher ist, dass die Zahl den Wertebereich nicht überschreitet.



$$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

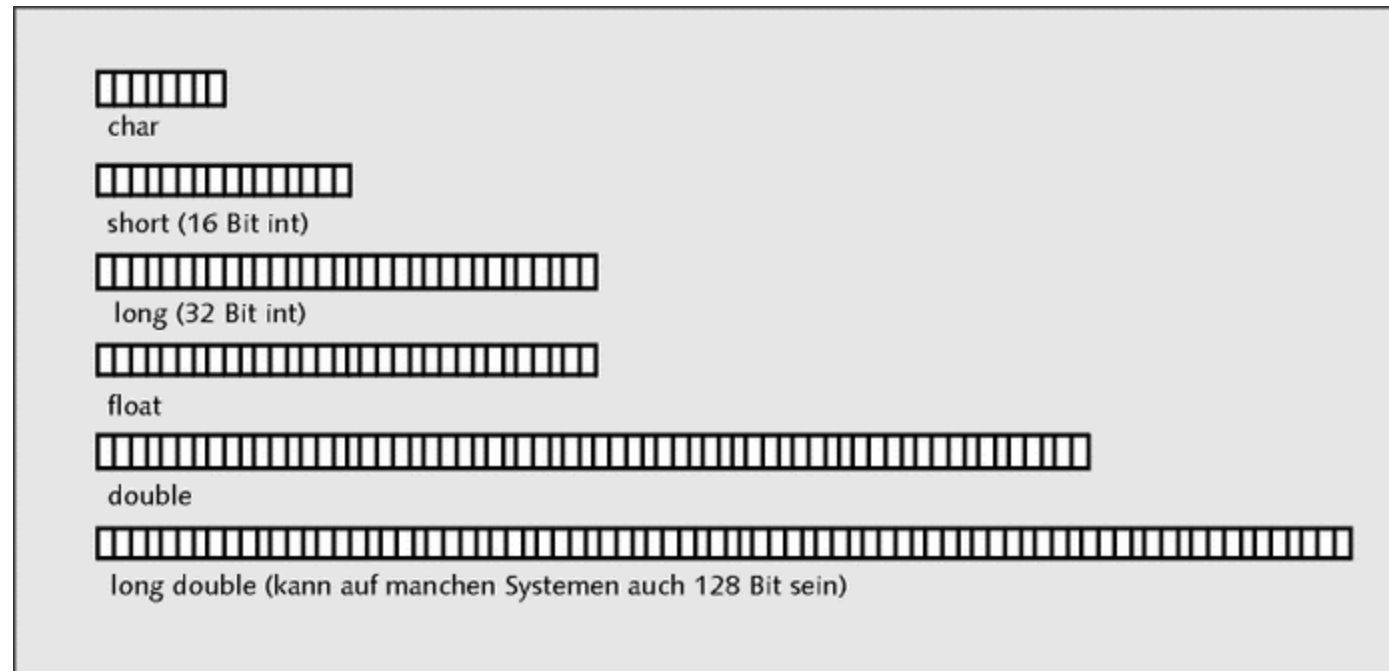
- Wie wir in der ersten Vorlesung gesehen haben, benötigt man in der Programmierung häufig boolesche Variablen
`bool = true / false bzw. 0 / ≥ 1`
- Boolesche Werte wurden erst mit dem C99-Standard eingeführt und müssen über den header `<stdbool.h>` eingebunden werden
`bool b = 0; // false`
`bool b = 1; // true`
`bool b = true;`
`bool b = false;`
`bool b = 3; // true`
- Boolesche Werte können mittels `printf("%d", b)` als Integer ausgegeben werden (implizite Typumwandlung)

- Obwohl bei der Deklaration von Variablen ein Datentyp festgelegt wird, kann es sein, dass der C-Compiler eine automatische, implizite Datentypumwandlung vornimmt
- Dies kommt vor, wenn einem Datentypen Werte eines anderen Datentypen zugewiesen werden, z.B.:

```
int x = 5, y = 2;  
float result = x / y; // Das Ergebnis ist 2.0, die Werte nach dem Komma werden bei der Typumwandlung abgeschnitten
```
- Datentypumwandlungen können auch von dem/der Programmierer:in vorgenommen werden
→ benutzerdefinierte explizite Typumwandlung

```
result = (float) x / (float) y; // Das Ergebnis ist 2.5, da x und y explizit in float-Typen umgewandelt werden und dann erst die Berechnung vorgenommen wird
```
- Es ist immer besser, Typumwandlungen explizit vorzunehmen um unerwartetes Verhalten zu vermeiden

