INF1004 procedural programming in C

DHBW Stuttgart
Christian Holz
christian.holz@lehre.dhbw-stuttgart.de

Lecture 03

PART I

- Recap of the previous contents
- Variadic functions
- Debugging

PART II

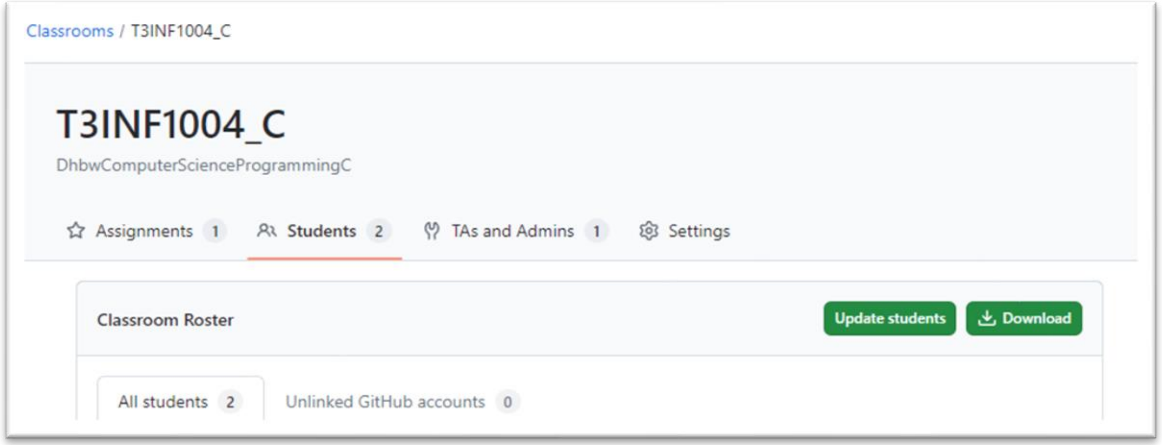
- Transformation
- Coding project

Recap of the previous contents

Live demo

Your Classroom for C coding Assignments

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



Classrooms / T3INF1004_C

T3INF1004_C

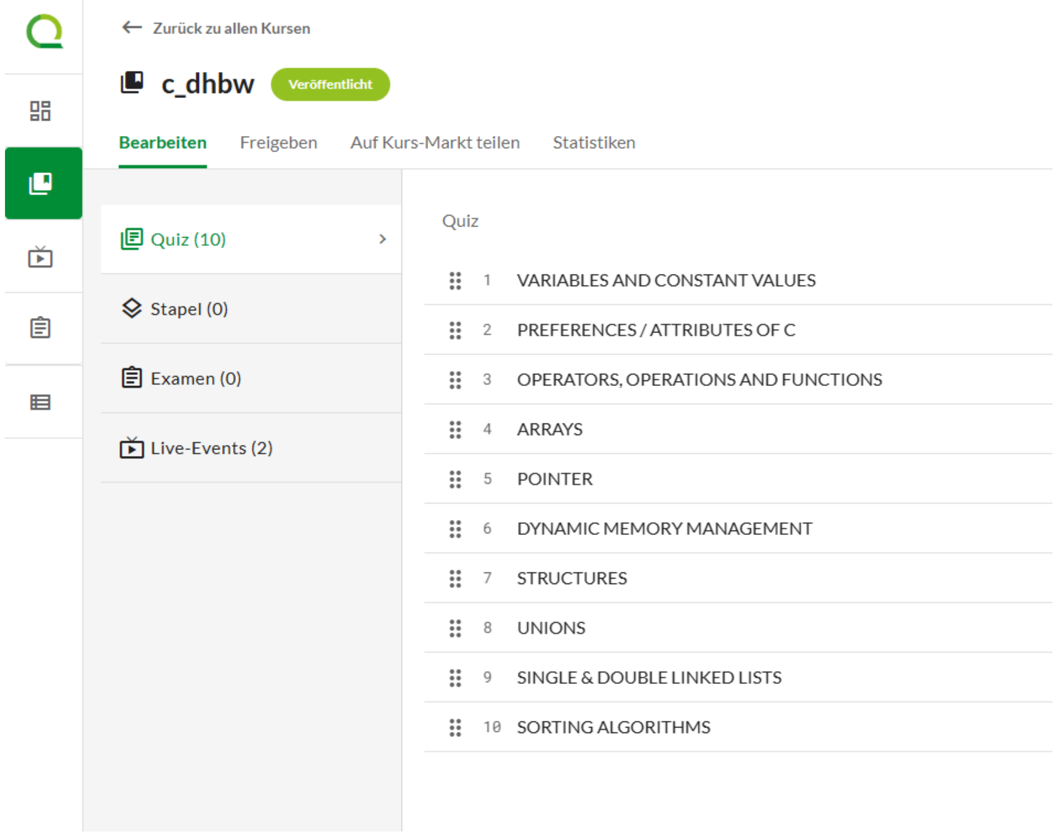
DhbwComputerScienceProgrammingC

Classroom Roster

https://classroom.github.com/classrooms/182848101-t3inf1004_c/roster

Live event

Let’s play



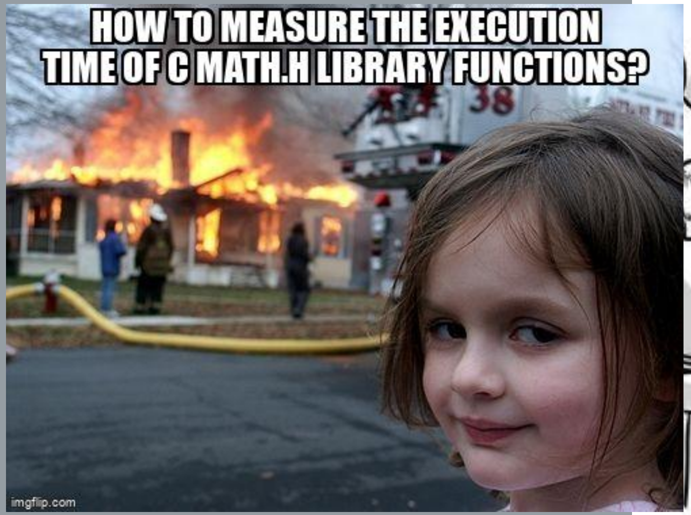
Quiz

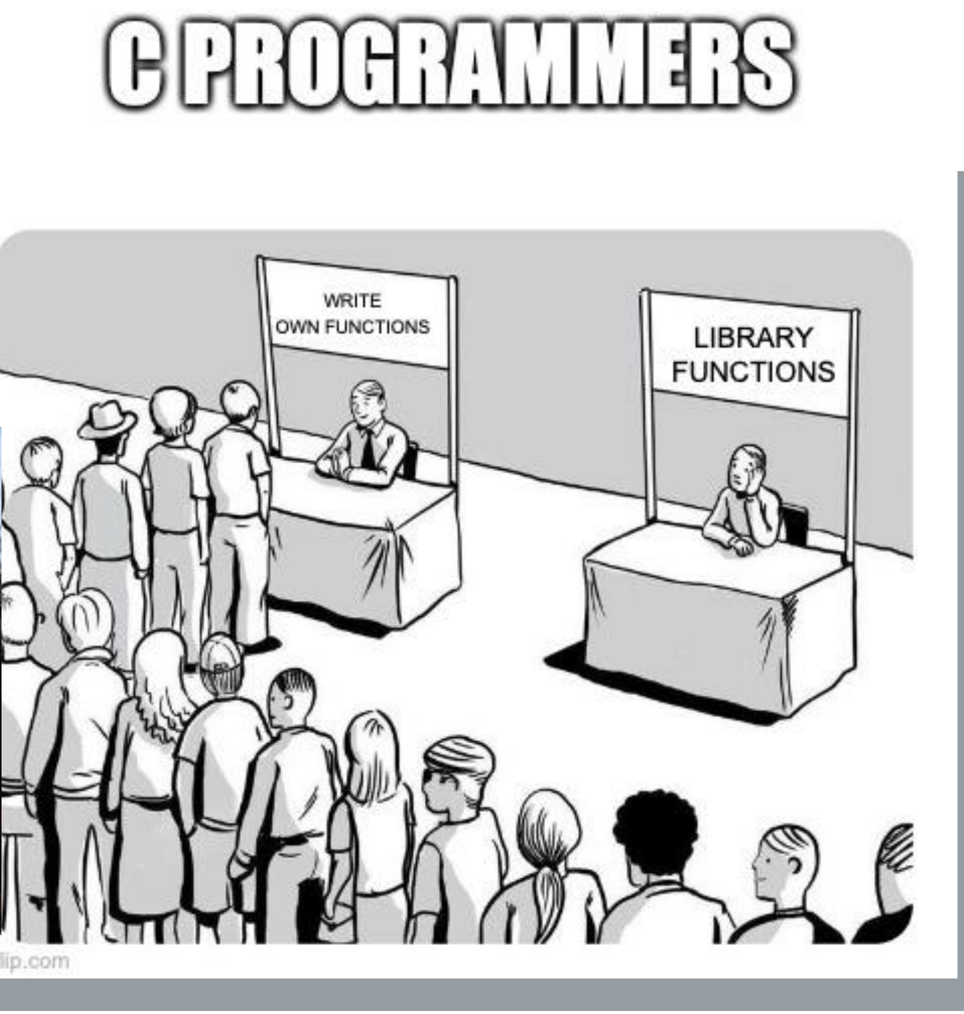
1 VARIABLES AND CONSTANT VALUES
2 PREFERENCES / ATTRIBUTES OF C
3 OPERATORS, OPERATIONS AND FUNCTIONS
4 ARRAYS
5 POINTER
6 DYNAMIC MEMORY MANAGEMENT
7 STRUCTURES
8 UNIONS
9 SINGLE & DOUBLE LINKED LISTS
10 SORTING ALGORITHMS

Variadic functions

