INF1004 procedural programming in C

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

Lecture 00

PART I

- Introduction of C
- Algorithms and programs
- Programming Tools
- Variables and Constant Values
- Version management with GIT
- Preferences / Attributes of C

PART II

- Operators, Operations and Functions
- Classes of Data Types
- Control Structures
- Input / Output std functions
- GitHub assignment 00

Lecture 00

PART I

- **Introduction of C**
- **Algorithms and programs**
- **Programming Tools**
- **Variables and Constant Values**
- **Version management with GIT**
- **Preferences / Attributes of C**

PART II

- Operators, Operations and Functions
- Classes of Data Types
- Control Structures
- Input / Output std functions
- GitHub assignment 00

Introduction of C

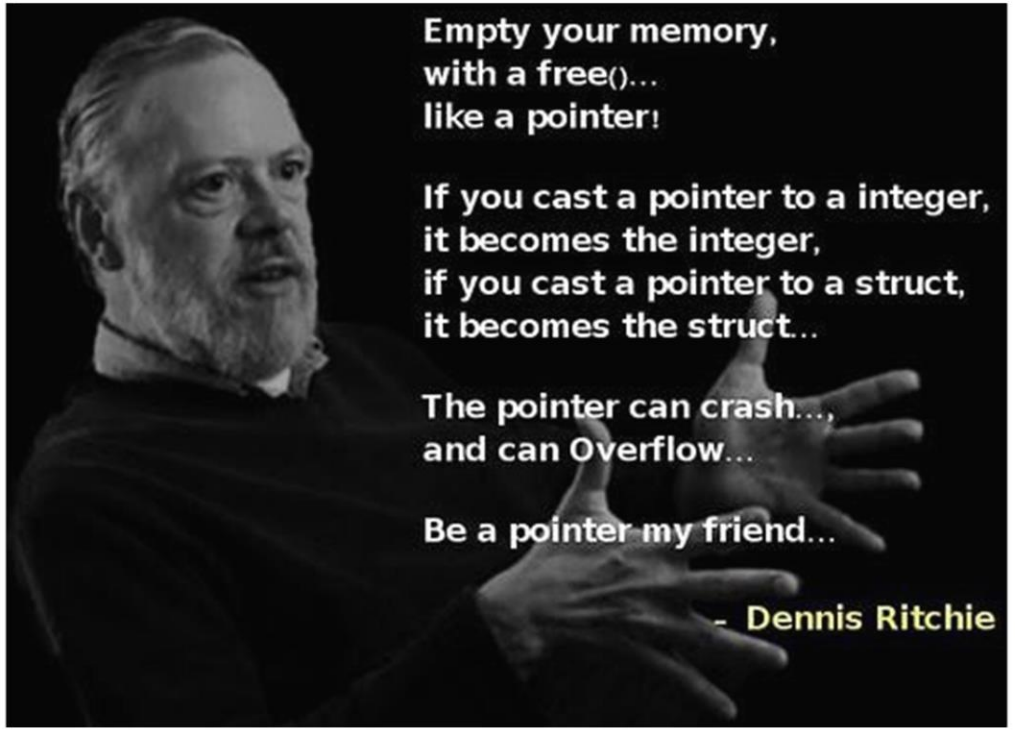
Introduction of C
History

In 1967 Thompson invented the programming language B in which the operation system UNIX is implemented

He looked for a super-assembler that is able to move programs to other systems and has the following attributes:

- structured programming is possible
- close to the hardware implementation like assembler
- performance is nearly equal to assembler

1972 Ritchie (Bell Labs) worked on the programming language C, a language with a code generator and predefined data types



https://media.licdn.com/dms/image/C4D12AQHL9c67LxaP2w/article-cover_image-shrink_600_2000/0/1583314712222?e=2147483647&v=beta&t=_iec1v7gvygLh2tjlrKD1nuoaN2Eej9EjuIbxf50_DU

Introduction of C
History

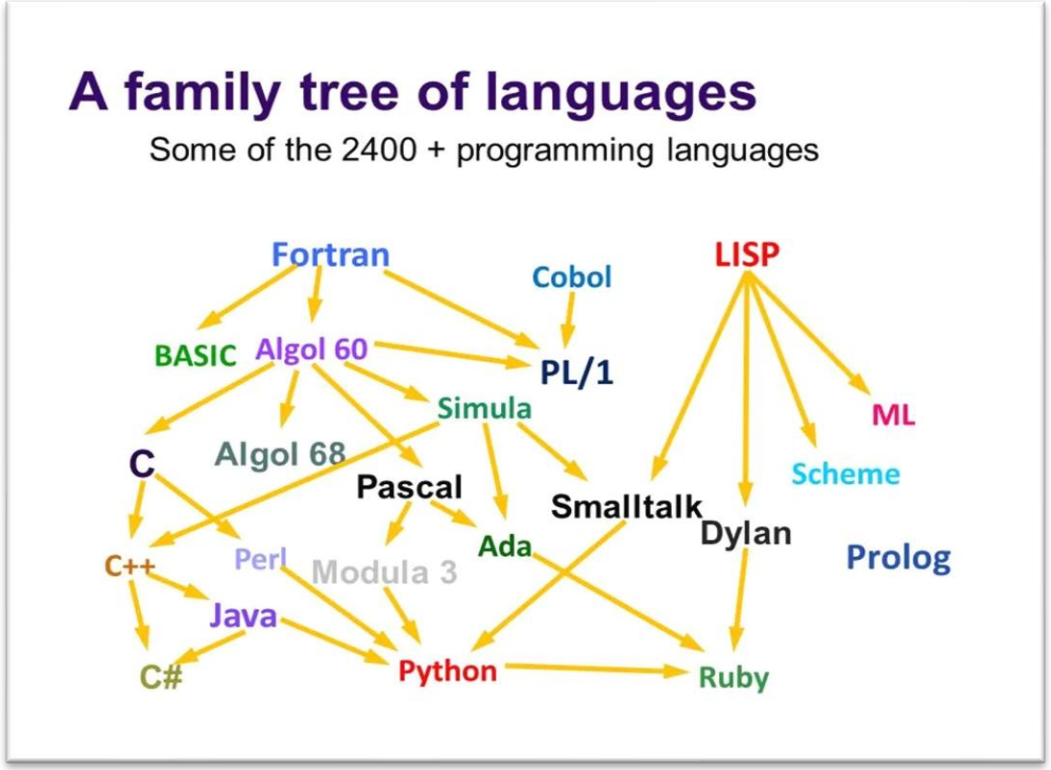
In 1973 UNIX was implemented in C (that means 1/10 of the assembler code)

1978 Kerninghan and Ritchie have written a book named

- “The Programming Language C” what is today known as the C-Bible

1989 there were several dialects unified within the ANSI C-Standard

- libraries had been included standard, too
- today, the ANSI standard has been replaced by the ISO Standard - ISO / IEC 9899



https://medium.com/@christianreyompad/other-variations-of-c-3a78634e7891

Introduction of C
Today

The C programming language continues to have an enormous influence

- Basis for modern languages
- System programming (UNIX, Linux and even large parts of Windows are written in C)
- Embedded Systems
- Performance-critical applications (GNU Compiler Collection (GCC))
- Safety-critical applications

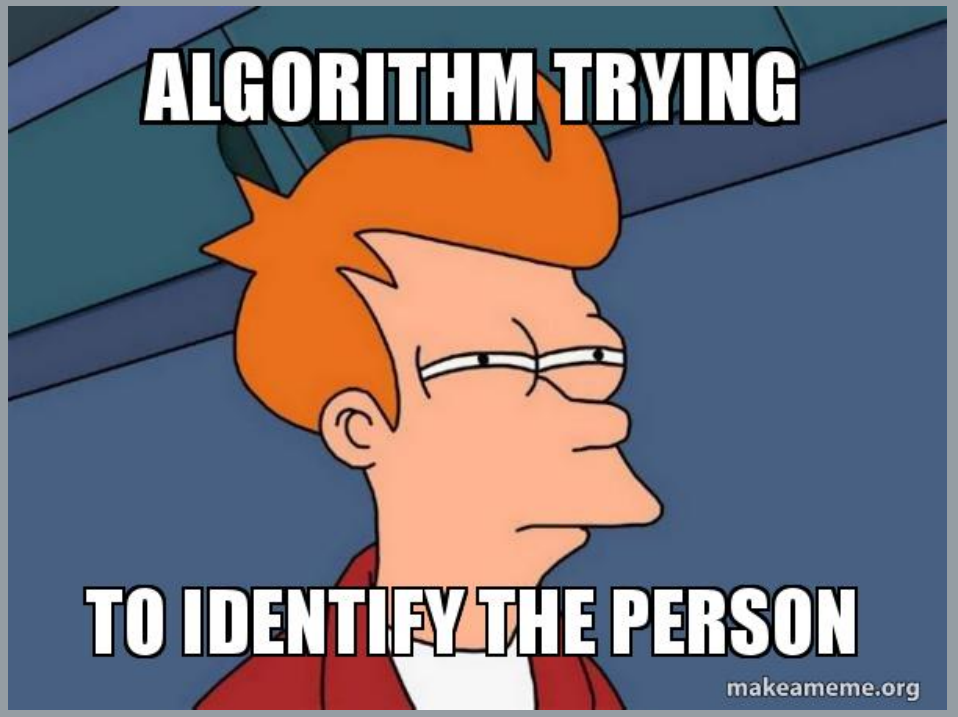
Sep 2024	Sep 2023	Change	Programming Language	Ratings	Change
1	1		Python	20.17%	+6.01%
2	3	↑	C++	10.75%	+0.09%
3	4	↑	Java	9.45%	-0.04%
4	2	↓	C	8.89%	-2.38%
5	5		C#	6.08%	-1.22%
6	6		JavaScript	3.92%	+0.62%
7	7		Visual Basic	2.70%	+0.48%
8	12	⇈	Go	2.35%	+1.16%
9	10	↑	SQL	1.94%	+0.50%
10	11	↑	Fortran	1.78%	+0.49%

