

# INF1004 procedural programming in C

DHBW Stuttgart  
Christian Holz  
[christian.holz@lehre.dhbw-stuttgart.de](mailto:christian.holz@lehre.dhbw-stuttgart.de)

## Lecture 03

### PART I


- Recap of the previous contents
- Variadic functions
- Debugging

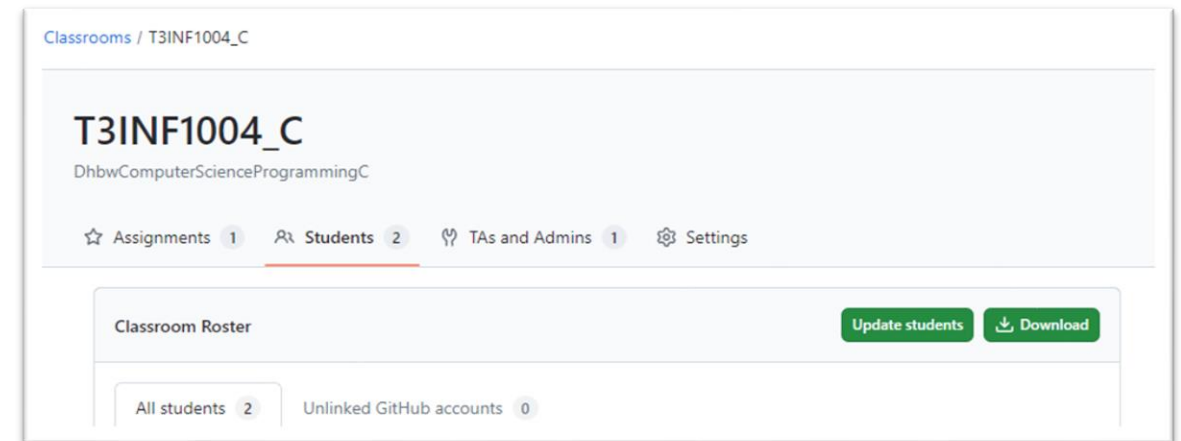
### PART II

- Transformation
- Coding project
- GitHub assignment 03

## Recap of the previous contents

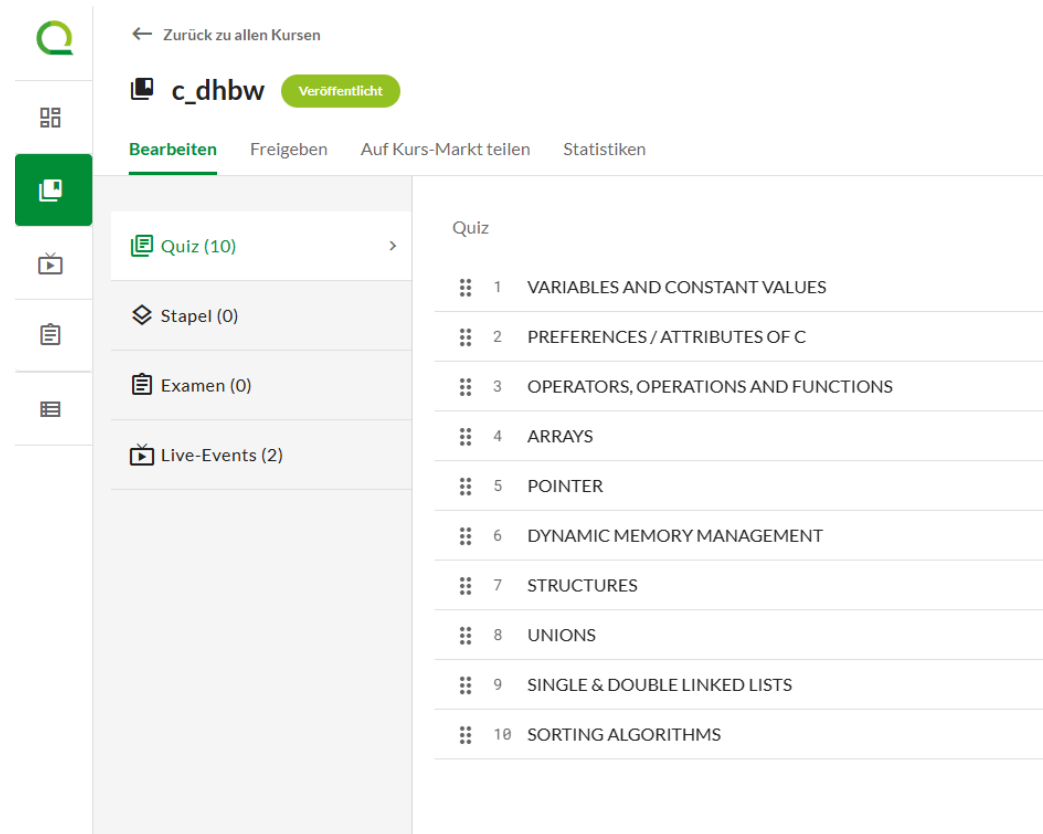
## Your Classroom for C coding Assignments