~~What is a Function in Programming~~?

How do we code functions at the moment?

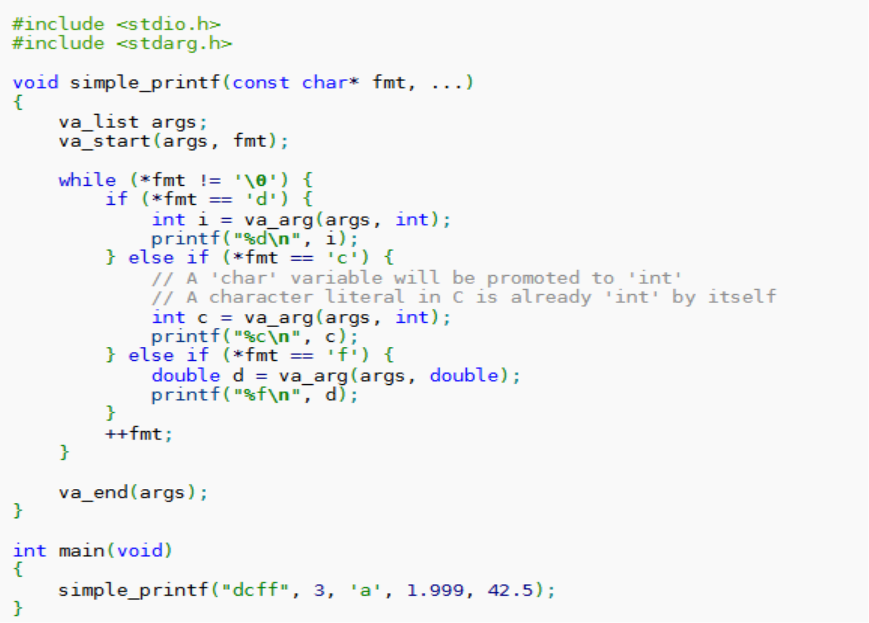

HOW TO MEASURE THE EXECUTION TIME OF C MATH.H LIBRARY FUNCTIONS?
imgflip.com



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.

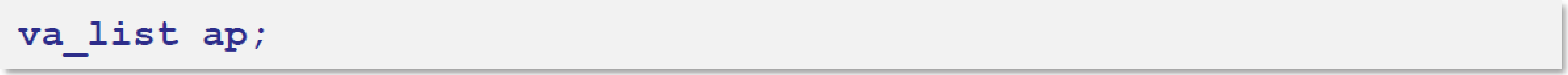


```c
#include <stdio.h>
#include <stdarg.h>

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

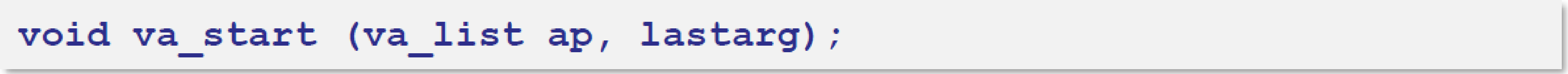
Variadic functions

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.



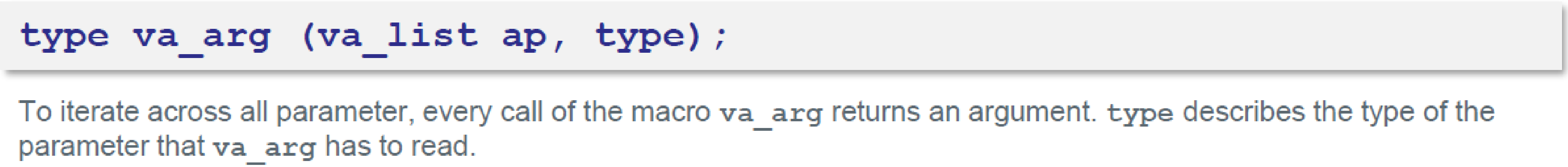
```
void va_start (va_list ap, lastarg);
```

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

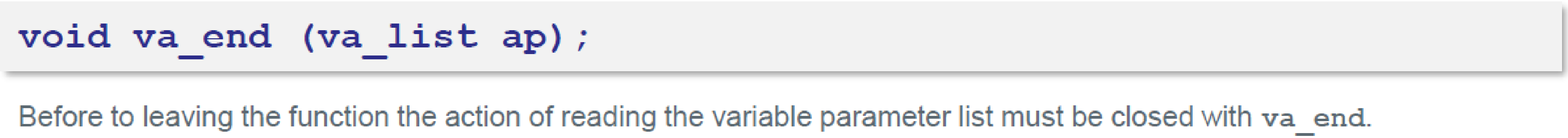
Variadic functions

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.



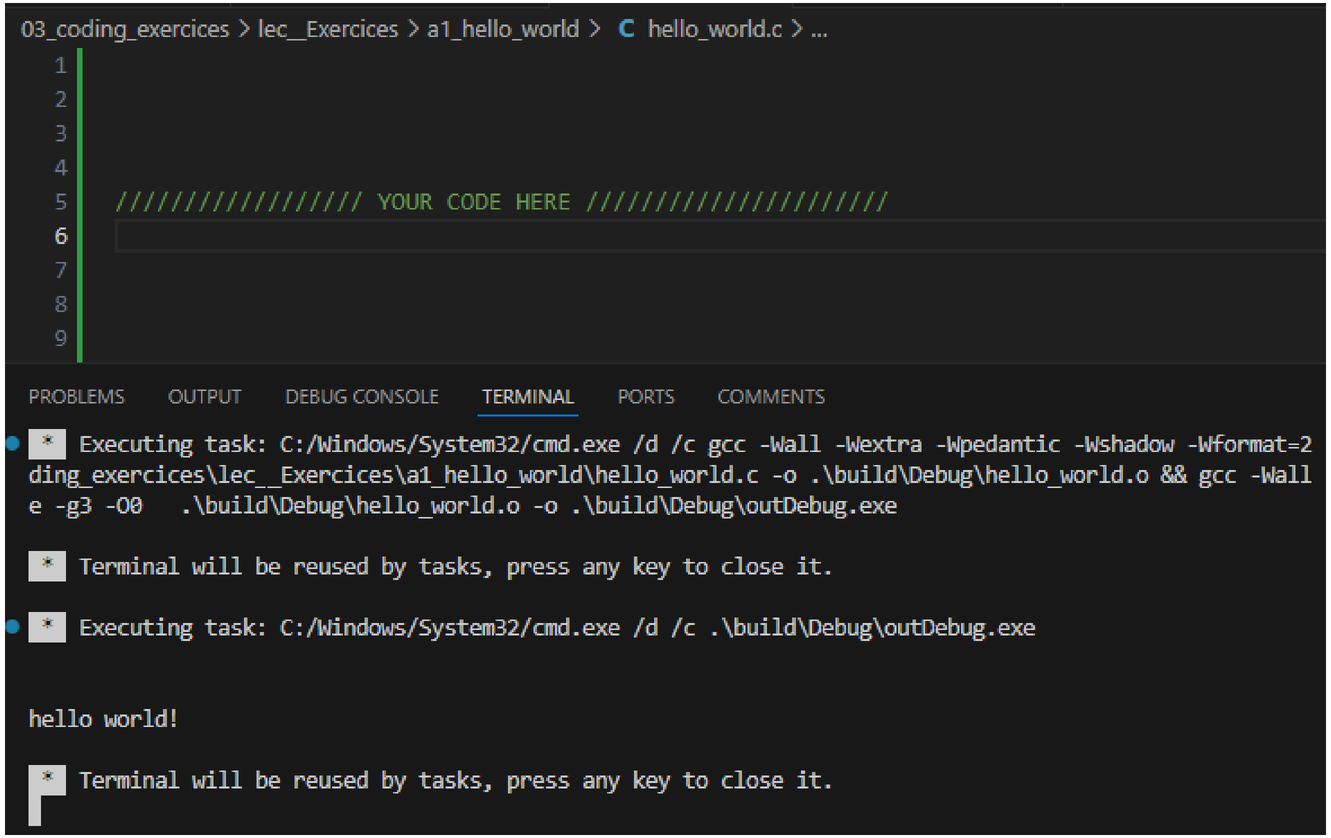
```
void va_end (va_list ap);
```

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

Live coding

Let's code



03_coding_exercices > lec__Exercices > a1_hello_world > C hello_world.c > ...