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular web sites Google, Amazon, Wikipedia, Bing and more than 20 others are used to calculate the ratings

https://www.tiobe.com/tiobe-index/

Algorithms and programs

What do you think an algorithm is in general?



Algorithms and programs
… and what is a program?

In order to be able to describe an algorithm in a form that can be executed by a machine, a formal language is used to represent algorithms.

A description of an algorithm is called a program, the formal language a programming language.

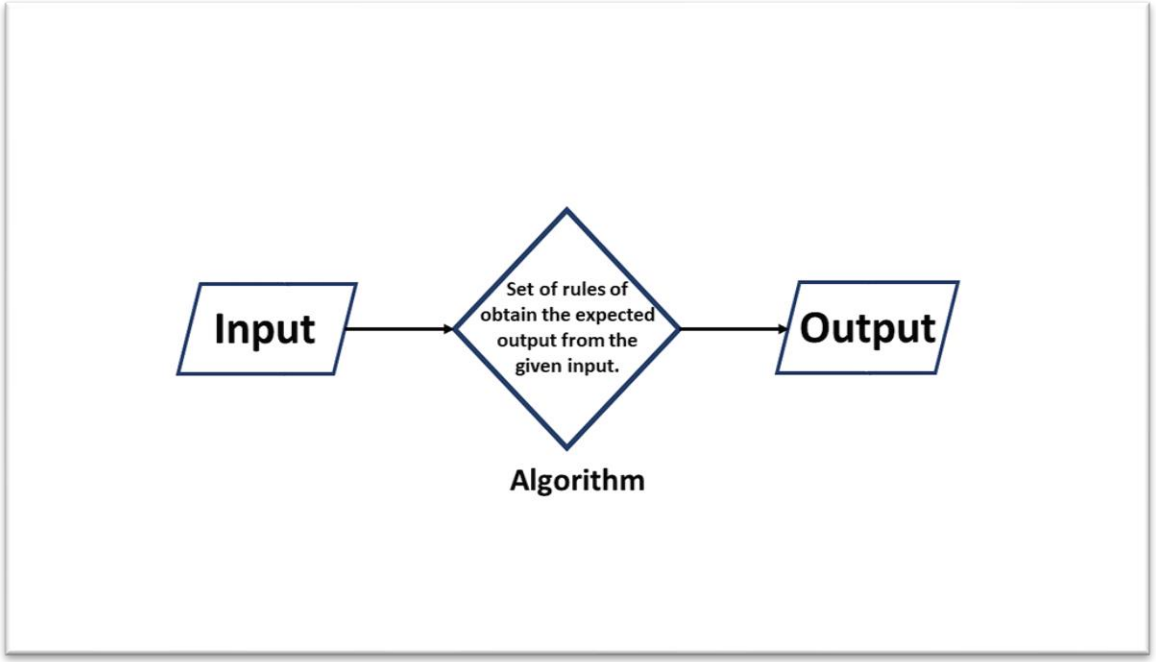
Manfred Broy, Informatik, Teil I, Springer 1992

Algorithms and programs
a systematic and clear sequence of instructions

An algorithm is the description of a procedure for calculating certain output variables from certain input variables.

The following conditions must be met:

- **Specification**
- **Feasibility**
- **Correctness**



https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-an-algorithm

Algorithms and programs
a systematic and clear sequence of instructions

Specification

- Input specification: It must be specified exactly which input variables are required and which requirements these variables must fulfil for the process to work
- Output specification: It must be specified exactly which output variables (results) are calculated with which properties

Algorithms and programs
a systematic and clear sequence of instructions

Feasibility

- Finite description: the process must be fully described in a finite text
- Effectiveness: each step of the process must be effectively (i.e. actually) "mechanically" executable "Effectiveness" is not to be confused with "efficiency" ("economy")
- Determinacy: the process sequence is fixed at all times, i.e. there are no degrees of freedom in the description of the algorithm for decisions relating to the process sequence

Algorithms and programs
a systematic and clear sequence of instructions

Correctness

- Partial correctness: Each calculated result fulfils the output specification, provided that the inputs have fulfilled the input specification
- Termination: The algorithm stops after a finite number of steps with a result, provided the inputs have satisfied the input specification
- Total correctness: partial correctness and scheduling

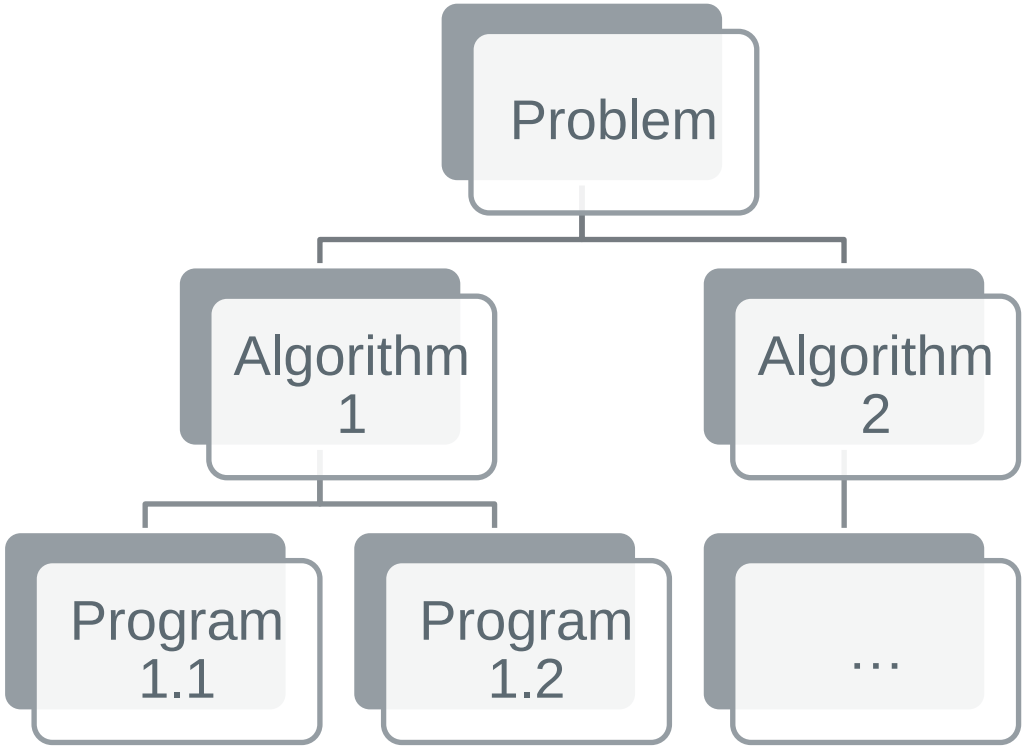
Algorithms and programs
basic connection between the central concepts

Each program represents a specific algorithm.

An algorithm is represented by many different programs.