- Der Datentyp legt fest, wieviel Speicherplatz für eine Variable reserviert wird
Insbesondere Embedded-Systeme(Mikrocontroller) haben wenig Speicherplatz, entsprechend sorgfältig werden bei der Programmierung die Datentypen ausgewählt



- Wer es ganz genau wissen will: https://openbook.rheinwerk-verlag.de/c_von_a_bis_z/005_c_basisdatentypen_001.htm#mjeda2957449ddc812dad62b400fe57752

Variablen, Datentypen & Adressoperator

Eingabe von Text über die Konsole

```
#include <stdio.h>
```

```
int main(){  
    int i;  
    printf("Please enter a \\  
           number: \n");  
    scanf("%d", &i);  
    printf("Your number is: %d\n", i)  
  
    return 0;  
}
```

```
>> ./read  
Please enter a number: 42  
Your number is: 42
```

- Die scanf-Funktion liest Text über die Standardeingabe (Tastatur) ein
- Syntax:

scanf("%d", &i);

↙ schreibt den eingelesenen Wert in die Variable *i*

↑

Formatiertes Einlesen der Eingabe
%d steht für eine Ganzzahl mit mindestens 16 Bit

- ~~Was sind Variablen?~~
- ~~Was sind Datentypen?~~
- Was bedeutet das &?

- Eine Variable besteht aus dem Datentyp, Variablenname, dem zugewiesenen Wert sowie der Speicheradresse, an der er abgelegt ist
- Die Speicheradresse weist der Compiler der Variablen bei der Definition automatisch zu

Datentyp	Name	Speicheradresse	Wert
int	i	0x123A	1337

- Das &-Zeichen ist der sogenannte Adressoperator, das heißt `&i` steht für die Adresse, an der die Variable *i* gespeichert wurde
- `scanf("%d", &i)` // speichere an der Adresse der Variablen *i* die Ganzzahl, die über die Tastatur eingelesen wird

Mehr zu Adressoperatoren später im Kurs!

```
#include <stdio.h>

int main(){
    int i;
    printf("Please enter a \
          number: \n");
    scanf("%d", &i);
    printf("Your number is: %d\n", i)

    return 0;
}
```

```
>> ./read
Please enter a number: 42
Your number is: 42
```

- Die scanf-Funktion liest Text über die Standardeingabe (Tastatur) ein
- Syntax:

scanf("%d", &i);

↙ schreibt den eingelesenen Wert in den **Speicherplatz der Variable *i***

↑

Formatiertes Einlesen der Eingabe
%d steht für eine Ganzzahl mit mindestens 16 Bit

- ~~Was sind Variablen?~~
- ~~Was sind Datentypen?~~
- ~~Was bedeutet das &?~~

- C-Programme können Text ausgeben und einlesen
- Die printf-Funktion gibt Text sowie Variablenwerte aus

```
#include <stdio.h>

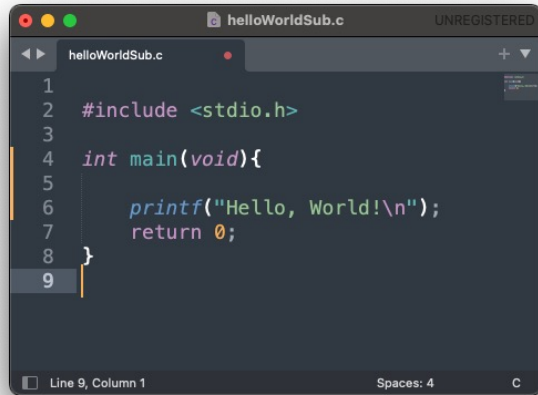
int main(){
    int i = 0, j = 1, k = 2;
    float m = 0.10100;

    printf("This outputs plain text.\n");
    printf("Let us print the first variable i: %d\n", i);
    printf("Let us print all the variables: %d, %d, and %d\n", i,j,k);
    printf("Let us calculate %d squared: %d\n", k, k*k);
    printf("Finally, a floating point: %1.2f\n", m);
    return 0;
}
```