- Let's come together in the  Classroom
- **Assignment Q&A**



[https://classroom.github.com/classrooms/182848101-t3inf1004\\_c/roster](https://classroom.github.com/classrooms/182848101-t3inf1004_c/roster)

# Let's play



The screenshot shows the course management interface for 'c\_dhbw'. The left sidebar contains navigation icons: a green circle with a white 'Q', a grid icon, a green square with a white document icon (highlighted), a TV icon, a clipboard icon, and a list icon. The main content area has a top bar with a back arrow and 'Zurück zu allen Kursen', the course name 'c\_dhbw' with a 'Veröffentlicht' badge, and tabs for 'Bearbeiten', 'Freigeben', 'Auf Kurs-Markt teilen', and 'Statistiken'. Below the tabs is a list of course items: 'Quiz (10)', 'Stapel (0)', 'Examen (0)', and 'Live-Events (2)'. The 'Quiz (10)' item is selected, showing a list of 10 topics: 1. VARIABLES AND CONSTANT VALUES, 2. PREFERENCES / ATTRIBUTES OF C, 3. OPERATORS, OPERATIONS AND FUNCTIONS, 4. ARRAYS, 5. POINTER, 6. DYNAMIC MEMORY MANAGEMENT, 7. STRUCTURES, 8. UNIONS, 9. SINGLE & DOUBLE LINKED LISTS, and 10. SORTING ALGORITHMS.

← Zurück zu allen Kursen

**c\_dhbw** Veröffentlicht

Bearbeiten Freigeben Auf Kurs-Markt teilen Statistiken

Quiz (10) >

Stapel (0)

Examen (0)

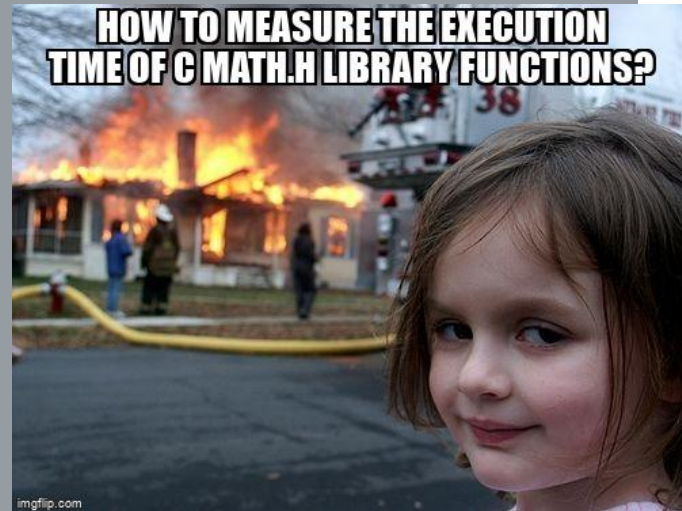
Live-Events (2)

Quiz

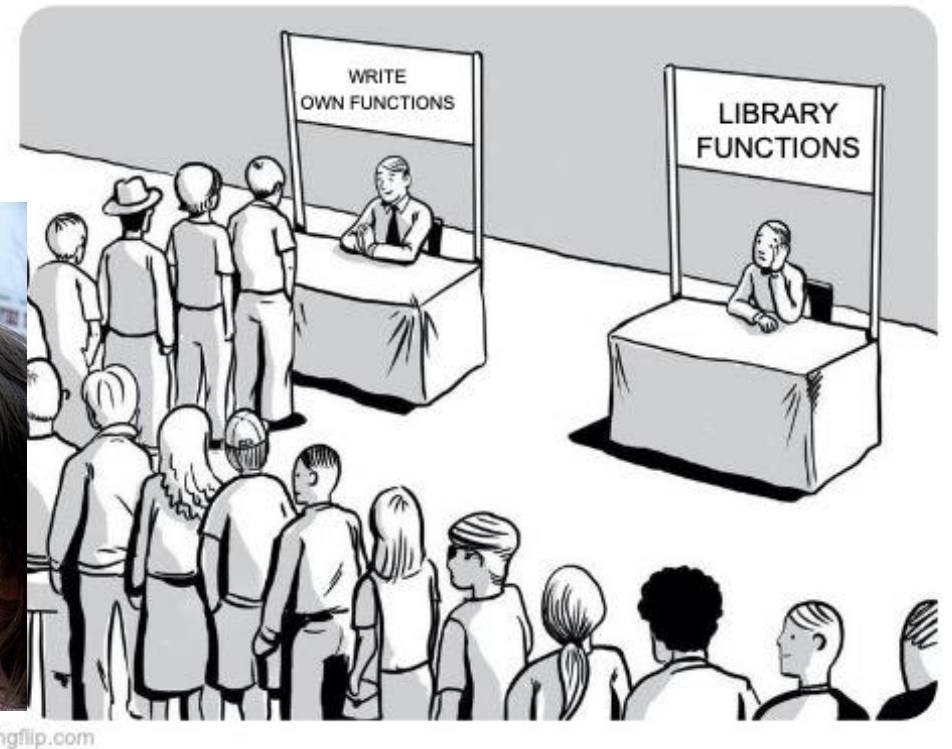
- 1 VARIABLES AND CONSTANT VALUES
- 2 PREFERENCES / ATTRIBUTES OF C
- 3 OPERATORS, OPERATIONS AND FUNCTIONS
- 4 ARRAYS
- 5 POINTER
- 6 DYNAMIC MEMORY MANAGEMENT
- 7 STRUCTURES
- 8 UNIONS
- 9 SINGLE & DOUBLE LINKED LISTS
- 10 SORTING ALGORITHMS

# Variadic functions

What is a Function in Programming?  
How do we code functions at the moment?