```c




///////////////// YOUR CODE HERE ////////////////////
```

```
* Executing task: C:/Windows/System32/cmd.exe /d /c gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2
ding_exercices\lec__Exercices\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 dubugging?

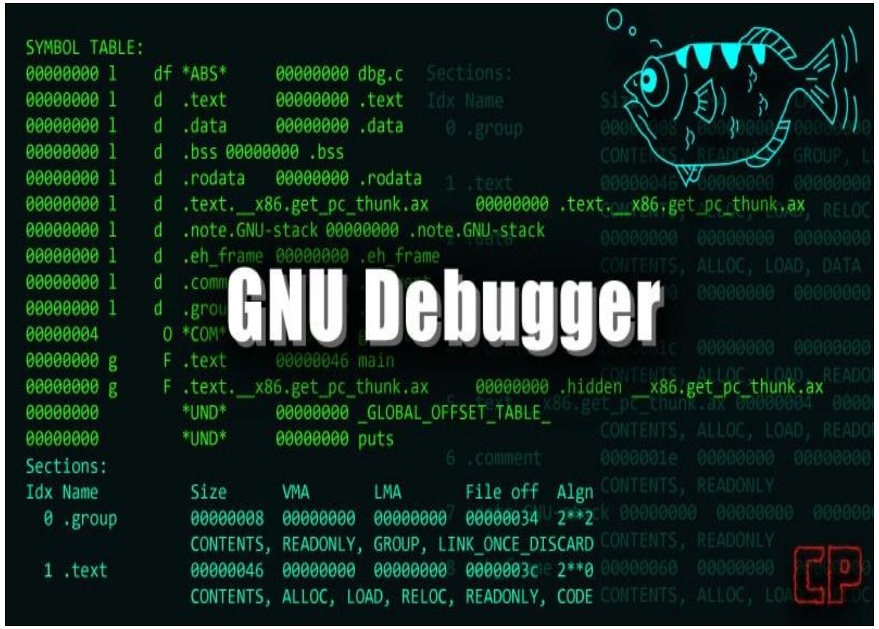
DEBUGGING
I DON'T KNOW WHERE YOU ARE, I DON'T KNOW HOW YOU WORK, BUT I WILL FIND YOU, AND
I WILL FIX YOU

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.



```
SYMBOL TABLE:
00000000 l    df *ABS*     00000000 dbg.c
00000000 l    d  .text     00000000 .text
00000000 l    d  .data     00000000 .data
00000000 l    d  .bss 00000000 .bss
00000000 l    d  .rodata   00000000 .rodata
00000000 l    d  .text.__x86.get_pc_thunk.ax     00000000 .text.__x86.get_pc_thunk.ax
00000000 l    d  .note.GNU-stack 00000000 .note.GNU-stack
00000000 l    d  .eh_frame 00000000 .eh_frame
00000000 l    d  .comm
00000000 l    d  .grou
00000004       O *COM*
00000000 g     F .text     00000046 main
00000000 g     F .text.__x86.get_pc_thunk.ax     00000000 .hidden __x86.get_pc_thunk.ax
00000000         *UND*     00000000 _GLOBAL_OFFSET_TABLE_
00000000         *UND*     00000000 puts
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .group        00000008  00000000  00000000  00000034  2**2
                  CONTENTS, READONLY, GROUP, LINK_ONCE_DISCARD
  1 .text         00000046  00000000  00000000  0000003c  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
```

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

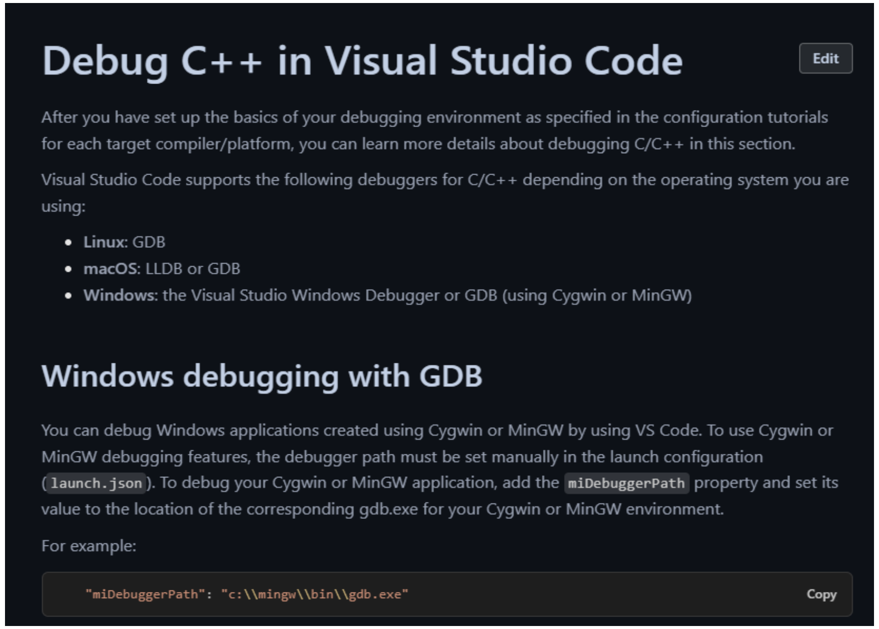
Vishal
@VishalMalvi_
When a developer is debugging :