- C-Programme können Text ausgeben und einlesen
- Die printf-Funktion gibt Text sowie Variablenwerte aus
- Die scanf-Funktion liest formatierten Text ein und speichert ihn in einer Variablen
- Obwohl printf und scanf in allen C-Lehrbüchern vorkommen, werden sie in der Praxis kaum verwendet, da sie nicht sicher sind
 - Für unsere Anwendung reichen sie aus
 - Wir werden später im Kurs Alternativen kennenlernen
- Ebenfalls später im Kurs werden wir Strings behandeln, die uns das Einlesen von mehr als einem Zeichen ermöglichen

C-Code schreiben, kompilieren und ausführen

Mit Texteditor und Konsole



```
helloWorldSub.c
1
2 #include <stdio.h>
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

Texteditor mit C-Quellcode
HelloWorld.c

compiler



```
maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1
~/V/P/C/V_1 clang helloWorldSub.c -o helloWorldSub
~/V/P/C/V_1 ./helloWorldSub
Hello World
~/V/P/C/V_1
```

Konsole

1. ruft den Compiler auf, der den C-Quellcode zu Maschinencode kompiliert

Maschinencode
HalloWorld

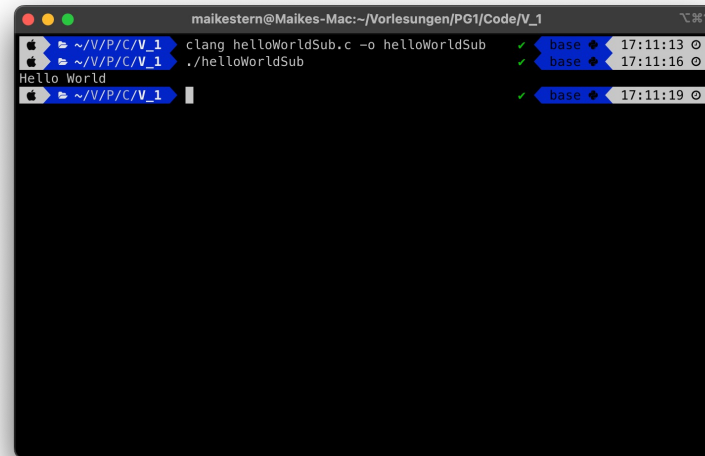
```
01101010001
11010100101
01010001011
11101010111
00011101010
01010101011
01010100101
11010100011
```

(für das Betriebssystem
ausführbare Datei)



```
helloWorldSub.c
1
2 #include <stdio.h>
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

Texteditor mit C-Quellcode
HelloWorld.c



```
maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1
~/V/P/C/V_1 clang helloWorldSub.c -o helloWorldSub
~/V/P/C/V_1 ./helloWorldSub
Hello World
~/V/P/C/V_1
```

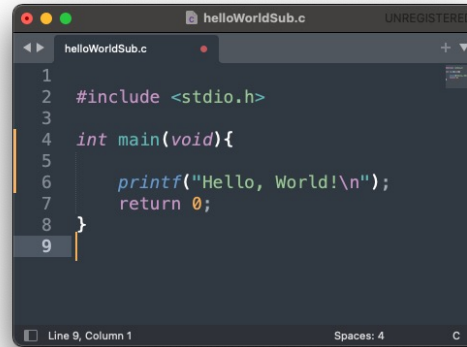
Konsole

1. ruft den Compiler auf, der den C-Quellcode zu Maschinencode kompiliert
2. führt den Maschinencode aus und schreibt „Hello World“ in die Konsole

Maschinencode
HalloWorld

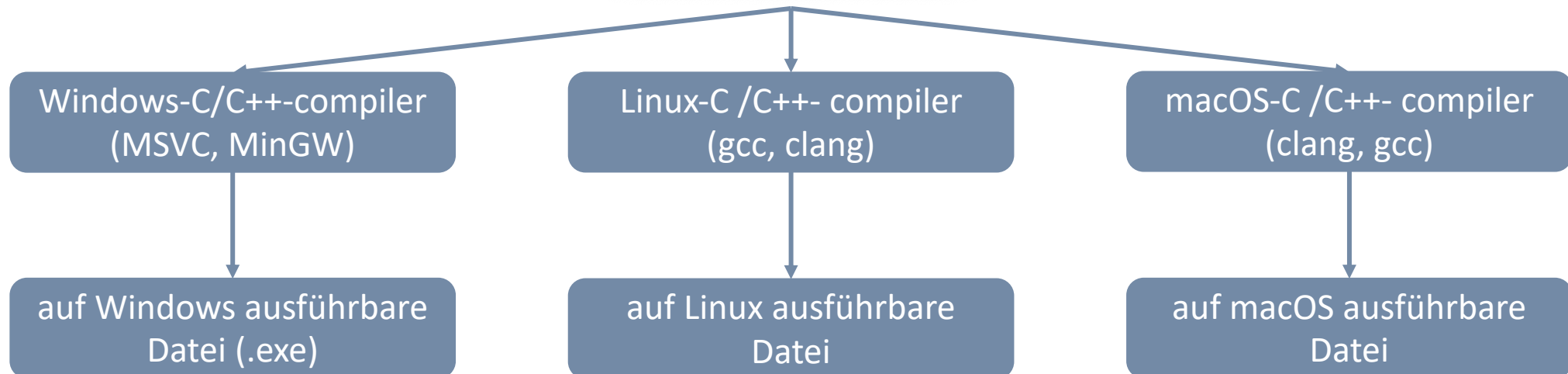
01101010001
11010100101
01010001011
11101010111
00011101010
01010101011
01010100101
11010100011