## C PROGRAMMERS



# Variadic functions

## Intro

Variadic functions are functions in C that accept a variable number of arguments.

- A well-known example of a variadic function in C is the `printf()` function, which accepts a variable number of arguments and outputs them formatted accordingly.
- Variadic functions are implemented using the standard library functions `stdarg.h`.

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

void simple_printf(const char* fmt, ...)
{
    va_list args;
    va_start(args, fmt);

    while (*fmt != '\0') {
        if (*fmt == 'd') {
            int i = va_arg(args, int);
            printf("%d\n", i);
        } else if (*fmt == 'c') {
            // A 'char' variable will be promoted to 'int'
            // A character literal in C is already 'int' by itself
            int c = va_arg(args, int);
            printf("%c\n", c);
        } else if (*fmt == 'f') {
            double d = va_arg(args, double);
            printf("%f\n", d);
        }
        ++fmt;
    }

    va_end(args);
}

int main(void)
{
    simple_printf("dcff", 3, 'a', 1.999, 42.5);
}
```

<https://en.cppreference.com/w/c/variadic>

# Variadic functions

## Intro

```
va_list ap;
```

`va_list` is a pointer data type. The variable of this data type references the elements of the variable part of the parameter list. The pointer `ap` must be initialised using the macro `va_start`.

```
void va_start (va_list ap, lastarg);
```

`va_start` is the macro to initialise the point `ap`. `lastarg` describes the last parameter before the variable part (“...”).

<https://en.cppreference.com/w/c/variadic>



# Variadic functions

## Intro

```
type va_arg (va_list ap, type) ;
```

To iterate across all parameter, every call of the macro `va_arg` returns an argument. `type` describes the type of the parameter that `va_arg` has to read.

```
void va_end (va_list ap) ;
```

Before to leaving the function the action of reading the variable parameter list must be closed with `va_end`.

<https://en.cppreference.com/w/c/variadic>

## Let's code

```
03_coding_exercises > lec_Exercises > a1_hello_world > C hello_world.c > ...  
1  
2  
3  
4  
5 ////////////////////////////////////////////////// YOUR CODE HERE ///////////////////////////////////  
6  
7  
8  
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

- \* Executing task: C:/Windows/System32/cmd.exe /d /c gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2 ding\_exercises\lec\_Exercises\a1\_hello\_world\hello\_world.c -o .\build\Debug\hello\_world.o && gcc -Wall e -g3 -O0 .\build\Debug\hello\_world.o -o .\build\Debug\outDebug.exe
- \* Terminal will be reused by tasks, press any key to close it.
- \* Executing task: C:/Windows/System32/cmd.exe /d /c .\build\Debug\outDebug.exe

hello world!

- \* Terminal will be reused by tasks, press any key to close it.

# Debugging

What is your way to go for debugging?



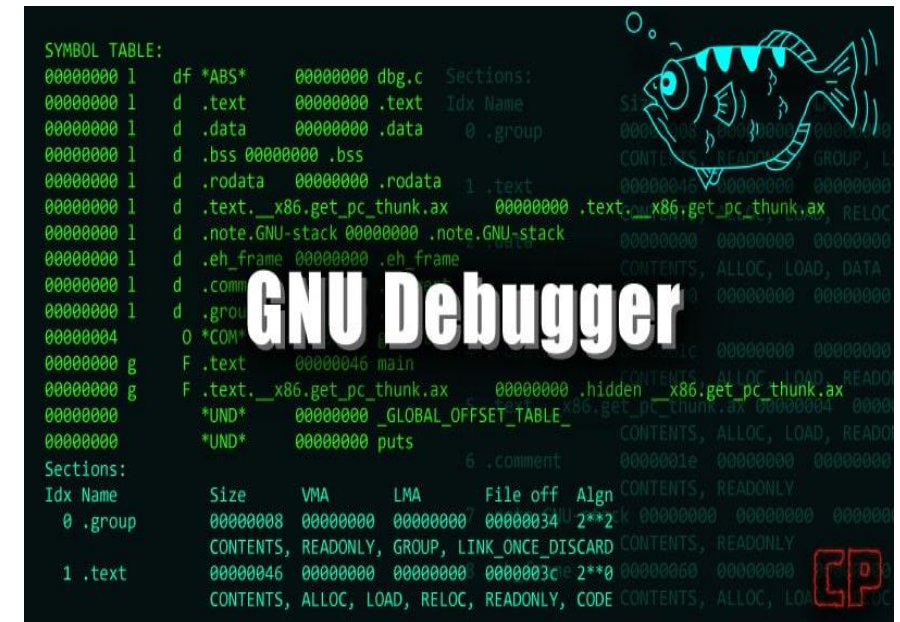
# Debugging

Find and rectify errors

Debugging is the process of finding, analysing and fixing errors (bugs) in code.