Programming requires algorithm development

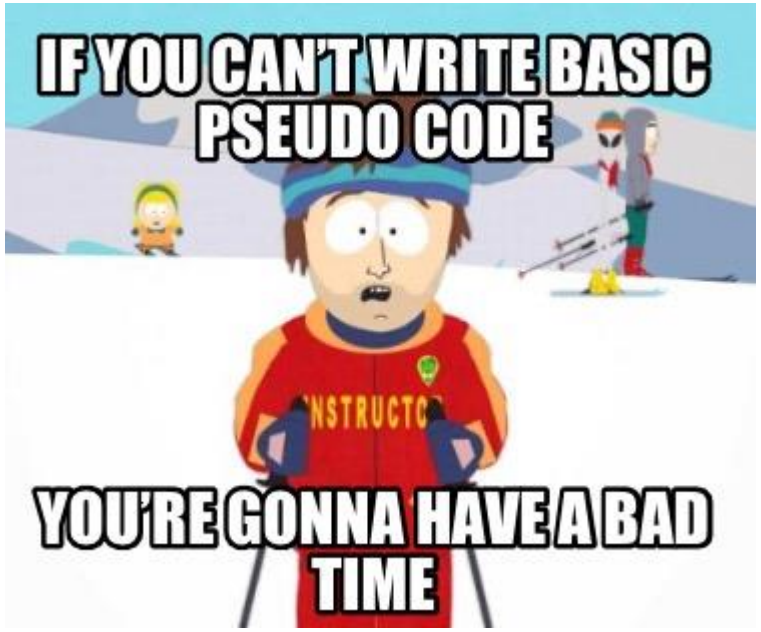
First step: Invest your time in efficient algorithms
Second step: efficient implementation



Algorithms and programs
description of algorithms

Description mechanisms

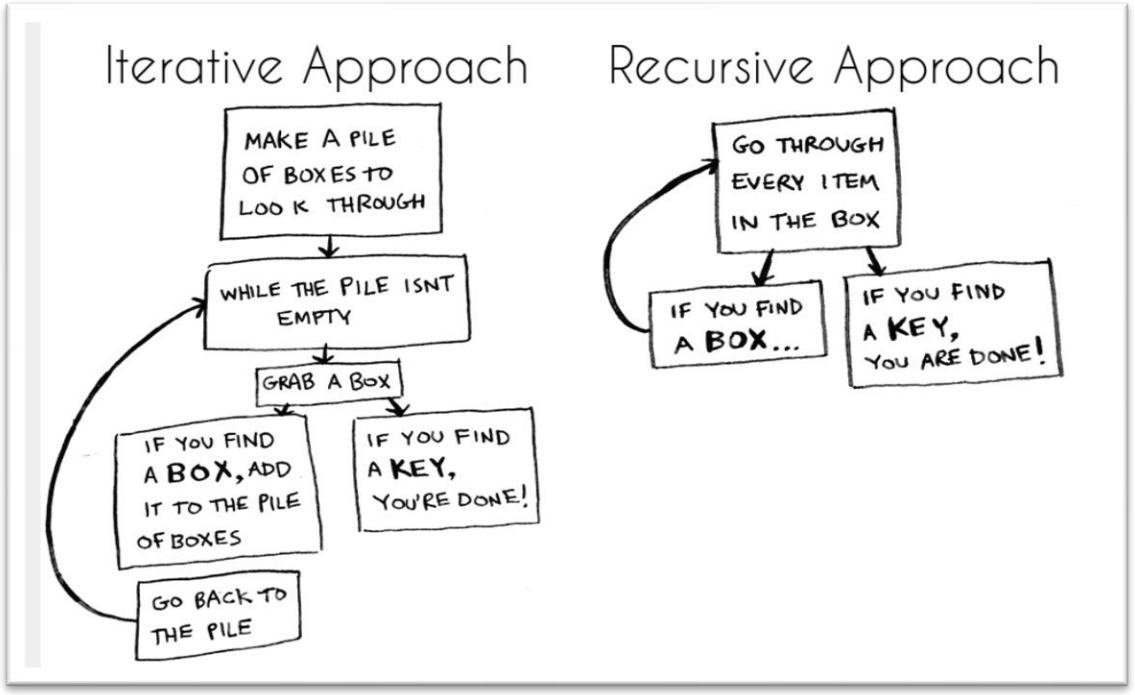
- Everyday language
- Concrete programming language
- In between: A variety of notations to facilitate the transition between problem description and program
  - Pseudocode
  - Flowcharts



Algorithms and programs
arrangement of the instructions

Algorithms follow individual processing steps that are repeated until the desired result is achieved.

- Iterative (continue with step n)
- Recursive (calling the algorithm again with output of n-1 as input)



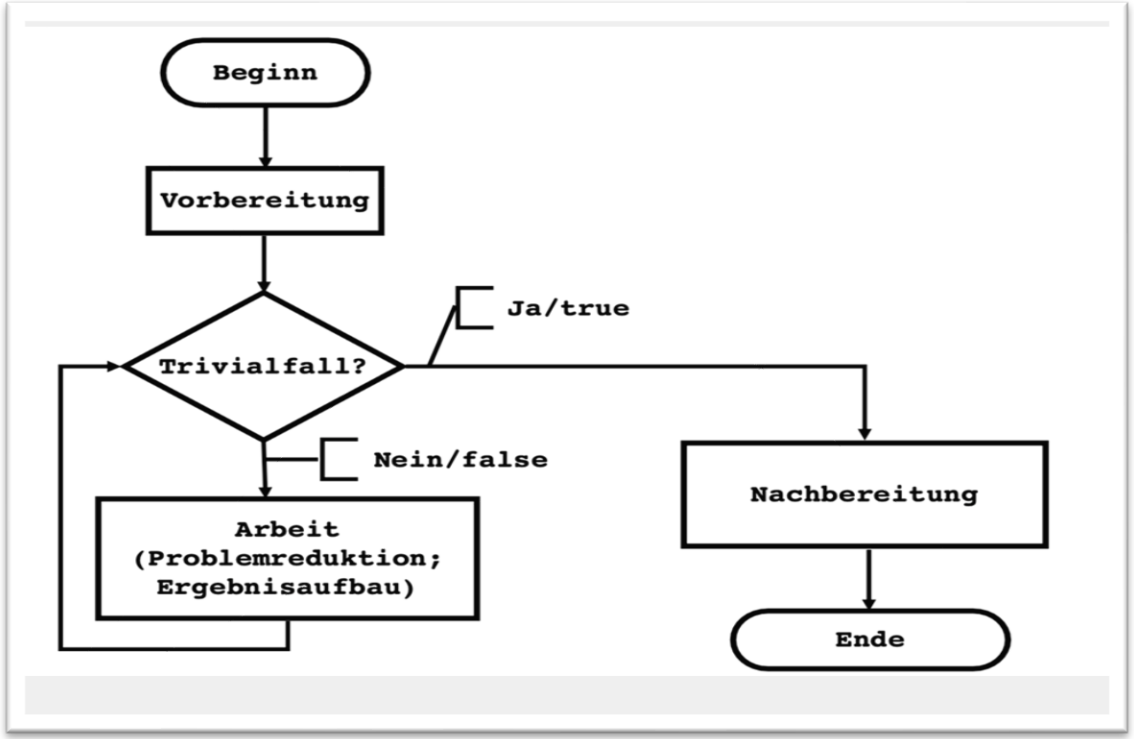
https://miro.medium.com/v2/resize:fit:2000/1*QrQ5uFKIhK3jQSFYeRBIRg.png

Algorithms and programs
arrangement of the instructions

Control flow of the algorithm

Graphical representation of the control process: flow charts

- The flowchart language uses the shown symbols
- Symbols are connected with an arrow
- Execution: follows arrows between boxes



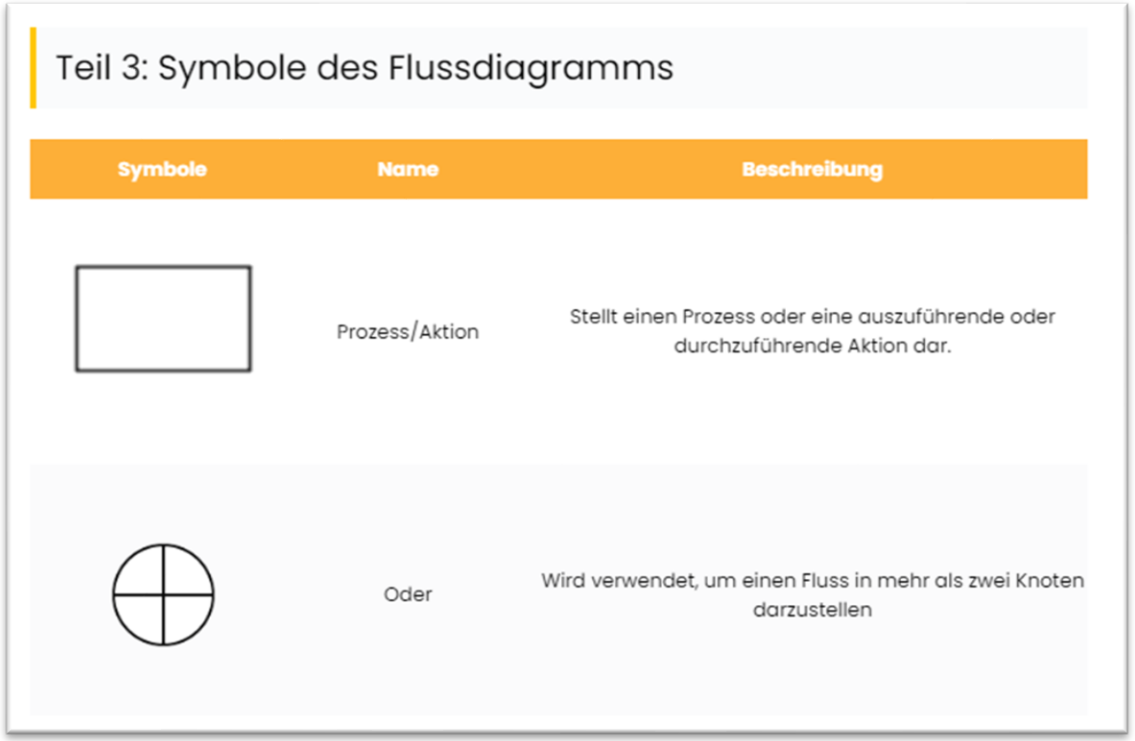
https://www.edrawmax.com/flowchart/

Algorithms and programs
arrangement of the instructions - **Control flow of the algorithm**

Despite the increasing spread of computer algorithms, flowcharts are still used to describe them.