(für das Betriebssystem
ausführbare Datei)



```
1 #include <stdio.h>
2
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

C-Quellcode muss für
jede
Plattform/Architektur
eigens kompiliert
werden



- Texteditor
Notepad, Sublime, Atom, oder gleich in der Konsole: VIM
- C-Compiler (abhängig vom Betriebssystem)
MSVC, MinGW (Windows), clang, gcc (MacOS & Linux)




A screenshot of a text editor window titled 'helloWorldSub.c'. The code is as follows:

```
1
2 #include <stdio.h>
3
4 int main(void){
5
6     printf("Hello, World!\n");
7     return 0;
8 }
9
```

The status bar at the bottom indicates 'Line 9, Column 1' and 'Spaces: 4'.

Textdatei mit dem Programm



A screenshot of a terminal window titled 'maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1'. The terminal shows the following commands and output:

```
clang helloWorldSub.c -o helloWorldSub
./helloWorldSub
Hello World
```

The terminal also shows the prompt 'maikestern@Maikes-Mac:~/Vorlesungen/PG1/Code/V_1' and a cursor.

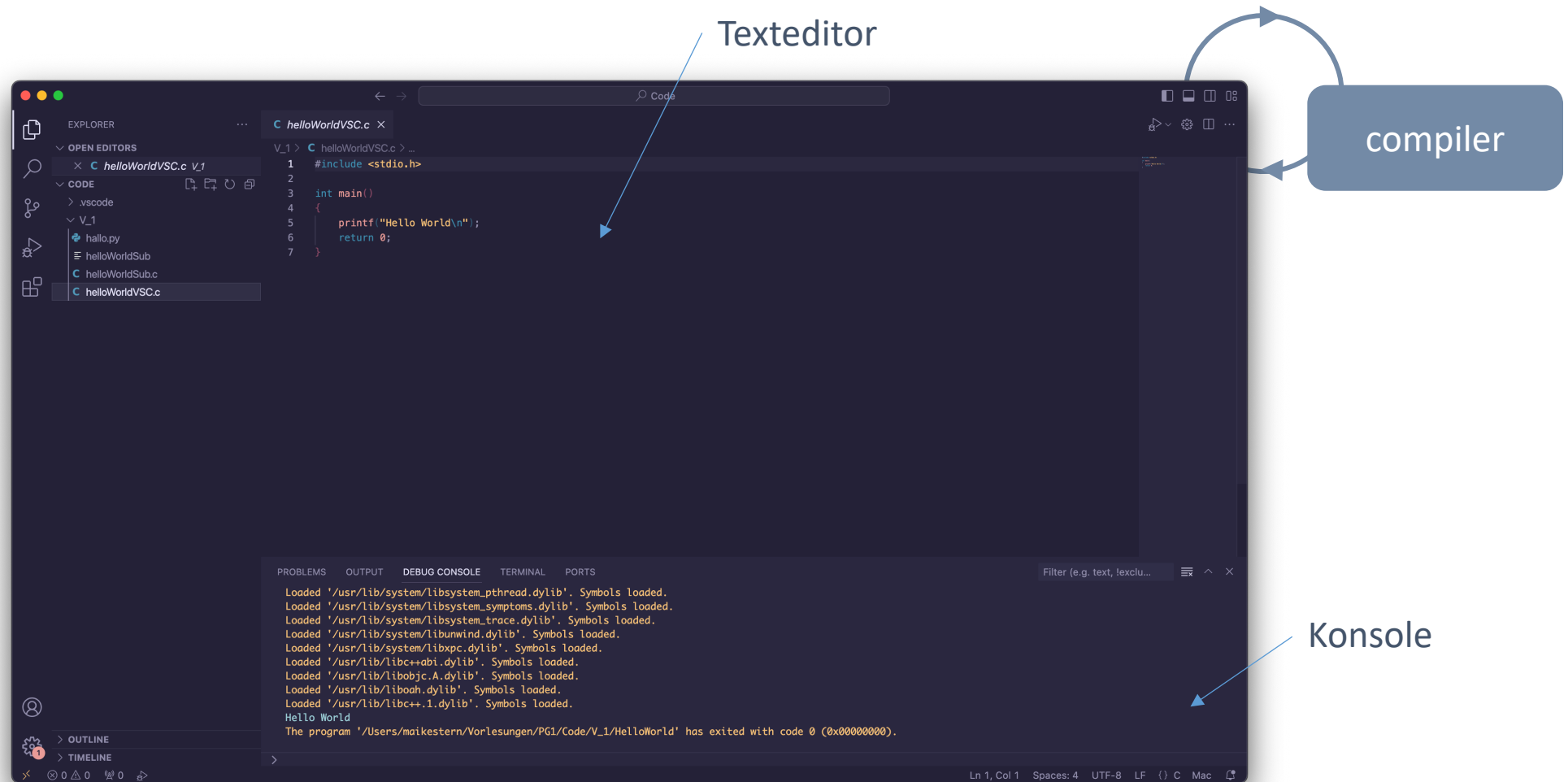
Konsole

C-Code schreiben, kompilieren und ausführen