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

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

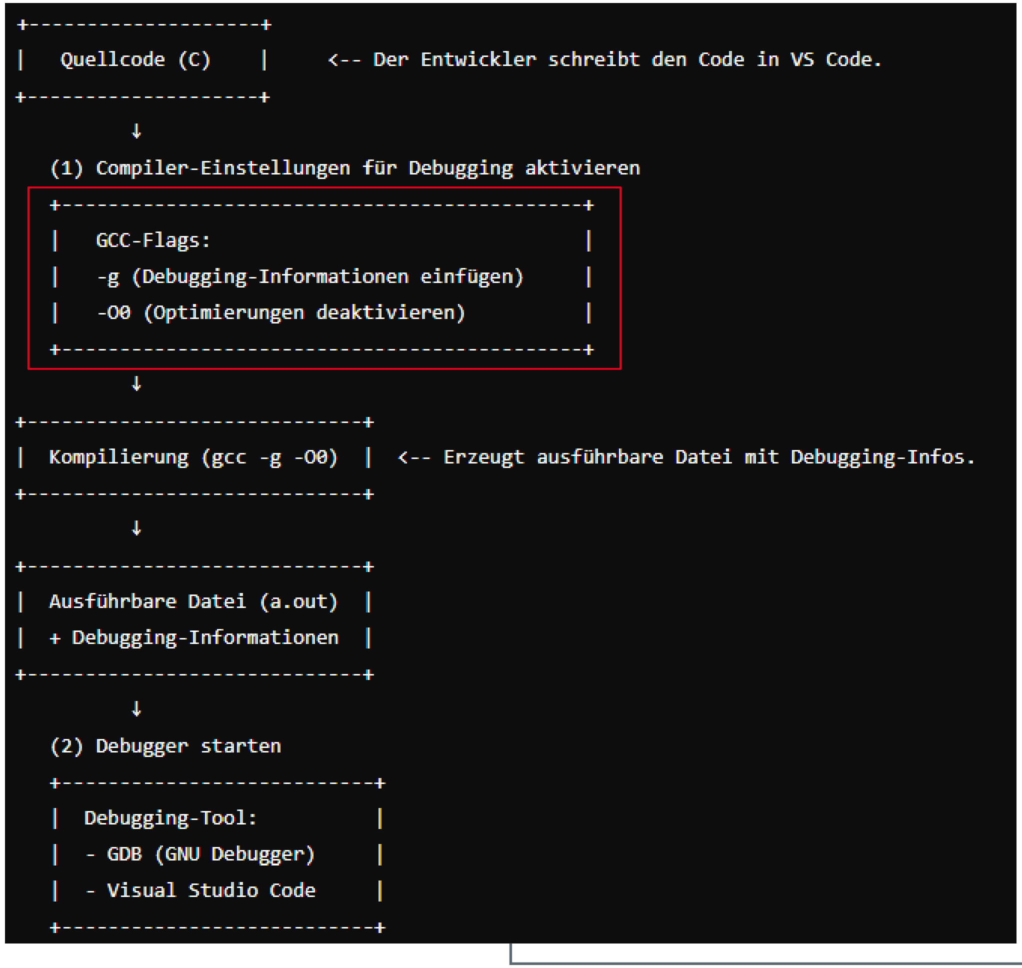
You can debug Windows applications created using Cygwin or MinGW by using VS Code. To use Cygwin or MinGW debugging features, the debugger path must be set manually in the launch configuration (`launch.json`). To debug your Cygwin or MinGW application, add the `miDebuggerPath` property and set its value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

For example:

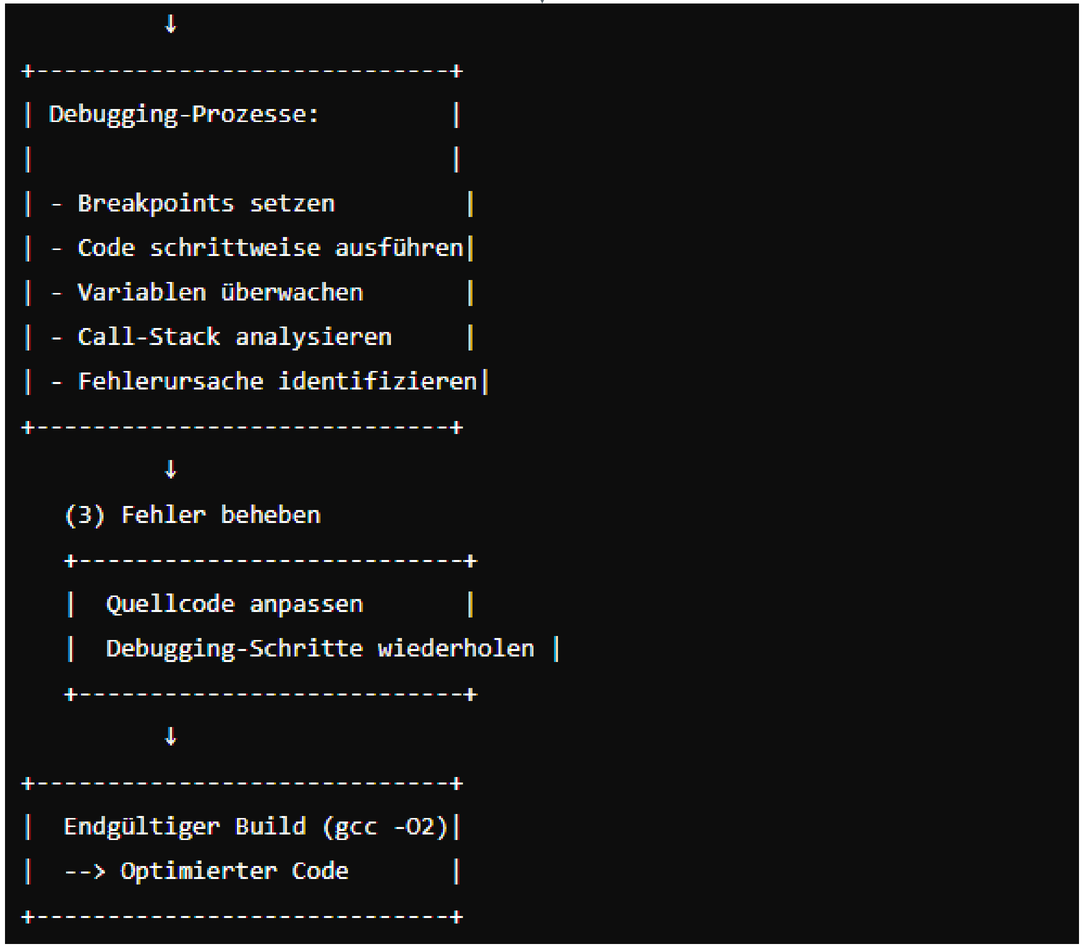
```
"miDebuggerPath": "c:\\mingw\\bin\\gdb.exe"
```

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

Debugging
Tools



```
+------------------+
|   Quellcode (C)    |     <-- Der Entwickler schreibt den Code in VS Code.
+------------------+
          ↓
   (1) Compiler-Einstellungen für Debugging aktivieren
   +------------------------------------------+
   |   GCC-Flags:                               |
   |   -g (Debugging-Informationen einfügen)     |
   |   -O0 (Optimierungen deaktivieren)          |
   +------------------------------------------+
          ↓
+---------------------------+
|  Kompilierung (gcc -g -O0)  |  <-- Erzeugt ausführbare Datei mit Debugging-Infos.
+---------------------------+
          ↓
+---------------------------+
|  Ausführbare Datei (a.out)  |
|  + Debugging-Informationen  |
+---------------------------+
          ↓
   (2) Debugger starten
   +-------------------------+
   |  Debugging-Tool:          |
   |  - GDB (GNU Debugger)     |
   |  - Visual Studio Code     |
   +-------------------------+
```



```
          ↓
+----------------------------+
| Debugging-Prozesse:          |
|                              |
| - Breakpoints setzen         |
| - Code schrittweise ausführen|
| - Variablen überwachen       |
| - Call-Stack analysieren     |
| - Fehlerursache identifizieren|
+----------------------------+
          ↓
   (3) Fehler beheben
   +--------------------------+
   |  Quellcode anpassen        |
   |  Debugging-Schritte wiederholen |
   +--------------------------+
          ↓
+----------------------------+
|  Endgültiger Build (gcc -O2)|
|  --> Optimierter Code        |
+----------------------------+
```