Aim of debugging is

- To ensure that the code works correctly.
- To gain an understanding of the programme flow.
- Prevent crashes and misbehaviour.



<https://www.cyberpunk.rs/gnu-debugger-tutorial-gdb-walkthrough>

## Debugging Methods

Manual debugging:

- Output with `printf()` or `log()`
- Fast, but often confusing

Debugging tools:

- Visual Studio Code, GDB
- Breakpoints, variable monitoring, call stack

Code analysis tools:

- Static analysis (e.g. compiler warnings)
- Automatic error detection



<https://programmerhumor.io/debugging-memes/debugging-story/>

## Debugging Tools

e.g. Visual Studio Code, GDB

- Set breakpoints:  
Stop the code at specific points to check values
- Step-Into/Step-Over:  
Step through individual lines of code or function calls
- Variable monitoring:  
Check the current value of variables
- Analyse call stack:  
Track which functions were called and how

### Debug C++ in Visual Studio Code Edit

After you have set up the basics of your debugging environment as specified in the configuration tutorials for each target compiler/platform, you can learn more details about debugging C/C++ in this section.

Visual Studio Code supports the following debuggers for C/C++ depending on the operating system you are using:

- Linux: GDB
- macOS: LLDB or GDB
- Windows: the Visual Studio Windows Debugger or GDB (using Cygwin or MinGW)

### Windows debugging with GDB

You can debug Windows applications created using Cygwin or MinGW by using VS Code. To use Cygwin or MinGW debugging features, the debugger path must be set manually in the launch configuration (`launch.json`). To debug your Cygwin or MinGW application, add the `miDebuggerPath` property and set its value to the location of the corresponding `gdb.exe` for your Cygwin or MinGW environment.