Mit integrierter Entwicklungsumgebung (IDE)

Entwicklungsumgebungen

IDE - Visual Studio Code

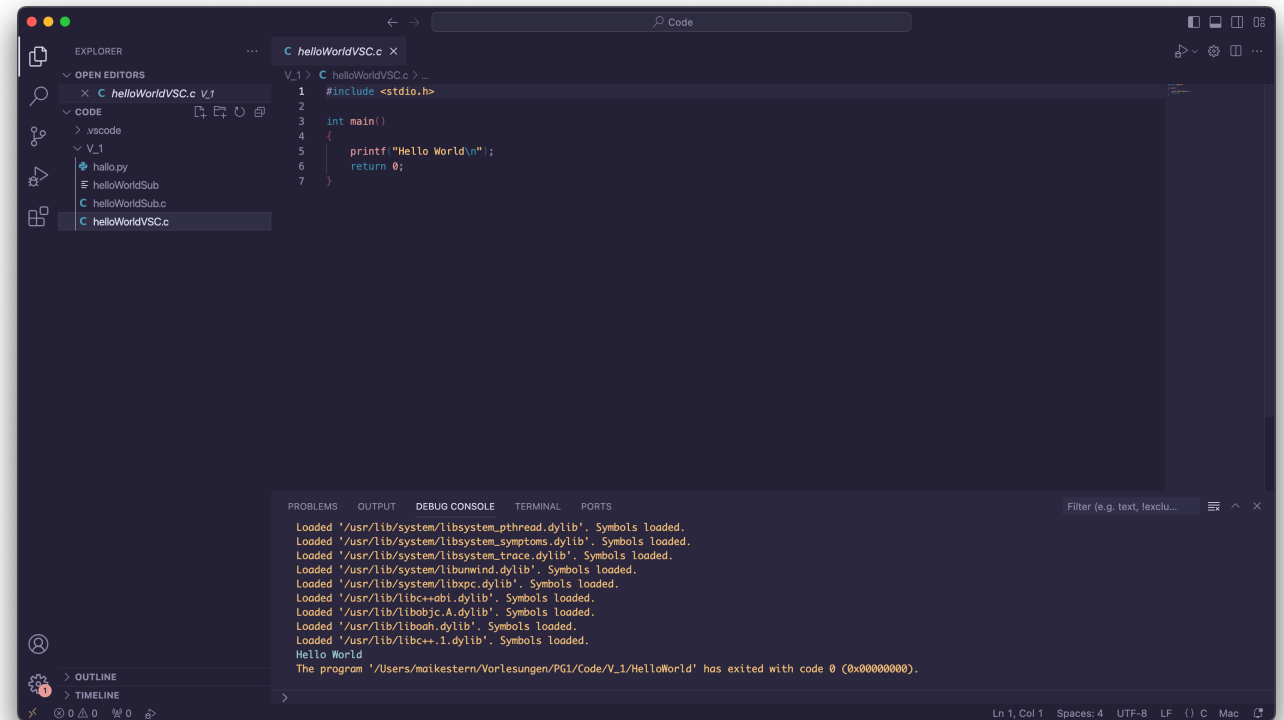


Vorteil IDE:

- Alles an einem Ort
- Sehr viele Erweiterungen erhältlich
- Wird auch von Programmierer:innen anderer Programmiersprachen verwendet, z.B. Java und Python
- Debugging

IDE-Optionen

- CLion, Code::Blocks, Visual Studio Code, Visual Studio



Windows:

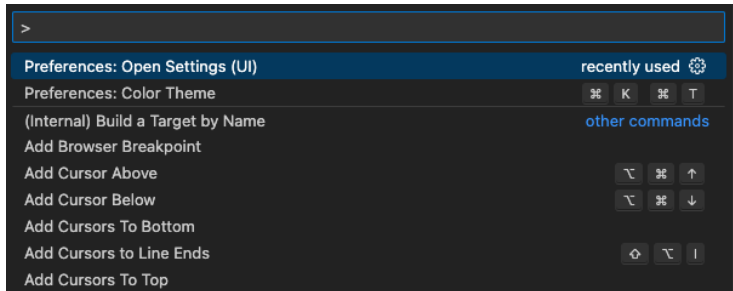
1. Compiler installieren
2. Visual Studio Code installieren und den Anweisungen für C folgen
3. VSCode-Extensions installieren:
 - C/C++ von Microsoft
 - Code Runner

MacBook

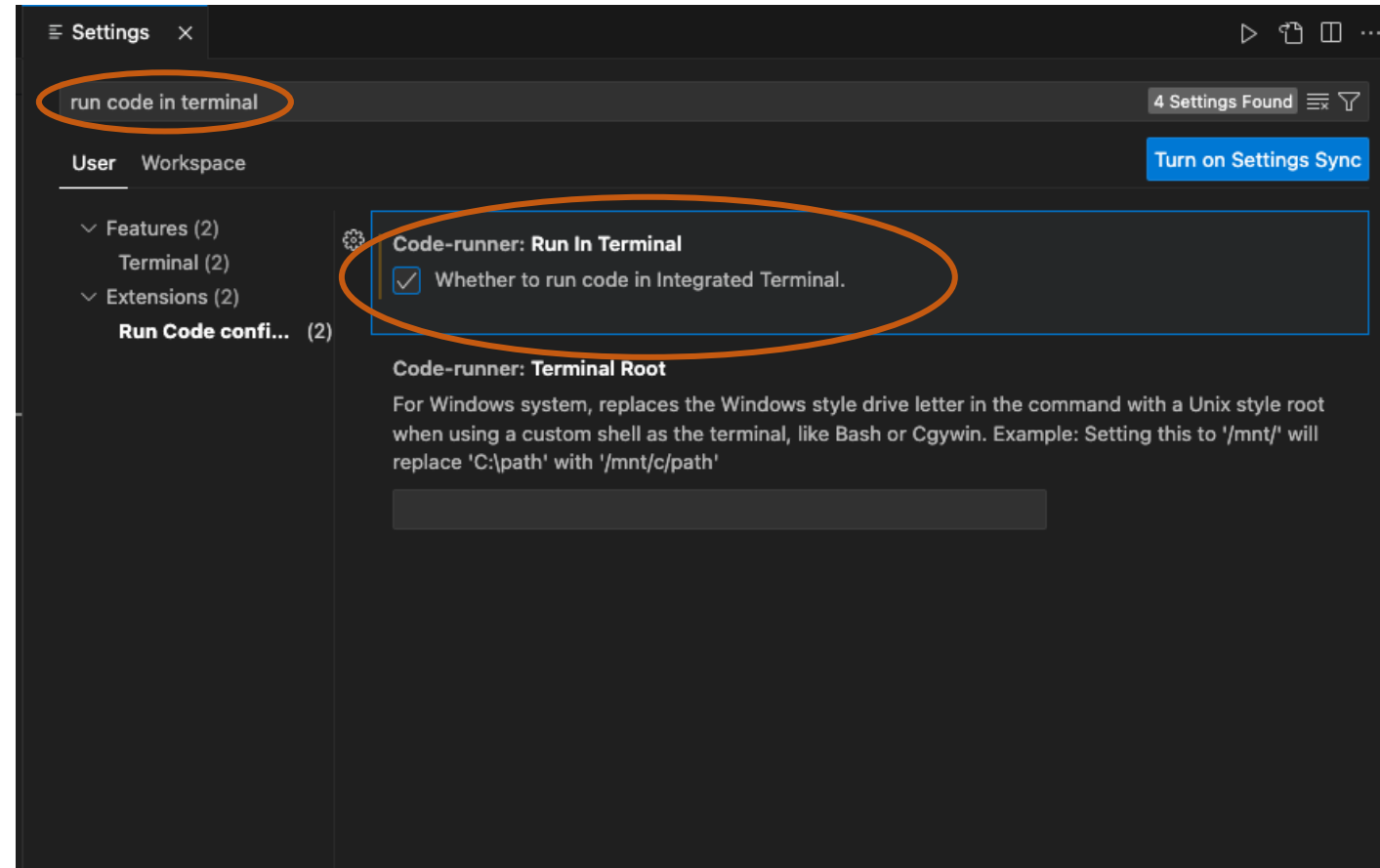
1. Xcode installieren (installiert unter anderem den Clang-Compiler)
2. Visual Studio Code installieren und den Anweisungen für C folgen
3. VSCode-Extensions installieren:
 - C/C++ von Microsoft
 - Code Runner

Code Runner

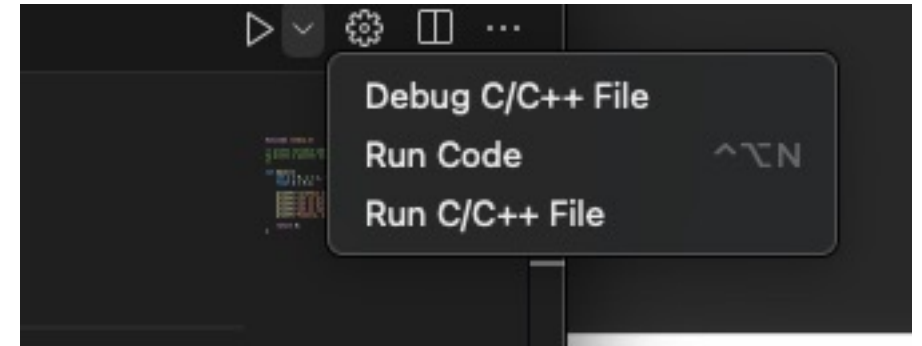
- Extension herunterladen
- Umschalt + Strg + P öffnet die Steuerungszeile
- In die Steuerungszeile "Settings" eingeben