Debugging
VS Code Overview

VARIABLES

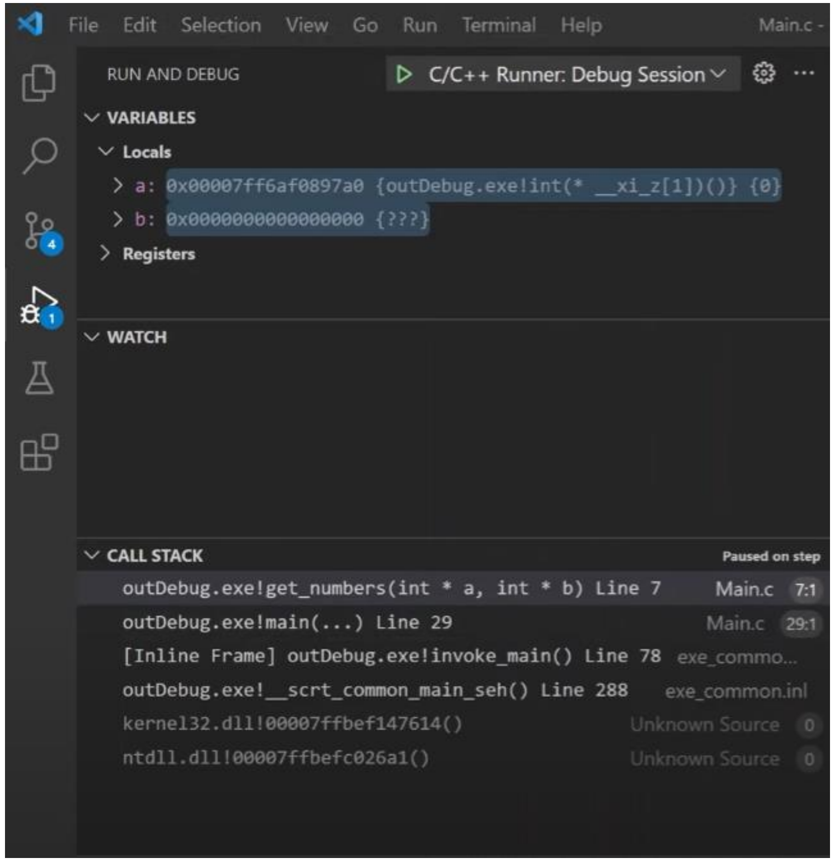
- Shows the current value of the variables in the code during programme execution
- Enables the states of variables to be monitored during runtime

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

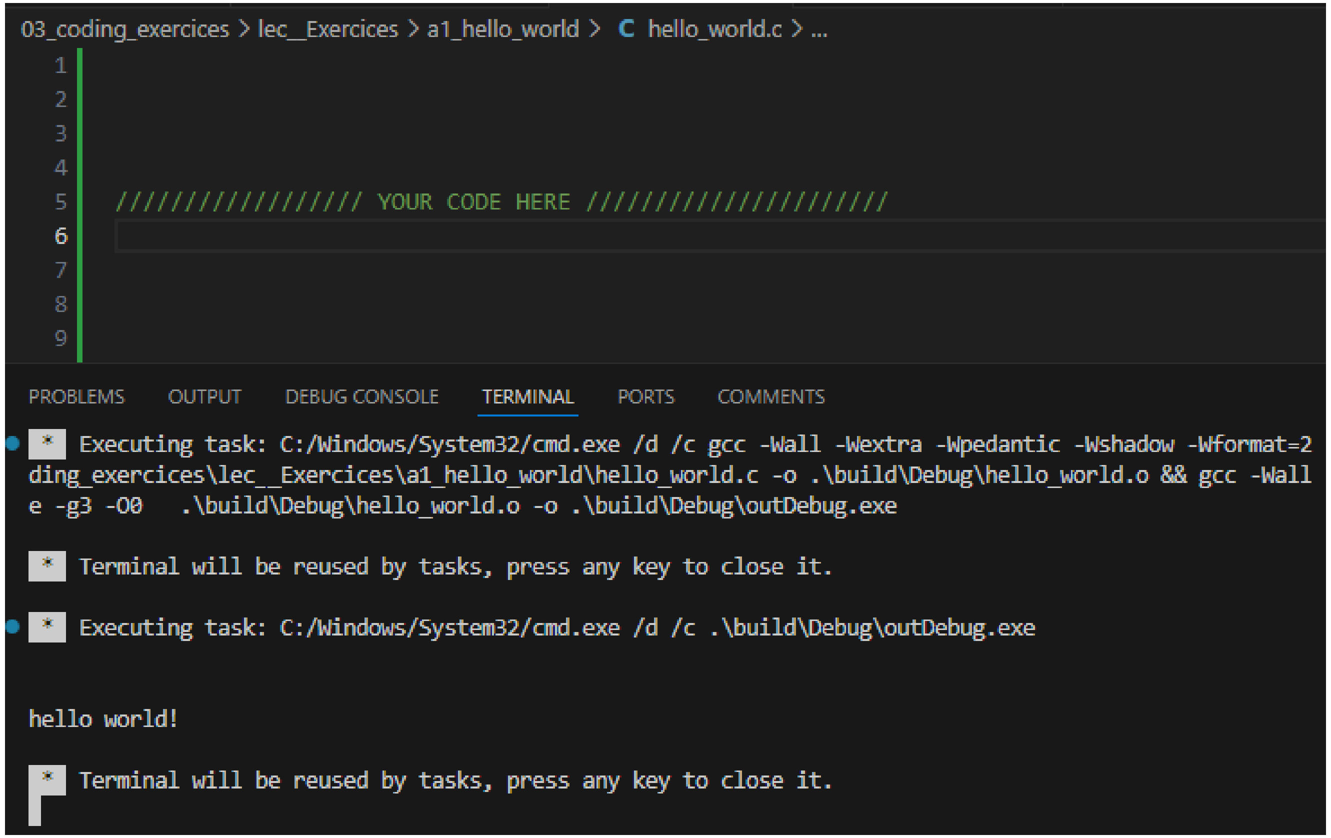
- Returns an overview of the sequence of function calls
- Shows the current function (top level) and the previous calls in chronological order
- At the bottom you may see the OS call



```
VARIABLES
  Locals
    > a: 0x00007ff6af0897a0 {outDebug.exe!int(* __xi_z[1])()} {0}
    > b: 0x0000000000000000 {???}
  > Registers

WATCH

CALL STACK                                                        Paused on step
  outDebug.exe!get_numbers(int * a, int * b) Line 7         Main.c        7:1
  outDebug.exe!main(...) Line 29                            Main.c        29:1
  [Inline Frame] outDebug.exe!invoke_main() Line 78   exe_commo...
  outDebug.exe!__scrt_common_main_seh() Line 288      exe_common.inl
  kernel32.dll!00007ffbef147614()                     Unknown Source      0
  ntdll.dll!00007ffbefc026a1()                        Unknown Source      0
```

Live coding

Let's code



03_coding_exercices > lec__Exercices > a1_hello_world > C hello_world.c > ...

```c




////////////////// YOUR CODE HERE //////////////////////




```

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS

```
 *  Executing task: C:/Windows/System32/cmd.exe /d /c gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2
ding_exercices\lec__Exercices\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.
```

exam
preparation

Tooling for your exam
VS Code and Extensions

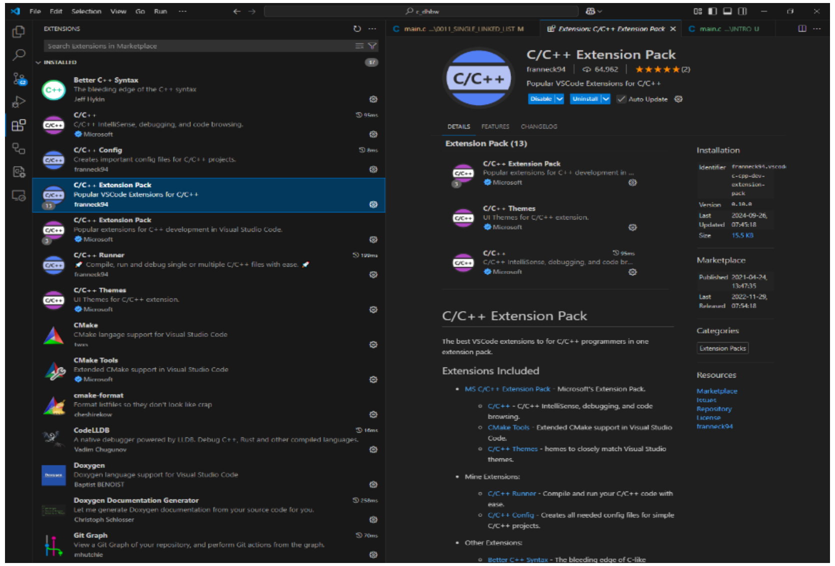
Machine:

- 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)
- Trailing Spaces (Shardul Mahadik)

Are there any other requests / questions?