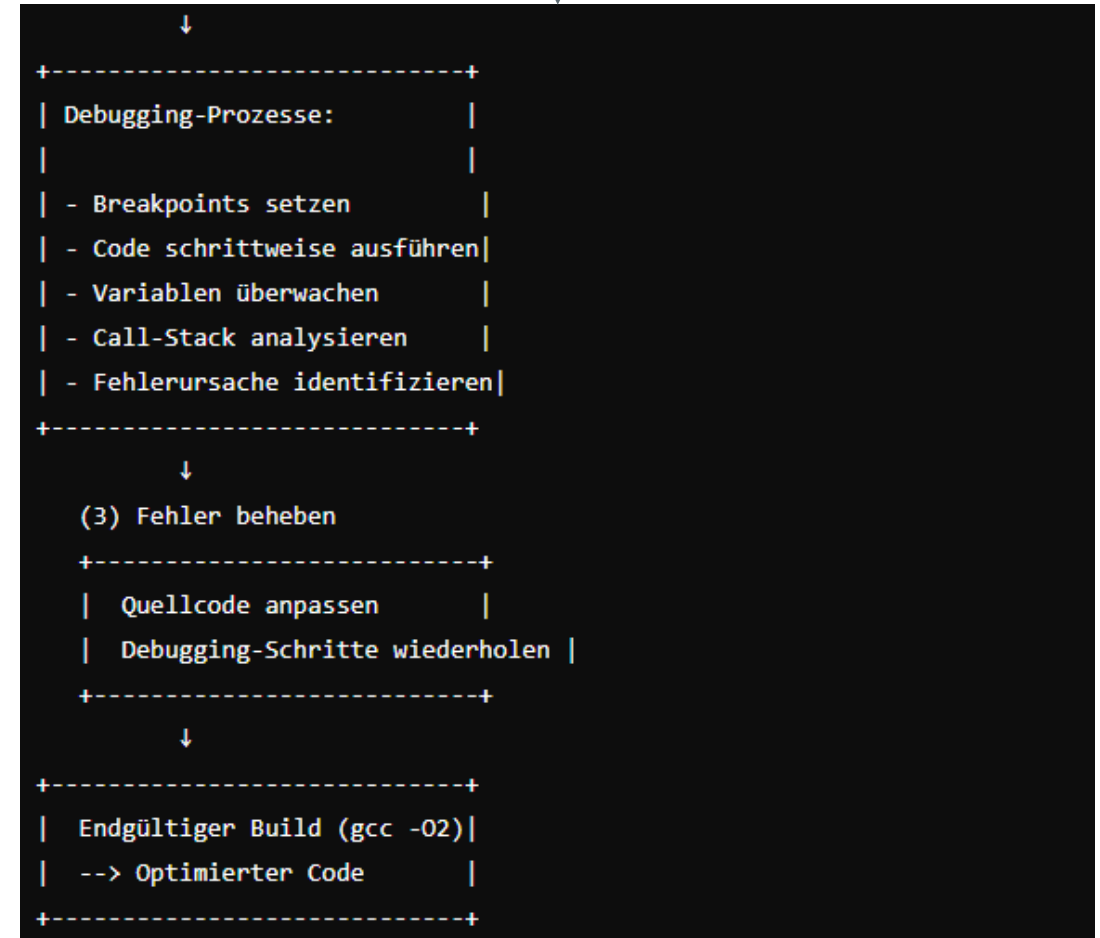
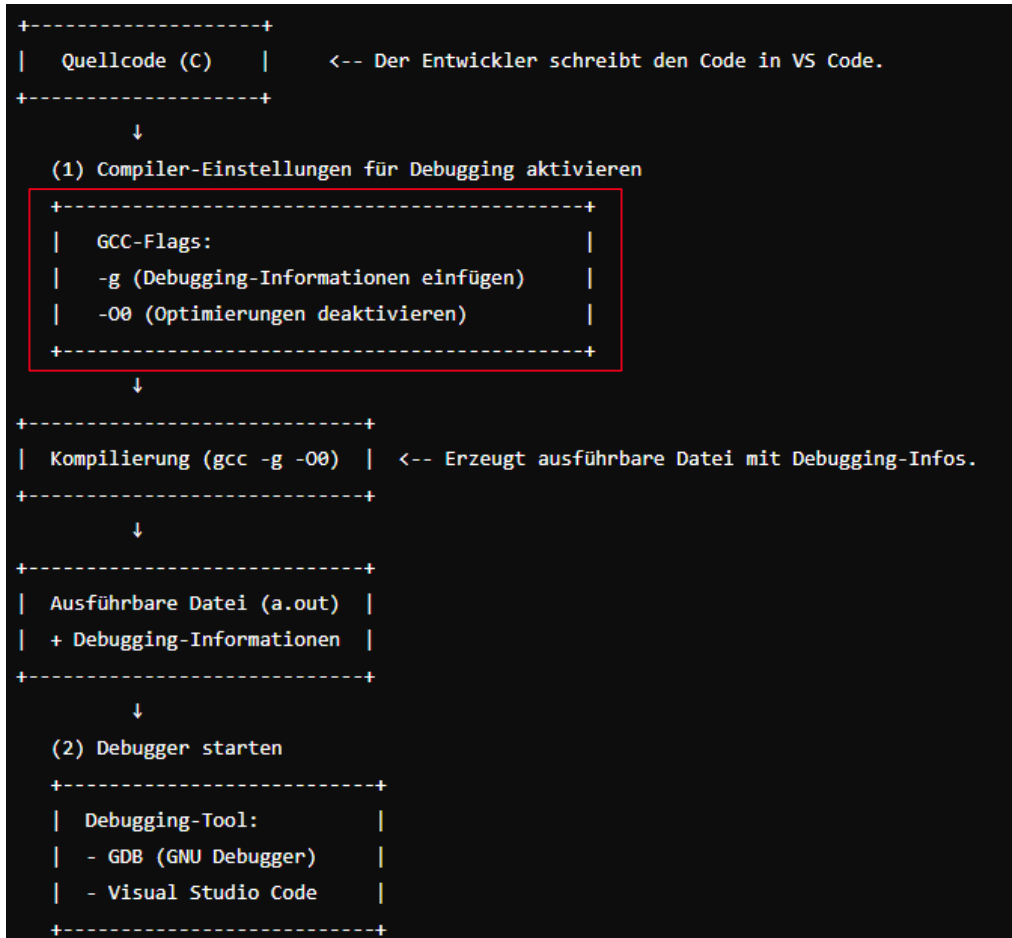
For example:

```
"miDebuggerPath": "c:\\mingw\\bin\\gdb.exe"
```

Copy

<https://code.visualstudio.com/docs/cpp/cpp-debug>

# Debugging Tools



# Debugging

## VS Code Overview

### VARIABLES

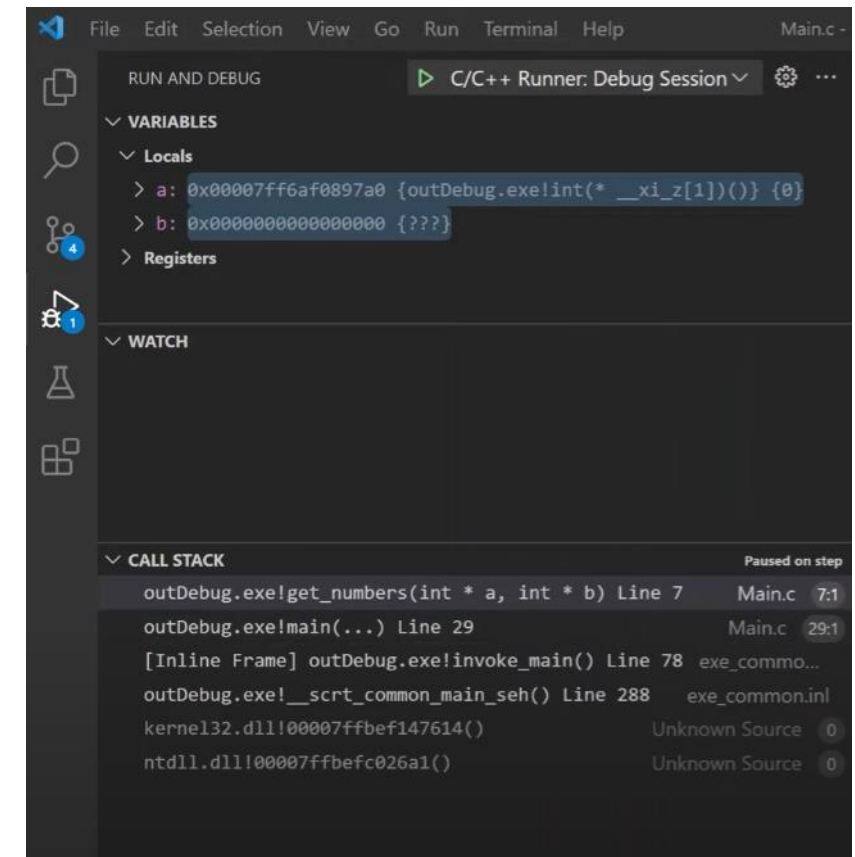
- Shows the current value of the variables in the code during programme execution
- Enables the states of variables to be monitored during runtime