- In den Settings nach "run code in terminal" suchen und Run in Terminal auswählen



- Code schreiben
- Code speichern (mit Dateiendung .c)
- Code kompilieren und linken (run C/C++ File)
 - Compilerwarnungen fixen
- Code im Terminal ausführen und testen
 - Bugs fixen



- Die typische C-Syntax beinhaltet die main-Funktion, die der Startpunkt jedes Programmes ist
`int main(){}`
- Die printf-Funktion ermöglicht es, Text auf die Konsole zu schreiben
`printf("Hallo, World!");`
- Die scanf-Funktion ermöglicht es, Zahlen und Zeichen von der Konsole einzulesen (und später werden wir auch Zeichenketten einlesen)
`scanf("%f", &f);`
- Variablen können Ganzzahlen, Gleitpunktzahlen oder Zeichen speichern
`int i = 1337;`
- Datentypen legen fest, wieviel Speicherplatz reserviert wird für eine Variable
- Es gibt verschiedene Möglichkeiten C-Code zu entwickeln

- Funktionen

VSCode Einstellungen

Die stdbool.h-Headerdatei wird nicht immer automatisch von IntelliSense gefunden (vom Compiler aber schon)

1. stdbool.h-Datei-Systempfad suchen
2. Pfad im c_cpp_properties.json unter configurations > includePath