Better C++ Syntax
The bleeding edge of the C++ syntax
Jeff Hykin

C/C++ 95ms
C/C++ IntelliSense, debugging, and code browsing.
Microsoft

C/C++ Config 8ms
Creates important config files for C/C++ projects.
franneck94

C/C++ Extension Pack
Popular VSCode Extensions for C/C++
franneck94

C/C++ Extension Pack
Popular extensions for C++ development in Visual Studio Code.
Microsoft

C/C++ Runner 199ms
🚀 Compile, run and debug single or multiple C/C++ files with ease. 🚀
franneck94

C/C++ Themes
UI Themes for C/C++ extension.
Microsoft

CMake
CMake langage support for Visual Studio Code
twxs

CMake Tools
Extended CMake support in Visual Studio Code
Microsoft

cmake-format
Format listfiles so they don't look like crap
cheshirekow

CodeLLDB 16ms
A native debugger powered by LLDB. Debug C++, Rust and other compiled languages.
Vadim Chugunov

Doxygen
Doxygen language support for Visual Studio Code
Baptist BENOIST

Doxygen Documentation Generator 258ms
Let me generate Doxygen documentation from your source code for you.
Christoph Schlosser

Git Graph 70ms
View a Git Graph of your repository, and perform Git actions from the graph.
mhutchie

C/C++ Extension Pack
franneck94 | 64,962 | ★★★★★ (2)
Popular VSCode Extensions for C/C++

Extension Pack (13)

C/C++ Extension Pack
Popular extensions for C++ development in ...
Microsoft

C/C++ Themes
UI Themes for C/C++ extension.
Microsoft

C/C++ 95ms
C/C++ IntelliSense, debugging, and code br...
Microsoft

C/C++ Extension Pack

The best VSCode extensions to for C/C++ programmers in one extension pack.

Extensions Included

- MS C/C++ Extension Pack - Microsoft's Extension Pack.
  - C/C++ - C/C++ IntelliSense, debugging, and code browsing.
  - CMake Tools - Extended CMake support in Visual Studio Code.
  - C/C++ Themes - hemes to closely match Visual Studio themes.
- Mine Extensions:
  - C/C++ Runner - Compile and run your C/C++ code with ease.
  - C/C++ Config - Creates all needed config files for simple C/C++ projects.
- Other Extensions:
  - Better C++ Syntax - The bleeding edge of C-like

Installation

Identifier	franneck94.vscode-c-cpp-dev-extension-pack
Version	0.10.0
Last Updated	2024-09-26, 07:45:18
Size	15.5 KB

Marketplace

Published	2021-04-24, 13:47:35
Last Released	2022-11-29, 07:54:18

Categories

Extension Packs

Resources

Marketplace
Issues
Repository
License
franneck94

Transformation

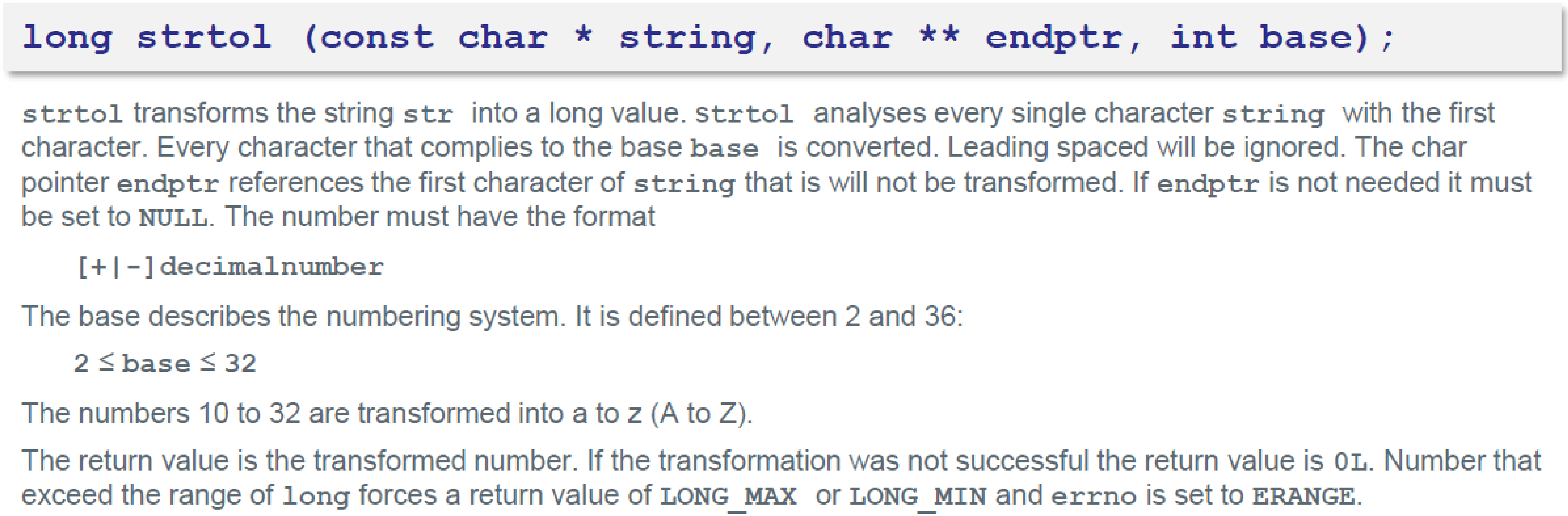
Let's have a deeper understanding of stdlib.h

```
#include <stdio.h>

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

```
#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,
     -0.00066207711643008821,
};
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
```

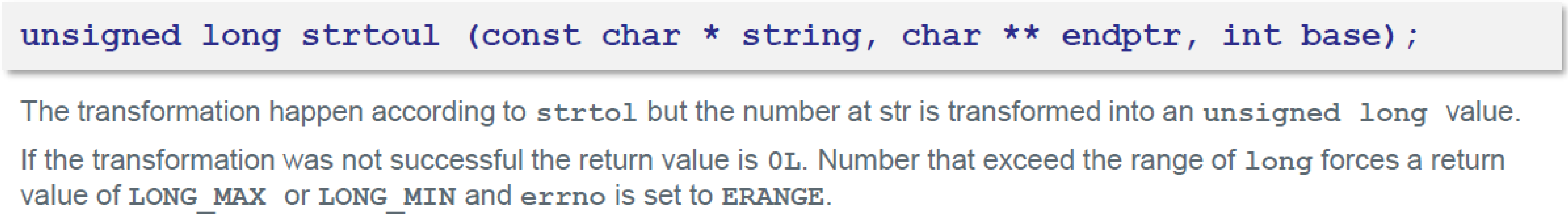
The base describes the numbering system. It is defined between 2 and 36:

$2 \leq$ `base` ≤ 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);
```