### WATCH

- Allows certain variables or expressions to be added manually for monitoring
- Can also contain complex expressions or calculations to check their results

### CALL STACK

- Returns an overview of the sequence of function calls
- Shows the current function (top level) and the previous calls in chronological order
- At the bottom you may see the OS call





## Let's code

```
03_coding_exercises > lec_Exercises > a1_hello_world > C hello_world.c > ...  
1  
2  
3  
4  
5 ////////////////////////////////////////////////// YOUR CODE HERE ///////////////////////////////////  
6  
7  
8  
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

- \* Executing task: C:/Windows/System32/cmd.exe /d /c gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2 ding\_exercises\lec\_Exercises\a1\_hello\_world\hello\_world.c -o .\build\Debug\hello\_world.o && gcc -Wall e -g3 -O0 .\build\Debug\hello\_world.o -o .\build\Debug\outDebug.exe
- \* Terminal will be reused by tasks, press any key to close it.
- \* Executing task: C:/Windows/System32/cmd.exe /d /c .\build\Debug\outDebug.exe

hello world!

- \* Terminal will be reused by tasks, press any key to close it.

# Tooling for your exam

## VS Code and Extensions

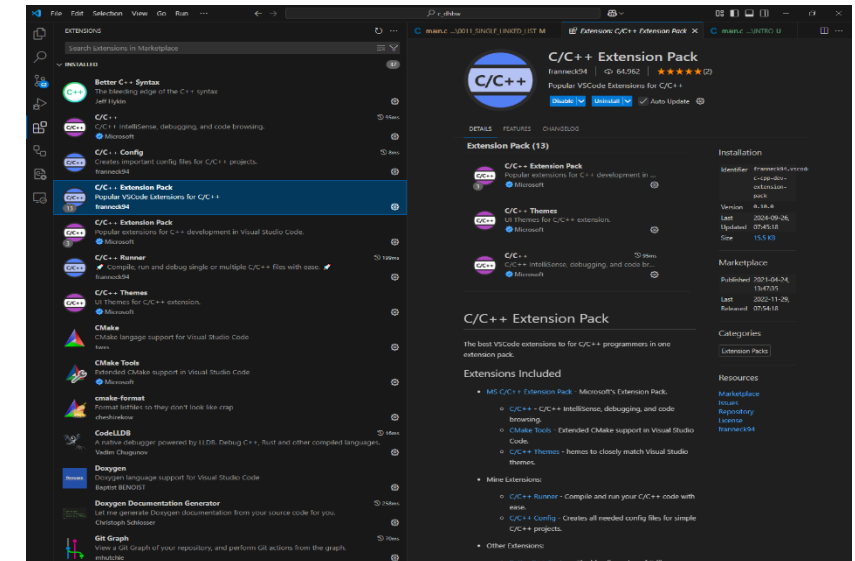
Machine:

- OS: Ubuntu
- IDE: VS Code

This will be the list of extensions I will install on your machine:

- C/C++ Extension Pack (Microsoft)
- C/C++ Extension Pack (franneck94)
- C/C++ Runner (franneck94)
- Vscode-icons (VSCode Icons Team)

Are there any other requests / questions?



# Transformation



```
#include <stdio.h>

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

Lets have a deeper understanding of `stdlib.h`



```
#include <stdio.h>
#include <math.h>
const double coeffs[] = {
    72,
    -5601.524004527918,
    15839.254381218243,
    -17990.084825043516,
    11078.251262340189,
    -4157.194592942288,
    1004.360782045639,
    -159.60952959110847,
    16.59282398377248,
    -1.0862681816910813,
    0.0406327158120081,
    -0.0006620771164300821,
};

int main() {
    for(int x = 0; x < 12; x++) {
        double y = 0;
        double t = 1;
        for(int i = 0; i < 12; i++) {
            y += coeffs[i]*t;
            t *= x;
        }
        printf("%c", (char)round(y));
    }
    printf("\n");
}
```

## Transformation

### Function Prototypes `<stdlib.h>`

```
long strtol (const char * string, char ** endptr, int base);
```

`strtol` transforms the string `str` into a long value. `strtol` analyses every single character `string` with the first character. Every character that complies to the base `base` is converted. Leading spaced will be ignored. The char pointer `endptr` references the first character of `string` that is will not be transformed. If `endptr` is not needed it must be set to `NULL`. The number must have the format

`[+|-]decimalnumber`

The base describes the numbering system. It is defined between 2 and 36:

$2 \leq \text{base} \leq 32$

The numbers 10 to 32 are transformed into a to z (A to Z).