Let's check the definitions

https://www.edrawsoft.com/de/flowchart-definition.html



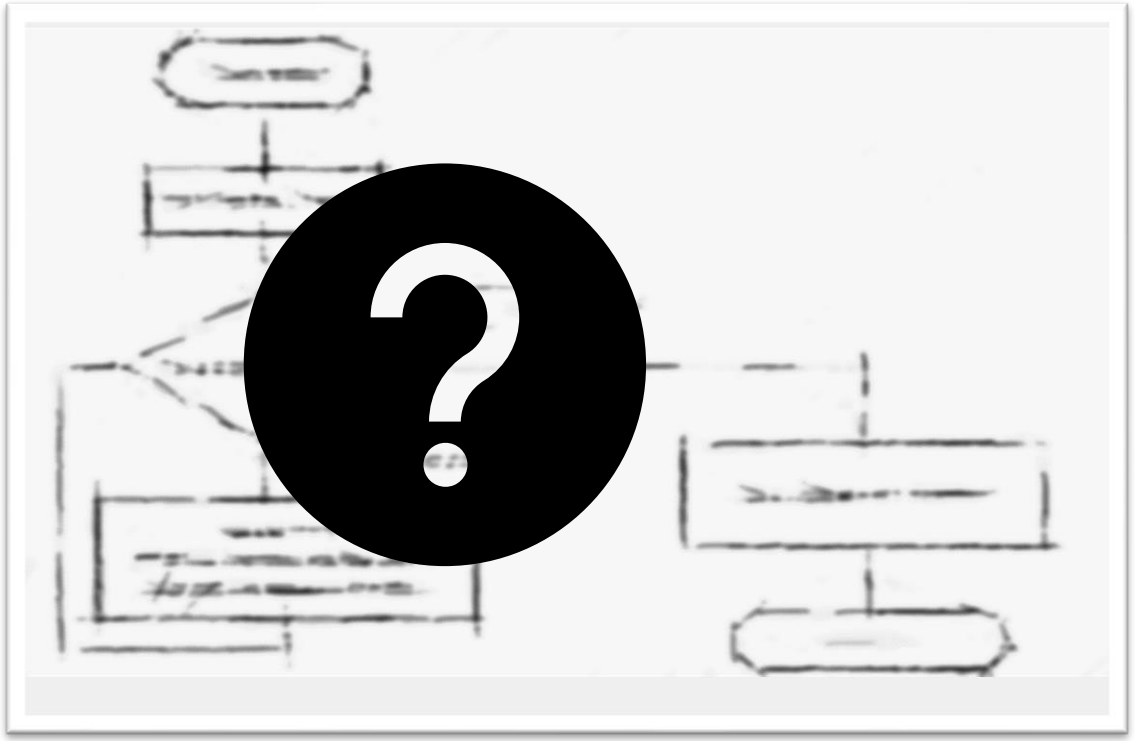
Teil 3: Symbole des Flussdiagramms

Symbole	Name	Beschreibung
	Prozess/Aktion	Stellt einen Prozess oder eine auszuführende oder durchzuführende Aktion dar.
	Oder	Wird verwendet, um einen Fluss in mehr als zwei Knoten darzustellen

Algorithms and programs
Let's build an algorithm

Task

- What is the famous Euclidean algorithm doing? How does it work?
- Let's create a control flow together of the algorithm of Euclid.



Algorithms and programs
Additional information

There are several ways to visualize the sequence of an algorithm, depending on your preference and level of detail:

- Process description (step-by-step)
- Pseudocode
- State diagram (State Diagram)
- **Flowchart**
- Structure diagram (Nassi-Shneiderman diagram)
- **UML activity diagram (Unified Modelling Language)**
- …

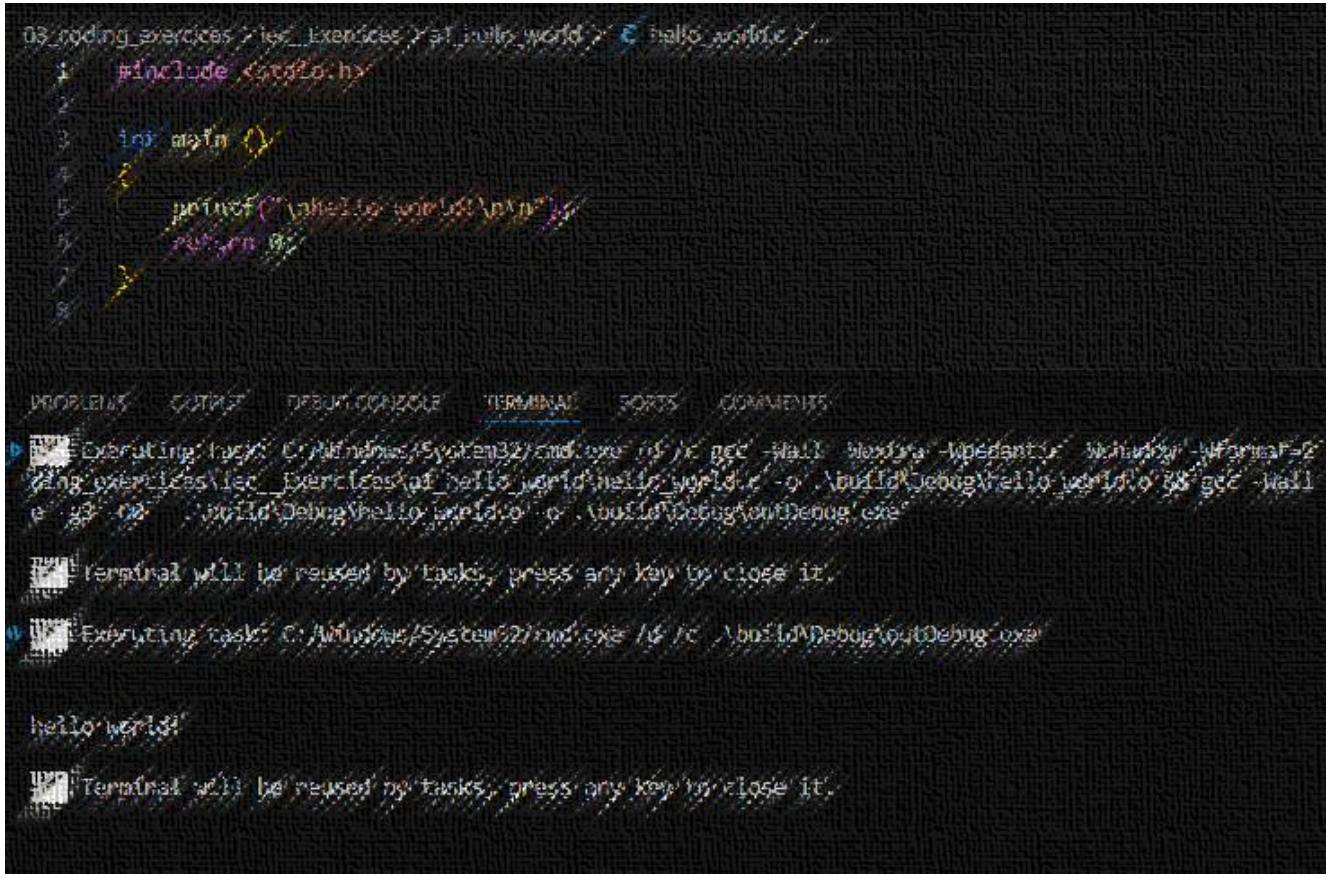
My personal favourite methods

Programming Tools

What do we need to be able to develop programs?