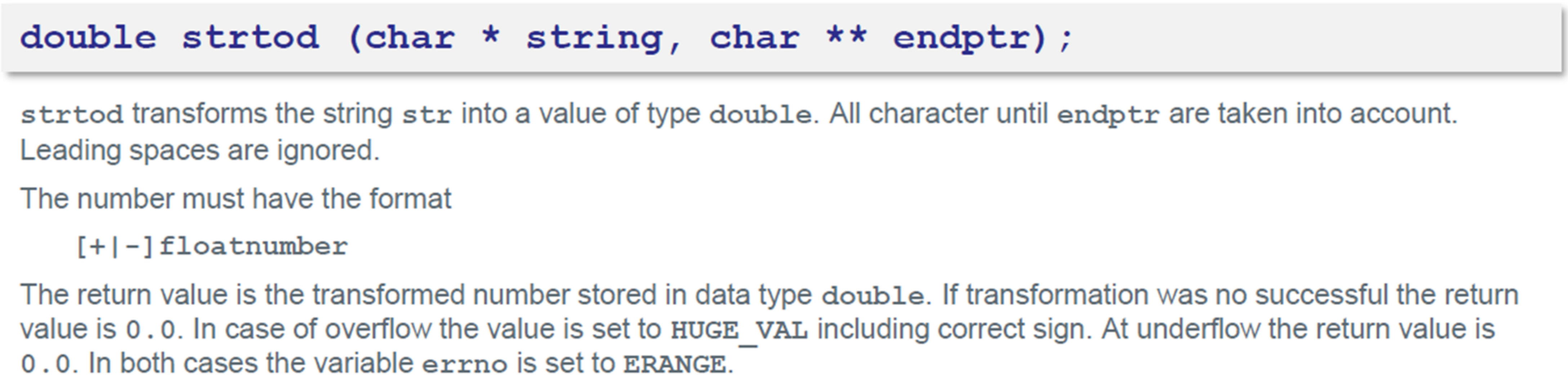
The transformation happen according to `strtol` but the number at str is transformed into an `unsigned long` value.

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);
```

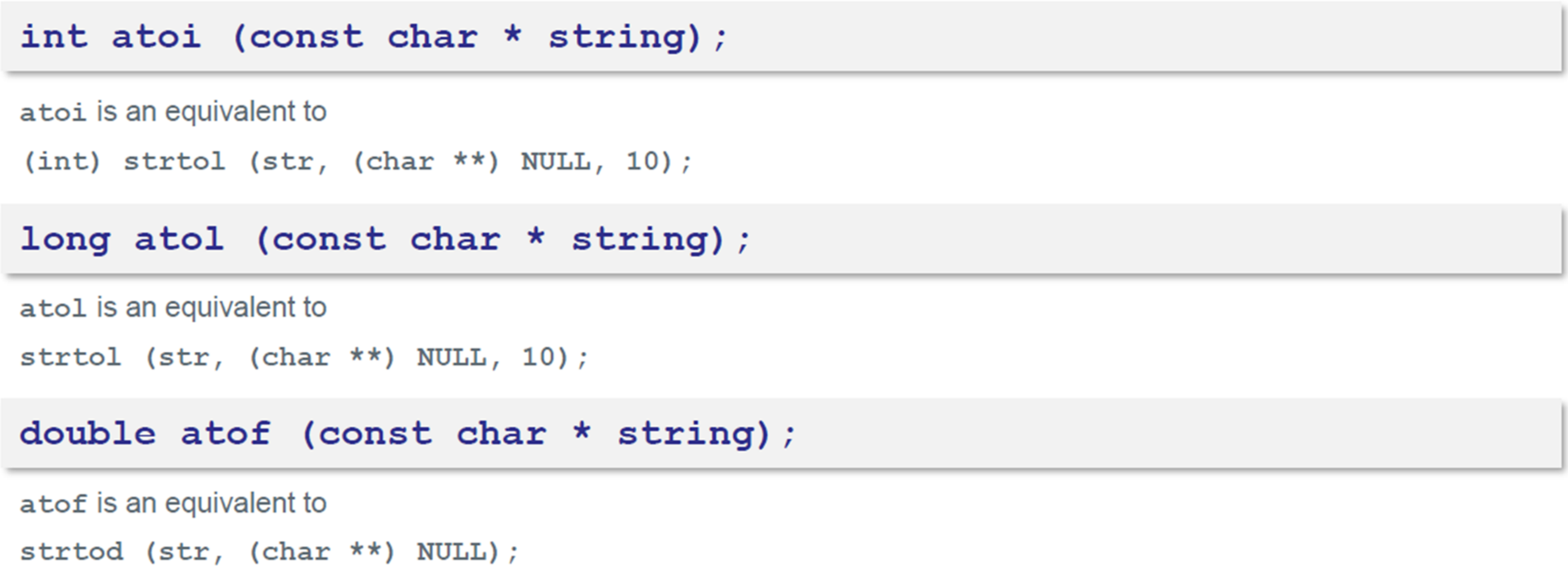
strtod transforms the string str into a value of type double. All character until endptr are taken into account. Leading spaces are ignored.

The number must have the format

```
    [+|-]floatnumber
```

The return value is the transformed number stored in data type double. If transformation was no 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.

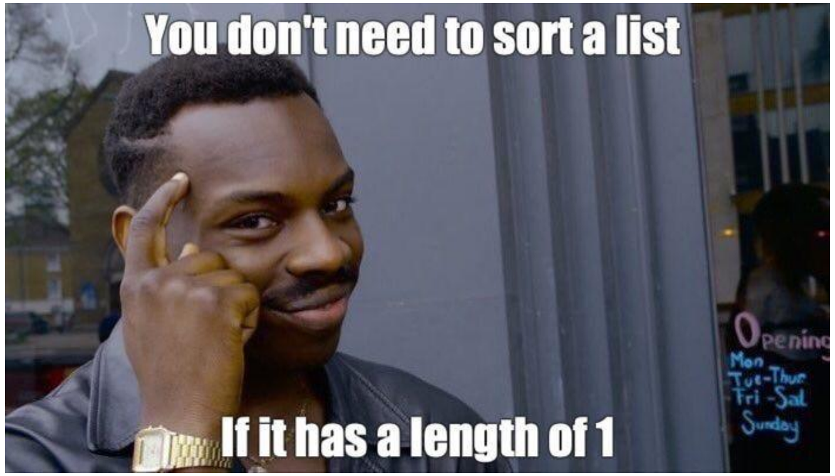
Me:

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**
- ...

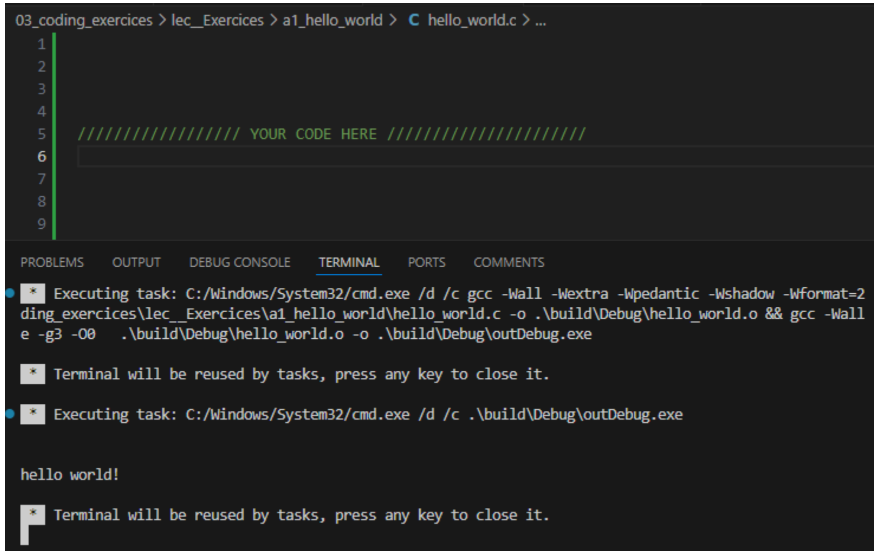


Live coding

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_exercices > lec__Exercices > a1_hello_world > C hello_world.c > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS

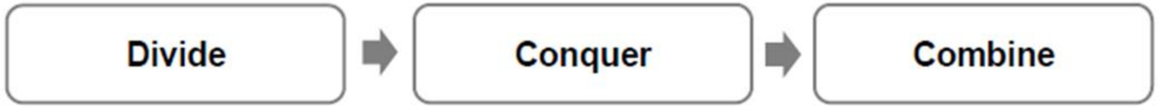
```
////////////////// YOUR CODE HERE //////////////////////
```