The return value is the transformed number. If the transformation was not successful the return value is 0L. Number that exceed the range of long forces a return value of `LONG_MAX` or `LONG_MIN` and `errno` is set to `ERANGE`.

## Transformation

### Function Prototypes `<stdlib.h>`

```
unsigned long strtoul (const char * string, char ** endptr, int base);
```

The transformation happen according to `strtol` but the number at `str` is transformed into an `unsigned long` value. If the transformation was not successful the return value is `0L`. Number that exceed the range of `long` forces a return value of `LONG_MAX` or `LONG_MIN` and `errno` is set to `ERANGE`.

- We can use those functions to transform parameters
- E.g. parameters passing when starting a programme

## Transformation

### Function Prototypes `<stdlib.h>`

```
double strtod (char * string, char ** endptr);
```

`strtod` transforms the string `str` into a value of type `double`. All character until `endptr` are taken into account. Leading spaces are ignored.

The number must have the format

`[+|-]floatnumber`

The return value is the transformed number stored in data type `double`. If transformation was not successful the return value is `0.0`. In case of overflow the value is set to `HUGE_VAL` including correct sign. At underflow the return value is `0.0`. In both cases the variable `errno` is set to `ERANGE`.

## Transformation

### Function Prototypes `<stdlib.h>`

```
int atoi (const char * string);
```

atoi is an equivalent to

```
(int) strtol (str, (char **) NULL, 10);
```

```
long atol (const char * string);
```

atol is an equivalent to

```
strtol (str, (char **) NULL, 10);
```

```
double atof (const char * string);
```

atof is an equivalent to

```
strtod (str, (char **) NULL);
```

# Coding Project

Sort sort sort sort ...

**Me:**

I am good in C language.

**Interviewer:**

Then write "Hello World" using C.

**Me:**



HELLO



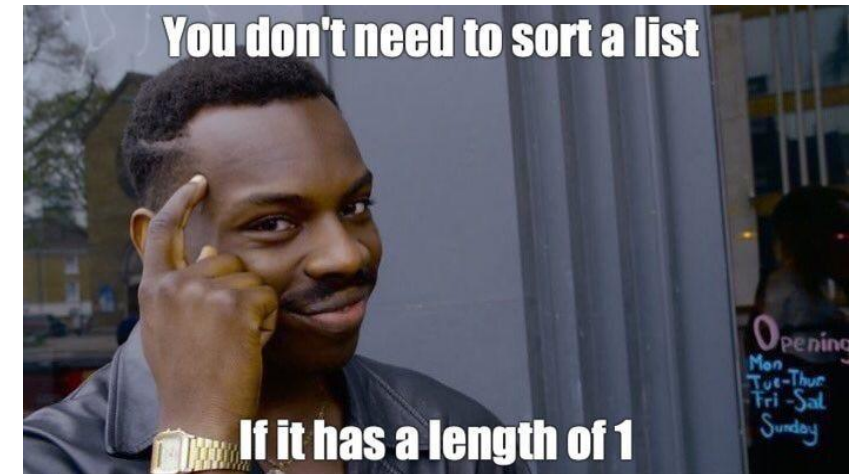
## Coding Project

### Sorting Algorithms

Sorting and Searching are the most frequent techniques in processing data. Remember, the predecessor of the in-processing data.

Focussing on the sorting there are a lot of algorithms:

- bubble sort
- insert sort
- selection sort
- merge sort
- heap sort
- quick sort
- radix sort
- count sort