```
#include <stdbool.h>
Die Datei Quelle kann nicht geöffnet werden: "stdbool.h".. Führen Sie den
Befehl „IntelliSense-Konfiguration auswählen...“ aus, um nach Ihren
Systemheadern zu suchen. C/C++(1696)
View Problem (⌘F8) Quick Fix... (⌘.)
```

```
.vscode > {} c_cpp_properties.json > [ ] configurations > {} 0 > [ ] includePath > 1
1  {
2      "configurations": [
3          {
4              "name": "Mac",
5              "includePath": [
6                  "${workspaceFolder}/**",
7                  "/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.s
8              ],
9              "defines": [],
10             "macFrameworkPath": [
11                 "/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/S
12             ],
13             "cStandard": "c17",
14             "cppStandard": "c++17",
15             "intelliSenseMode": "macos-clang-arm64",
16             "compilerPath": "/usr/bin/clang"
17         }
18     ],
19     "version": 4
20 }
```

Ausgeben aller Warnungen beim Kompilieren & Linken:

Sowohl beim reinen Kompilieren / Debuggen als auch beim Kompilieren + Ausführen in der VSC-Konsole mittels Code Runner können alle Compiler-Warnungen ausgegeben werden. Das ist hilfreich, weil man so lernt was der Compiler als unsauber erachtet.

Dabei gibt es zwei Argumente:

- Wall: Alle Compiler-Warnungen werden auf der Konsole ausgegeben
- Werror: Alle Warnungen werden als Fehler behandelt → Warnungen beim kompilieren führen dazu, dass der Code nicht kompiliert wird. Das ist insbesondere sinnvoll, wenn man "richtigen" Produktionscode schreibt aber auch, um sich beim Programmieren einen guten Stil anzueignen. Beim Ausprobieren kann es aber stören, daher ist -Werror bei mir nicht gesetzt.

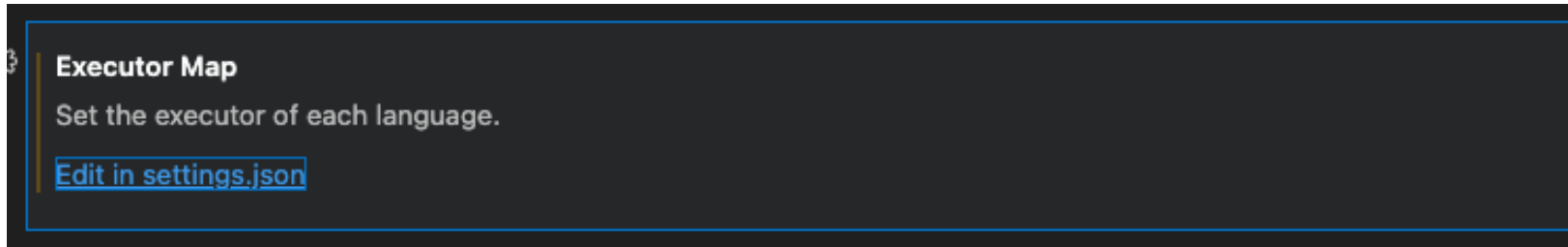
Warnungen ausgeben: Compiler (kompiliert den Code)

- Um den eigentlichen Compiler anzupassen muss die Datei tasks.json angepasst werden
- Strg + Umschalt + P > Tasks: Configure Default Build Task > (immer noch in der Steuerleiste) C/C++ Aktive Datei kompilieren (öffnet tasks.json)
- -Wall: Alle Warnung ausgeben
- -Werror: Alle Warnungen als Fehler behandeln