Live coding

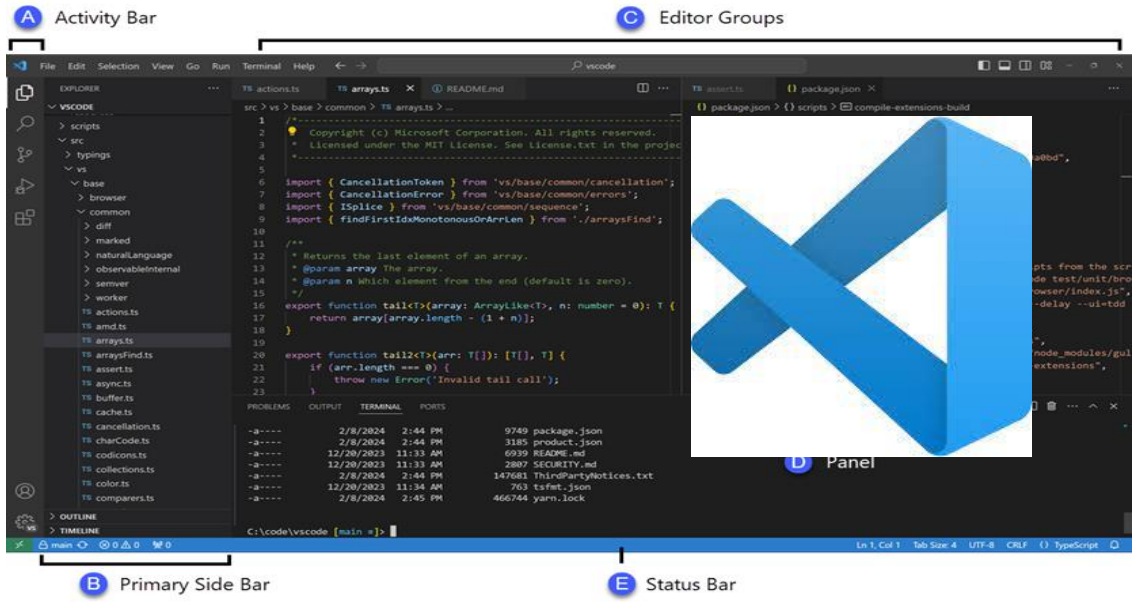
Programming Tools
HelloWorld



Programming Tools
Integrated Development Environment (IDE)

Visual Studio Code

- VS Code is used in the lecture
- You can of course use another IDE (I just can't help if in doubt)
- Powerful code editor for almost all programming languages
- Light weight and fast, no separate compiler as with Visual Studio
- Extensions can be installed via packages (there are all kinds of helpful extensions) Download extensions via Marketplace



https://code.visualstudio.com/docs/getstarted/userinterface

My personal favourite code editor

Live demo

Programming Tools
Set-up

Download Cygwin:
https://cygwin.com/install.html

Cygwin is an open-source software that provides a Unix-like environment under Microsoft Windows.

Download VS Code:
Download Visual Studio Code - Mac, Linux, Windows

Cygwin

Get that Linux feeling - on Windows

This is the home of the Cygwin project

What...

...is it?

Cygwin is:

- a large collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows.
- a DLL (cygwin1.dll) which provides substantial POSIX API functionality.

...isn't it?

Cygwin is not:

- a way to run native Linux apps on Windows. You must rebuild your application *from source* if you want it to run on Windows.
- a way to magically make native Windows apps aware of UNIX® functionality like signals, ptys, etc. Again, you need to build your apps *from source* if you want to take advantage of Cygwin functionality.

Cygwin version

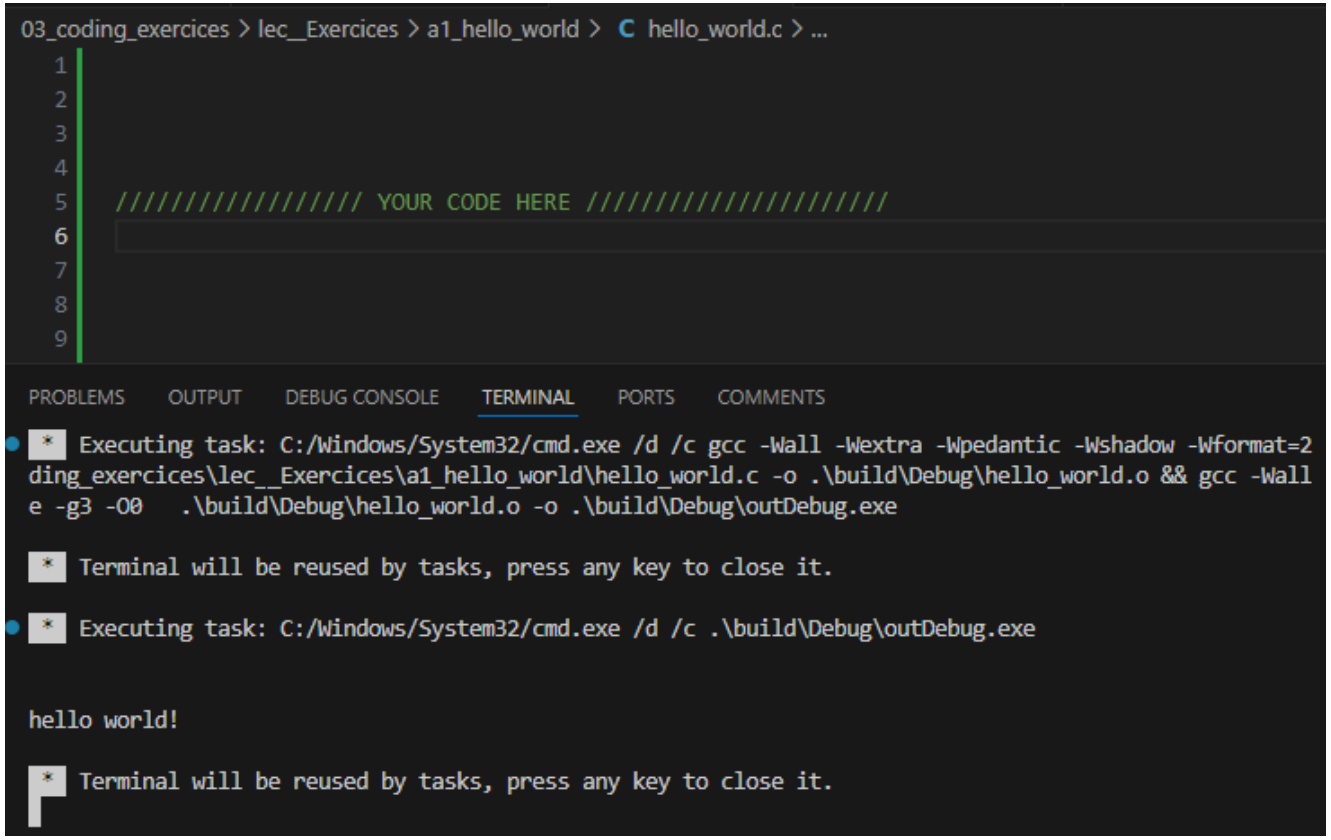
The most recent version of the Cygwin DLL is 3.5.4.

~~Programming Tools~~ Coding Conventions

What are the best practices in C

INF1004_c_coding_conventions.pptx

Programming Tools
Let's code your first .c



Programming Tools
Compilation process

- Pre-processor
- Compiler
- Linker
- Runtime Environment

~~**GNU C Compiler**~~
GNU Compiler Collection

https://de.wikipedia.org/wiki/GNU_Compiler_Collection

Live comments

Programming Tools
Compilation process

- Pre-processor
- Compiler
- Linker
- Runtime Environment

Programming Tools
The Difference between Compiler and Interpreter

A computer itself can not understand the programmer's instructions. It must either be compiled into machine language using a compiler or processed at runtime using an interpreter.

Programming languages are designed that people can read and utilize them easily. However, to enable a processor to understand the individual instructions of a program, the source code must be put into machine code. Depending on the programming language, this happens using a compiler or using an interpreter.

Compiler

A compiler converts source code into machine language, translating the entire program from a programming language into machine code. The code is completely translated before the program is executed. Often an interims step happens and the source code is first converted into an intermediate code (e.g. object code) and then translated into the machine code.

The advantage of the intermediate code is, that it usually works on different platforms and can often be used by an interpreter. The compiler or assembler translates the intermediate result into machine code that is understood by the respective target system. Finally, an executable file is generated via linker. Modern programming languages often use bytecode instead of machine code, a kind of pseudocode for machines that runs internally in specialized virtual machines.

The compiler initiates several steps in order to create an executable program from the available source code. For this he needs comparatively more time and resources. However, once the compiled program is up and running, it is more efficient than interpreted software, since all instructions have already been fully translated into machine code.

Pure compiler languages are C, C ++ and Pascal.

Interpreter

An interpreter processes the source code at runtime. To do so, the interpreter proceeds line by line: A statement is read, analysed and executed immediately. Continuing with the next instruction until the end of the program is reached; or until an error occurs. An interpreter stops working as soon as something is going wrong. Therefore developers usually know where the problem is, immediately, and can fix problems faster.