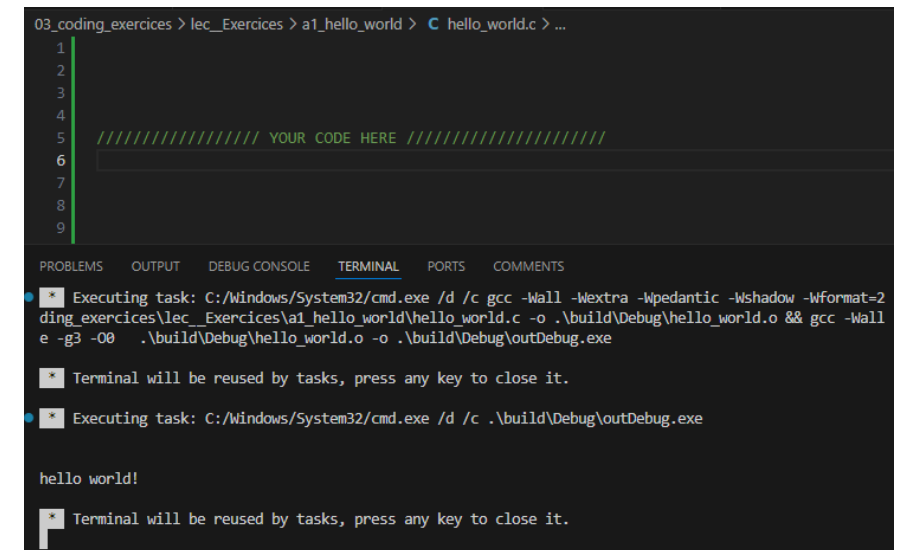


# Coding Project

## Sorting Algorithms

### Project goal

- Introduction and implementation of important sorting algorithms.
- Comparison of the algorithms in terms of time and memory complexity.
- Integration and application of previously covered topics from your course.



```
03_coding_exercises > lec_Exercises > a1_hello_world > C hello_world.c > ...
1
2
3
4
5 /////////////// YOUR CODE HERE ///////////////
6
7
8
9

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
* Executing task: C:/Windows/System32/cmd.exe /d /c gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2
ding_exercises\lec_Exercises\ai_hello_world\hello_world.c -o .\build\Debug\hello_world.o && gcc -Wall
e -g3 -O0 .\build\Debug\hello_world.o -o .\build\Debug\outDebug.exe

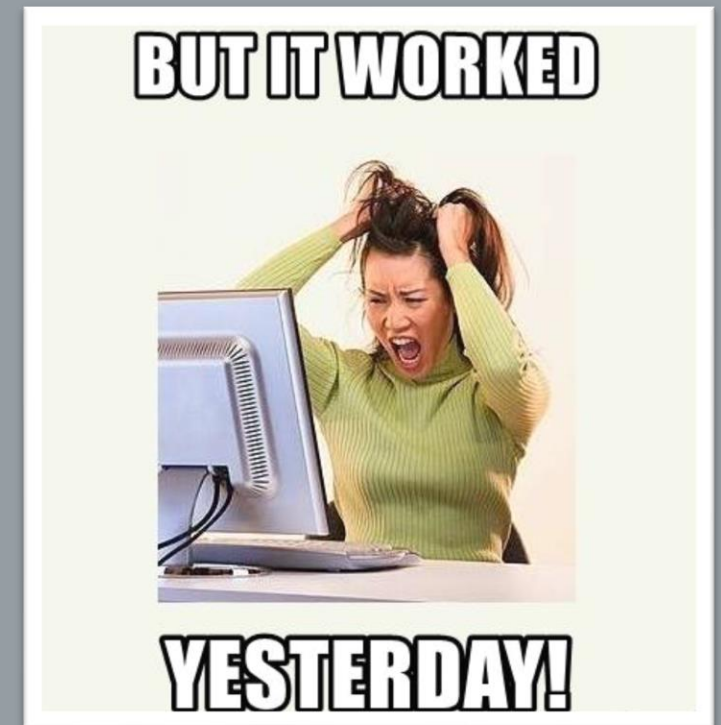
* Terminal will be reused by tasks, press any key to close it.

* Executing task: C:/Windows/System32/cmd.exe /d /c .\build\Debug\outDebug.exe


hello world!

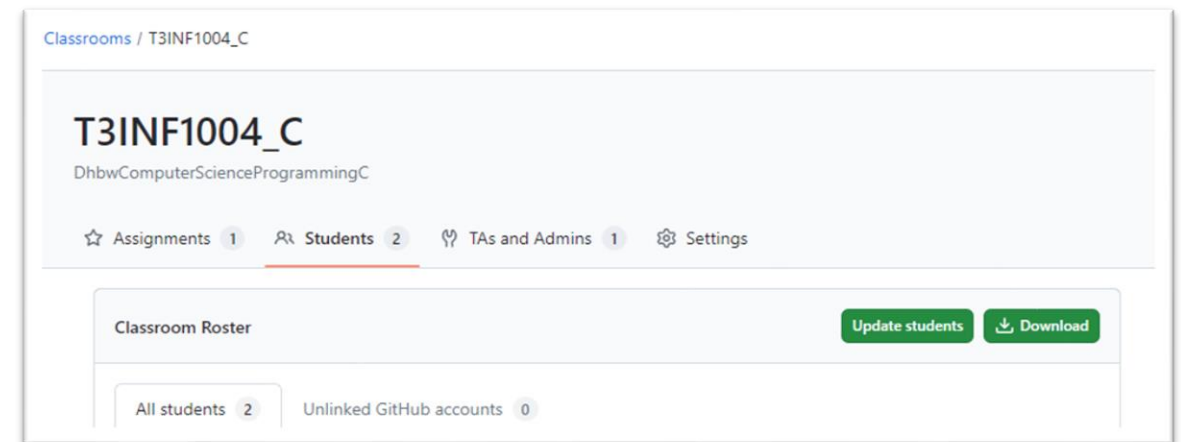
* Terminal will be reused by tasks, press any key to close it.
```

## BACK TO GIT AGAIN



## Your Classroom for C coding Assignments

- Let's come together in the  Classroom
- Assignment will be available for you
- Get the Repository
- **03-Assignment Q&A**



[https://classroom.github.com/classrooms/182848101-t3inf1004\\_c/roster](https://classroom.github.com/classrooms/182848101-t3inf1004_c/roster)