```
.vscode > {} tasks.json > [ ] tasks > {} 0
1  {
2      "tasks": [
3          {
4              "type": "cppbuild",
5              "label": "C/C++: clang Aktive Datei kompilieren",
6              "command": "/usr/bin/clang",
7              "args": [
8                  "-fcolor-diagnostics",
9                  "-fansi-escape-codes",
10                 "-Wall",
11                 "-g",
12                 "${file}",
13                 "-o",
14                 "${fileDirname}/${fileBasenameNoExtension}.o",
15             ],
16             "options": {
17                 "cwd": "${fileDirname}"
18             },
19             "problemMatcher": [
20                 "$gcc"
21             ],
22             "group": {
23                 "kind": "build",
24                 "isDefault": true
25             },
26             "detail": "Vom Debugger generierte Aufgabe."
27         },
28     ],
29     "version": "2.0.0"
30 }
```

Warnungen ausgeben: Code Runner (kompiliert und führt den Code in der Konsole aus)

- Strg + Umschalt + P: Settings
- In den Settings nach "Executor Map" suchen und "Edit in settings.json" auswählen



- -Wall: Alle Warnungen beim Kompilieren anzeigen
- -Werror: Alle Warnungen als Fehler behandeln

```
"code-runner.executorMap": {  
  "c": "cd $dir && clang $fileName -Wall -Werror -o $fileNameWithoutExt && $dir$fileNameWithoutExt",  
}
```

- Zusätzlich kann der Compiler angepasst werden (gcc, clang, msvc)