An interpreter does not create a file that could be executed several times. He also does not translate the source code into machine code, but acts as an intermediate layer between the programming language and the machine. The interpreter analyses every single instruction of a program at runtime and calls the corresponding routine from its internal libraries, which executes the appropriate action on the target system.

Because the interpreter's work is done during execution time, and every statement has to be processed individually, interpreted programs are usually slower than compilations. For example, they also re-execute recurring instructions when they are called again.

Python, Perl, or BASIC are examples of languages that are using interpreters.

Just-in-time compiler: the hybrid solution

There are also approaches that combine compilers and interpreters to compensate for the weaknesses of the respective systems. The Compreter or just-in-time compiler translates the program into machine code only at runtime. On the one hand, the hybrid solution offers good performance of compiled programs, on the other hand it enables the convenient troubleshooting of interpreted programs. JIT compilers are used above all when creating platform-independent and portable software.

Examples of programming languages with JIT compilers are Java, Visual Basic, C # and also C ++ .NET.

Source: DevInsider

Programming Tools
Compilation process

- Loader
- Debugger

Programming Tools
Compilation process

```
/* file: hello.c */
#include <stdio.h>

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

comment

name of included file

preprocessor directive

return value's data type

list of input parameters

function name

function called by name

punctuation mark

list of input parameters

return value of function

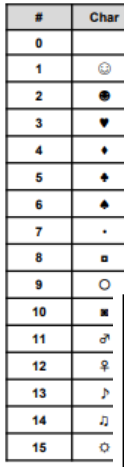
Programming Tools
Additional Information

ASCII Code (American Standard Code for Information Interchange)

- Definition: ASCII is a character encoding system that was originally developed in the 1960s to represent text in computers, communications equipment and other devices.

ASCII Code (128 characters)

Enhanced ASCII Code (256 characters)



#	Char	#	Char	#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
0		16	►	32		48	0	64	@	80	P	96	`	112	p
1	☺	17	◄	33	!	49	1	65	A	81	Q	97	a	113	q
2	☻	18	↕	34	"	50	2	66	B	82	R	98	b	114	r
3	♥	19	‼	35	#	51	3	67	C	83	S	99	c	115	s
4	♦	20	¶	36	$	52	4	68	D	84	T	100	d	116	t
5	♣	21	§	37	%	53	5	69	E	85	U	101	e	117	u
6	♠	22	▬	38	&	54	6	70	F	86	V	102	f	118	v
7	•	23	↨	39	'	55	7	71	G	87	W	103	g	119	w
8	◘	24	↑	40	(	56	8	72	H	88	X	104	h	120	x
9	○														
10	■														
11	♂														
12	♀														
13	♪														
14	♫														
15	☼														

ASCII Code Tab

#	Char	#	Char	#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
128	Ç	144	É	160	á	176	░	192	└	208	ð	224	Ó	240	
129	ü	145	æ	161	í	177	▒	193	┴	209	Ð	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	Ê	226	Ô	242	‗
131	â	147	ô	163	ú	179	│	195	├	211	Ë	227	Ò	243	¾
132	ä	148	ö	164	ñ	180	┤	196	─	212	È	228	õ	244	¶
133	à	149	ò	165	Ñ	181	Á	197	┼	213	ı	229	Õ	245	§
134	å	150	û	166	ª	182	Â	198	ã	214	Í	230	µ	246	÷
135	ç	151	ù	167	º	183	À	199	Ã	215	Î	231	þ	247	¸
136	ê	152	ÿ	168	¿	184	©	200	╚	216	Ï	232	Þ	248	°
137	ë	153	Ö	169	®	185	╣	201	╔	217	┘	233	Ú	249	¨
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Û	250	·
139	ï	155	ø	171	½	187	╗	203	╦	219	█	235	Ù	251	¹
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	ý	252	³
141	ì	157	Ø	173	¡	189	¢	205	═	221	¦	237	Ý	253	²
142	Ä	158	×	174	«	190	¥	206	╬	222	Ì	238	¯	254	■
143	Å	159	ƒ	175	»	191	┐	207	¤	223	▀	239	´	255	

Enhanced part of ASCII Table

Programming Tools
Additional Information

#	Char	#	Char	#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
0		16	►	32		48	0	64	@	80	P	96	`	112	p
1	☺	17	◄	33	!	49	1	65	A	81	Q	97	a	113	q
2	☻	18	↕	34	"	50	2	66	B	82	R	98	b	114	r
3	♥	19	‼	35	#	51	3	67	C	83	S	99	c	115	s
4	♦	20	¶	36	$	52	4	68	D	84	T	100	d	116	t
5	♣	21	§	37	%	53	5	69	E	85	U	101	e	117	u
6	♠	22	▬	38	&	54	6	70	F	86	V	102	f	118	v
7	•	23	↨	39	'	55	7	71	G	87	W	103	g	119	w
8	◘	24	↑	40	(	56	8	72	H	88	X	104	h	120	x
9	○	25	↓	41	)	57	9	73	I	89	Y	105	i	121	y
10	◙	26	→	42	*	58	:	74	J	90	Z	106	j	122	z
11	♂	27	←	43	+	59	;	75	K	91	[	107	k	123	{
12	♀	28	∟	44	,	60	<	76	L	92	\	108	l	124	|
13	♪	29	↔	45	-	61	=	77	M	93	]	109	m	125	}
14	♫	30	▲	46	.	62	>	78	N	94	^	110	n	126	~
15	☼	31	▼	47	/	63	?	79	O	95	_	111	o	127	⌂

ASCII Code Table

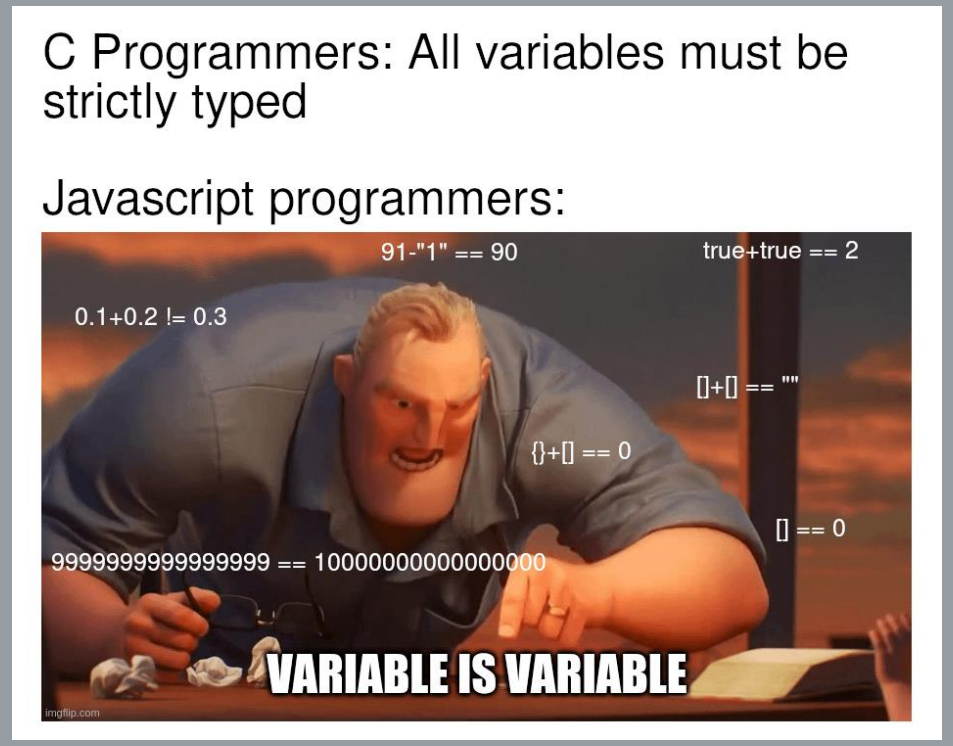
Programming Tools
Additional Information

#	Char	#	Char	#	Char	#	Char	#	Char	#	Char	#	Char	#	Char
128	Ç	144	É	160	á	176	░	192	└	208	ð	224	Ó	240	
129	ü	145	æ	161	í	177	▒	193	┴	209	Ð	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	Ê	226	Ô	242	‗
131	â	147	ô	163	ú	179	│	195	├	211	Ë	227	Ò	243	¾
132	ä	148	ö	164	ñ	180	┤	196	─	212	È	228	õ	244	¶
133	à	149	ò	165	Ñ	181	Á	197	┼	213	ı	229	Õ	245	§
134	å	150	û	166	ª	182	Â	198	ã	214	Í	230	µ	246	÷
135	ç	151	ù	167	º	183	À	199	Ã	215	Î	231	þ	247	¸
136	ê	152	ÿ	168	¿	184	©	200	╚	216	Ï	232	Þ	248	°
137	ë	153	Ö	169	®	185	╣	201	╔	217	┘	233	Ú	249	¨
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Û	250	·
139	ï	155	ø	171	½	187	╗	203	╦	219	█	235	Ù	251	¹
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	ý	252	³
141	ì	157	Ø	173	¡	189	¢	205	═	221	¦	237	Ý	253	²
142	Ä	158	×	174	«	190	¥	206	╬	222	Ì	238	¯	254	■
143	Å	159	ƒ	175	»	191	┐	207	¤	223	▀	239	´	255	

Enhanced part of ASCII Table

Variables and Constant Values

What makes a variable a variable?



Variables and Constant Values
What makes a variable a variable

a variable has four characteristics

- name
- data type
- value
- address

derived data types

- struct
- union
- pointer
- arrays

numeric data types

- char
- short
- int
- float
- double

Variables and Constant Values
Use of constants

In C there are two types constants:

- literal constants (magic numbers)
- symbolic constants

There are differences between symbolic and literal constants.

The syntax allows usage of constants or constant expressions.

Anywhere where the syntax allows constants or constant expressions you can use literal or symbolic constants.

