

C introduction

## Basic program structure

# Contents

# Getting started

The slides are at [fsrc.github.io/c-lessons/materials.html](https://fsrc.github.io/c-lessons/materials.html)

There will be tasks! You can find them at [fsrc.github.io/c-lessons](https://fsrc.github.io/c-lessons)

If you have questions, use the auditorium group:  
<https://auditorium.inf.tu-dresden.de/de/groups/110804109>

In case of big trouble, write an e-mail to your tutor.

# Programs for programming

For developing our own programs, we will use two kinds of software:

Plain text editors:

- ▶ vim (powerful, but needs getting used to)
- ▶ gedit (easy to use, comes with Gnome)
- ▶ Notepad(++) (for Windows users)

Compilers:

- ▶ gcc
- ▶ clang

## gcc for Unix-based operating systems

You maybe have to update your system before the installation !

Ubuntu / Debian:

```
$ sudo apt-get update  
$ sudo apt-get install gcc
```

Arch Linux:

```
$ sudo pacman -Syu  
$ sudo pacman -S gcc
```

Mac OS X:

```
$ brew install gcc
```

... and you're done ;-)

## gcc for Windows 10 (using *bash*)

For convenience you should use the new Ubuntu-based Linux subsystem.

- ▶ In *Settings*, got to *Update & Security > For Developers* and switch to *Developer Mode*
- ▶ In the *Control panel*, go to *Programs > Turn Windows Features On or Off* and enable the *Windows Subsystem for Linux (Beta)*
- ▶ Reboot as you're prompted
- ▶ Go to the Microsoft store and search for Ubuntu and install the app
- ▶ Search for “*bash*” and run the *bash* command
- ▶ Follow the installation instructions

You may now continue as if you were using Ubuntu ;-)

## gcc for older versions of Windows (using cygwin)

We highly recommend using the [Chocolatey](#) package manager.

- Follow the link and the instructions on the website to install it

Install cygwin (launch the shell as an administrator!):

```
$ choco install cygwin  
$ choco install cyg-get
```

Install gcc and optionally vim: ([How to fix the "Cannot find file" error](#))

```
$ cyg-get gcc-core vim
```

Now start *cygwin* to get a command line similar to Unix.

# The first program

- ▶ Create a new file named “main.c”.
- ▶ Open it in your text editor of choice.
- ▶ Fill it as follows:

```
1 #include <stdio.h>
2
3 int main(void) {
4     printf("Hello World!\n");
5     /* Print "Hello World!" on the
6        command line */
7     return 0;
8 }
```



# From source to binary

Source code



```
$ gcc main.c
```

(Preprocessing, compiling, assembling, linking)



Executable program

Linux/Mac OS X (a.out)

```
$ ./a.out
```

Windows (a.exe)

```
$ ./a.exe
```

# A basic program

```
1 #include <stdio.h>
2
3 int main(void) {
4
5     printf(" Hello World!\n");
6     /* Print "Hello World!" on the
7        command line */
8
9     return 0;
10 }
```

} Preprocessor statements

} Main function

# Preprocessor statements

- ▶ Processed before compilation
- ▶ Have their own language; start with a `#`

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

- ▶ Includes the *input/output header* from the C standard library
- ▶ Needed to use *printf()*

Preprocessor statements have way more use cases, but they form their own language which is very different from actual C.

In this course, we will use them for inclusions only.

# The main function

- ▶ Core function of every program
- ▶ Exists exactly once in every program
- ▶ Called on program start

```
3 int main(void) {
```

- ▶ As a function, *main()* can take parameters and return a value
- ▶ Get used to *void* and *int*. They will be explained later
- ▶ '{' marks the start of the main function scope

# The main function scope

- ▶ Contains program statements
- ▶ They are processed from top to bottom

```
9   return 0;  
10 }
```

- ▶ Last statement; ends main function (and thus the whole program)
- ▶ `0` tells the OS that everything went right
- ▶ `'}'` marks the end of the main function scope

# Statements

- ▶ Instructions for the computer
- ▶ End with a ; (semicolon)

5

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

- ▶ Here is the empty statement:

```
;
```

- ▶ All statements are located in function blocks

# Comments

- ▶ Information for you and others who use your code
- ▶ Cut out before compilation

Single-line comments:

```
6 // Prints "Hello World!" on the command line
```

Block comments (multi-line):

```
6 /* Prints "Hello World!"  
7    on the command line */
```

Better style of block comments:

```
6 /*  
7  * Prints "Hello World!"  
8  * on the command line  
9  */
```

## A few words on style

- ▶ There can be multiple statements on one line
- ▶ Indentation is not necessary at all



# Write enjoyable code

- ▶ Put each statement onto its own line
- ▶ Indent every statement in the main function by one *tab* or a fixed number of *spaces*
- ▶ Decide on a commenting style and stick to it (`/* .. */` recommended)
- ▶ Leave blank lines between different parts of the program
- ▶ Use *spaces* and *newlines* consistently

```
1 int _main( void ) _{  
2     _printf( " Hello _World!" );  
3     _/*_ Prints _" Hello _World!" _*/  
4     _return _0;  
5 }
```