```
* Executing task: C:/Windows/System32/cmd.exe /d /c gcc -Wall -Wextra -Wpedantic -Wshadow -Wformat=2
ding_exercices\lec__Exercices\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.
```

Sorting Algorithms

Merge Sort

MergeSort is a Divide and Conquer algorithm.



Divide

Conquer

Combine

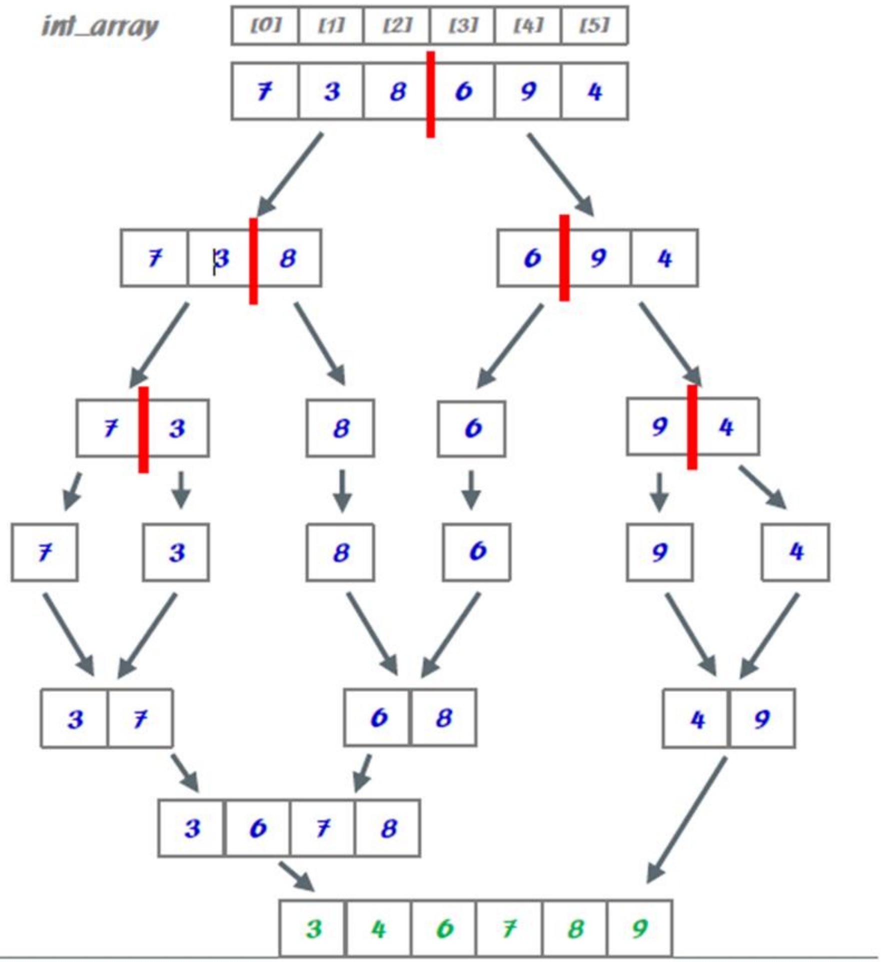
Divide the problem into smaller problems

Solve sub-problems by calling a function recursively

Combine the sub-problems to get the final solution

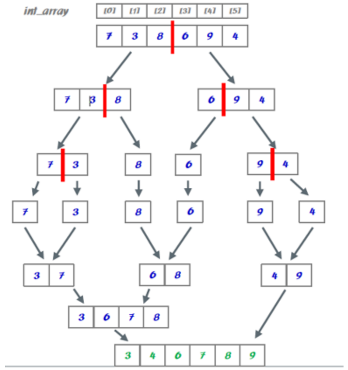
Divide the array in two halves, sort each half and then merge the sorted halves back together

Video by Timo Bingmann:
Visualization and "audibilization" of the Merge Sort algorithm.



Sorting Algorithms

Merge Sort

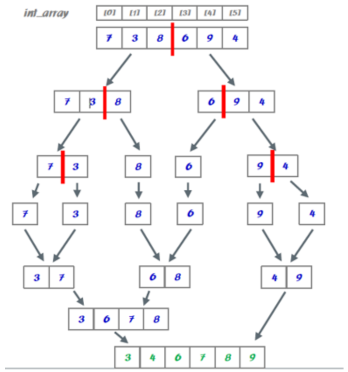


Advantages of Merge Sort

- Stable sorting: The relative order of identical elements is retained.
  (e.g. [3a, 2, 3b, 1] is sorted to [1, 2, 3a, 3b] → 3a and 3b remain in sequence.)
- Guaranteed time complexity of O(n log n)
  Merge Sort always has a runtime of O(n log n), regardless of whether the array is already (partially) sorted or not. Other algorithms such as Quick Sort can drop to $O(n^2)$ in the worst case.
- Efficient for large amounts of data
  Merge Sort is particularly suitable for large arrays and lists.
- Suitable for linked lists
  It does not require direct access to elements (such as Quick Sort) and works well with linked lists.
- Can be parallelised
  Can be easily distributed to multiple processors/threads (divide-and-conquer principle)

Sorting Algorithms

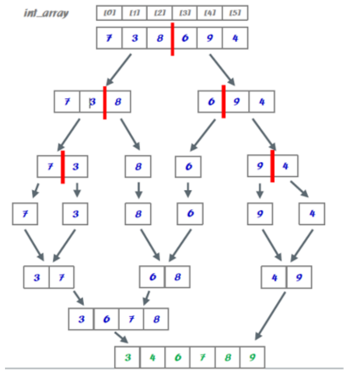
Merge Sort



Disadvantages of Merge Sort

- High memory consumption (O(n))
  Merge Sort requires additional memory for the temporary arrays (leftArray & rightArray).
  Particularly problematic for restricted environments (e.g. embedded systems).
- Slower than Quick Sort for small arrays
  Due to the additional memory required and the many function calls, Merge Sort is often slower for small arrays than Quick Sort or Insertion Sort.
- Bad for in-place sorting
  Standard implementations of Merge Sort are not in-place (i.e. they require additional memory).
  There are optimisations for in-place merge sort, but they are more complex and often slower than quick sort.
- Cache-unfriendly
  Because of the many memory accesses (read & write in temporary arrays), Merge Sort can reduce CPU cache efficiency. Quick Sort is often more cache-friendly as it utilises more local memory accesses.

Sorting Algorithms
Merge Sort



When to use Merge Sort?

- When memory is not a problem and stability is important (e.g. databases, linked lists).
- For very large amounts of data, especially if external sorting (e.g. on a hard drive) is required.
- For parallel processing (e.g. multithreading).

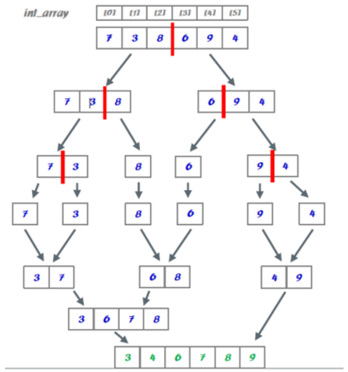
When is it better to use a different algorithm?

- If memory space is limited → Quick Sort or Heap Sort
- When small arrays are sorted → Insertion Sort or Quick Sort
- If in-place sorting is required → Quick Sort or Heap Sort

Sorting Algorithms

…

We will finish this in Lecture 04



Sorting Algorithms
Overview

Algorithmus	Laufzeit (Worst)	Speicherverbrauch	Stabil?	In-Place?	Optimal für
Merge Sort	O(n log n)	O(n)	Ja	Nein	Große, verteilte Daten
Quick Sort	$O(n^2)$ (schlecht)	O(log n)	Nein	Ja	Allgemeiner Standard
Heap Sort	O(n log n)	O(1)	Nein	Ja	Speichereffizienz
Insertion Sort	$O(n^2)$	O(1)	Ja	Ja	Kleine Arrays