```
printf("This is a Integer Literal DEC-style: %d\n\n", 42);
printf("This is a Integer Literal HEX-style: %d\n\n", 0x2a);
printf("This is a %s Literal.\n\n", "String");

printf("This is my first CONST VAL const-main-scope-style :) %d\n\n", CONST_UNSIGNED_VAL_1);
printf("This is my second CONST VAL const-main-scope-style :) %d\n\n", CONST_UNSIGNED_VAL_2);
printf("This is my first CONST VAL macro-style :) %f\n\n", PI);
```

Here we learn the possibilities. As a rule, no magic numbers appear in good code

Live coding

Variables and Constant Values
Symbolic Constants

Symbolic constants are defined by pre-processor keywords

#define PI 3.1415926

If a parameter variable is used as a literal constant, then you have to change each appearance of this parameter.

3.1415926 has a floating-point part therefore PI has the type float. PI is the symbolic constant

Type: const is a keyword in C that declares variables as constant.

This means that the value of a variable declared as const cannot be changed once it has been initialized.

```
printf("This is my first CONST VAL const-main-scope-style :) %d\n\n", CONST_UNSIGNED_VAL_1);
printf("This is my second CONST VAL const-main-scope-style :) %d\n\n", CONST_UNSIGNED_VAL_2);
printf("This is my first CONST VAL macro-style :) %f\n\n", PI);
```

Variables and Constant Values
Integer Constants

You can enforce an integer constant to be of a certain data type.

- 12445L → is of data type long
- 2234l → is of data type long, too
- 123ul → is of data type unsigned long

If the value is greater than the data type can accept the next appropriate data type is used implicitly.

Variables and Constant Values
Integer Constants

Some examples for floating point constants are:

- 300.0 300 in float
- 1E3 1000 in float
- 3.E2
- .55E-3

The first part of the scientific notation (before E) is Mantissa the second part (after the E) is the exponent.

Live coding

Variables and Constant Values
Enumeration Constants

One example for enumeration constants is TRUE and FALSE.

```
enum boolean {FALSE, TRUE};
```

The first value in enumerations has the numeric value 0 (zero) the next one 1, then 2, and so on …

```
TRUE +1
```

The data type of the value TRUE (and FALSE, too) is int.

```
printf("It is funny, because it is %s.\n\n", c == 1? "true" : "false");
```

Variables and Constant Values
Constant of String Type

Character constants have only one character in it, indicated by single quotation marks.

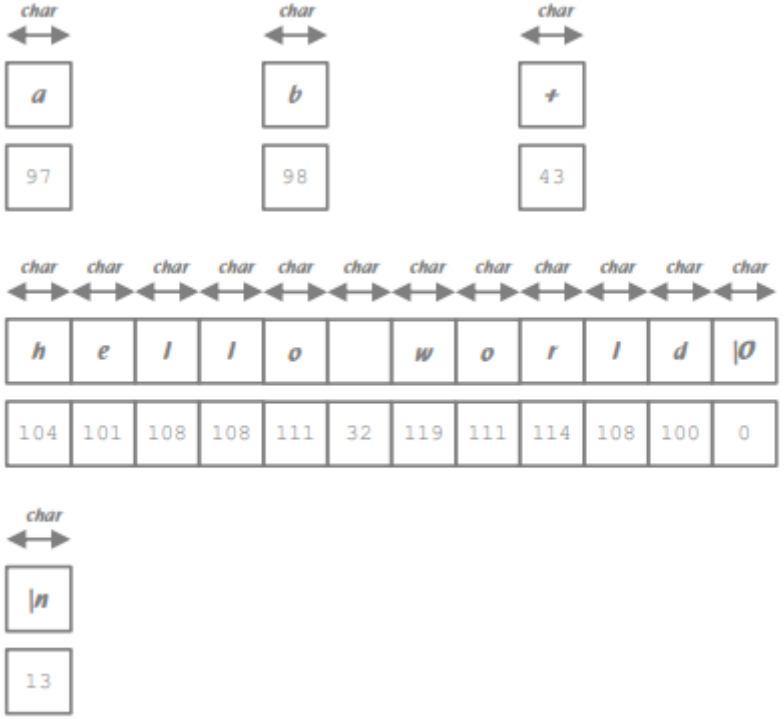
‘a’ ‘b’ ‘+’

Constants of string type mean more than one character, indicated by quotation marks

“hello world!\n”

Escape sequences are alternative Representations

‘\n’ one character with LF meaning



~~Variables and Constant Values~~ Comments

Comments are part of the program code

Comments are not a substitute for bad code

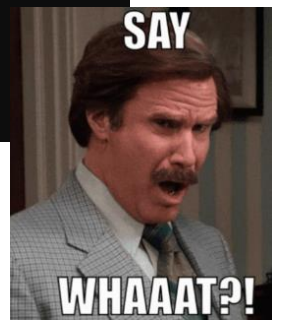
- Well-written code should be self-explanatory as much as possible.
- Comments should only be used when necessary, particularly to explain complex logic.

Comments should explain the "why", **not the "what“**

- Comments should describe why something is done, not what the code does.
- The "what" should be evident from the code itself.

```c
main.c > calcGcdEuklid(int, int)
#include <stdio.h>

// Function: calcGcdEuklid
// Description: Implements the Euclidean algorithm to calculate the greatest common divisor (GCD).
// Parameters:
// - a: First integer input.
// - b: Second integer input.
// Returns: The GCD of the two integers.
int calcGcdEuklid(int a, int b)
{
```

```c
// This is a loop that iterates 10 times
for (int i = 0; i < 10; i++) {
    ...
}
```



Variables and Constant Values
Variable Definition in your code

```
int i;
```

Variables are defined at the beginning of the program code

- Outside main if a variable is global
- At the beginning of the function if it is local

A defined variable is not initialized with a dedicated value. A defined variable contains any value (random value).

A variable can be initialized at definition time.

Multiple variables can be define in one line if of same type separated by a comma.

Example Program Code

```
#include <stdio.h>

int main (void)
{
   int i;

   int x=0;              // explicit initialization

   char a, b;            // two variables

   float f1=0.5, f2;     // two variables, one explicitly
                         // initialized

   return (0);
}
```

Variables and Constant Values
Local and Global Variables

```
int i;
```

A variables can be defined a global or local variable.

It depends on the place where the variable is defined whether it is global or local.

- global – outside the functions
- local – inside the functions

Global variables are available and visible during the whole program and from all functions.

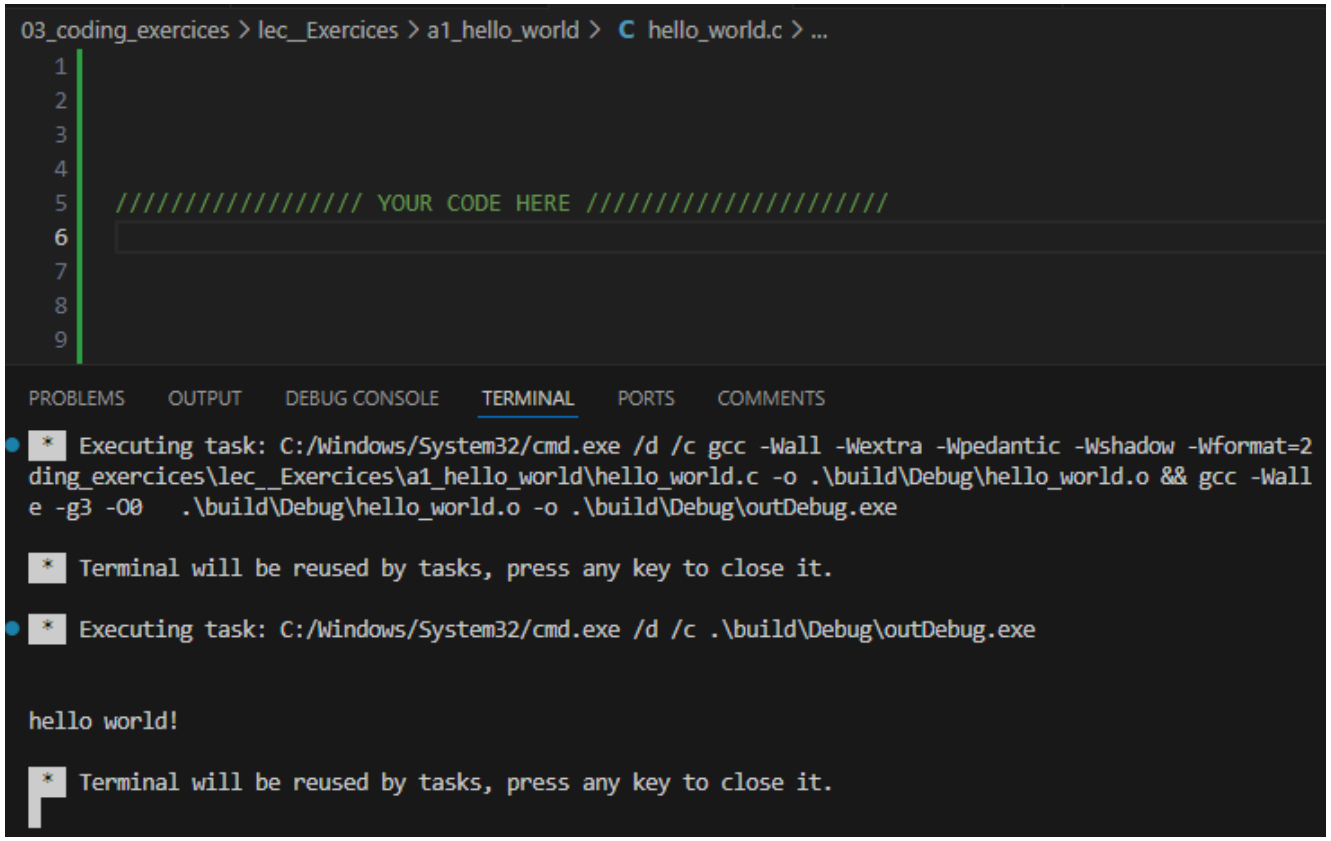
Local variables are visible for the function in which defined.

Local variables are alive as long the function (block) is not finished in which the variable is defined.

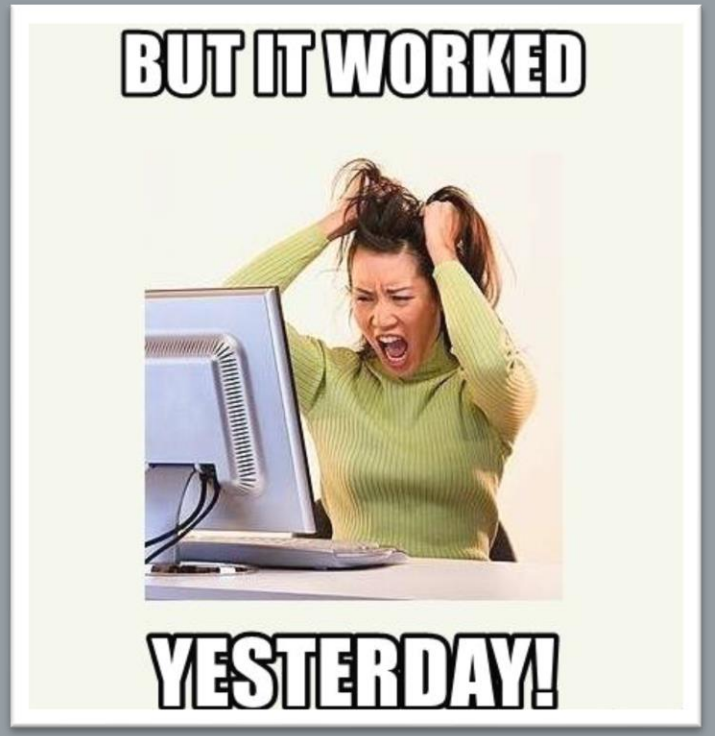
Example Program Code

```
#include <stdio.h>
int a = 1;                  // global variable;
int main (void)
{
   int i=0:                 // local variable
   printf ("a has the value %d\n", a);
   printf ("i has the value %d\n", i);
   return (0);
}
```

Let's code


03_coding_exercices > lec__Exercices > 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_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.

Version management with GIT



Version management with GIT

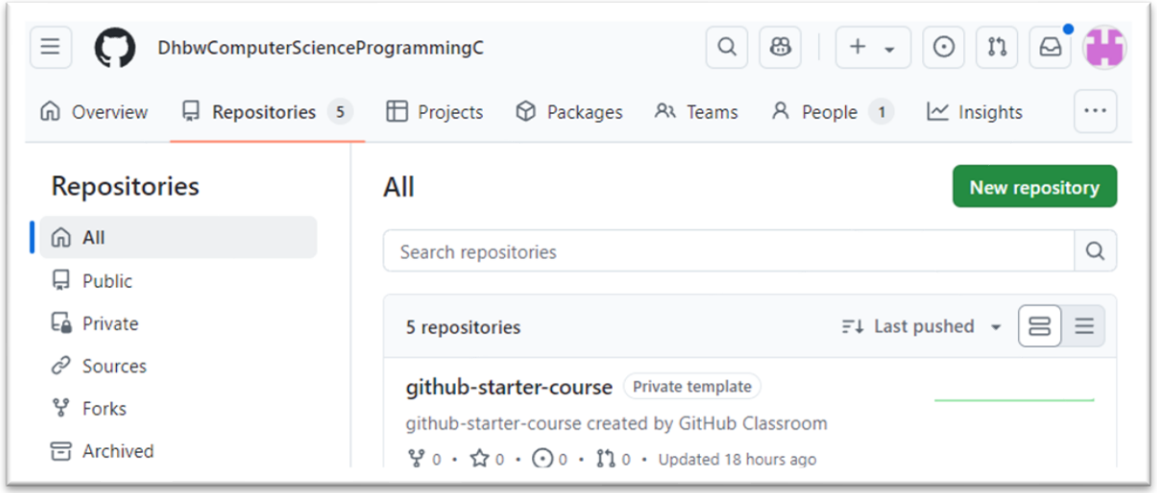
Open-source version control software to track changes

Provides possibility ...

- Revert to any saved old version
- Visualize the difference between two versions
- Simultaneous work on the same code with handling of conflicts etc.

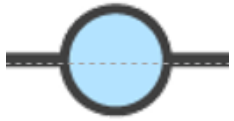
In this course:

- First experiences with Git
- Regular backup copies
- Working on a projects in GitHub classroom

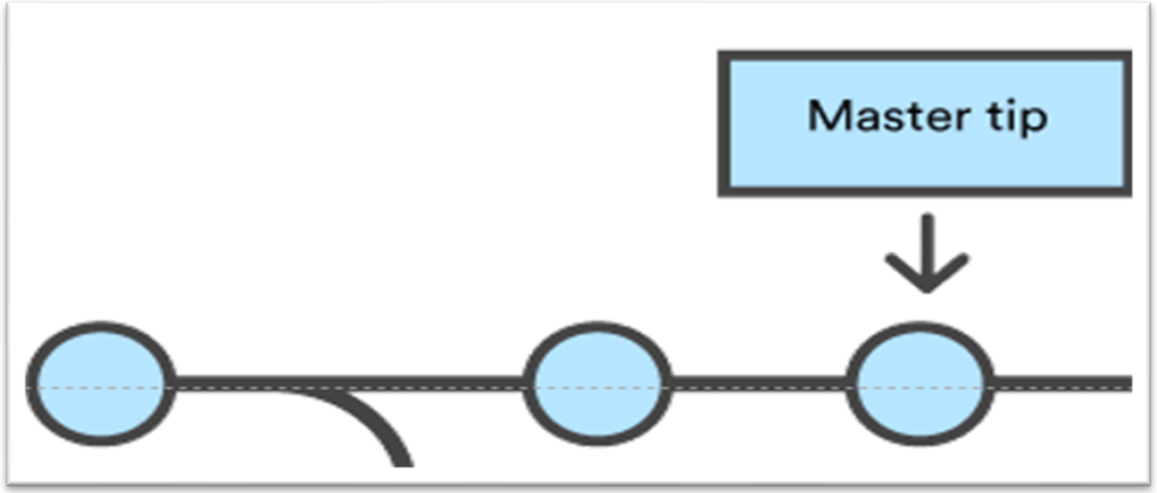


https://github.com/

Version management with GIT
Git - Commits

Visual representation Commit: 

- collection of changes "registered" in the Git repository
  - + author
  - + timestamp
  - + commit message
- Corresponds to a "snapshot" of the project codes



https://github.com

Live demo

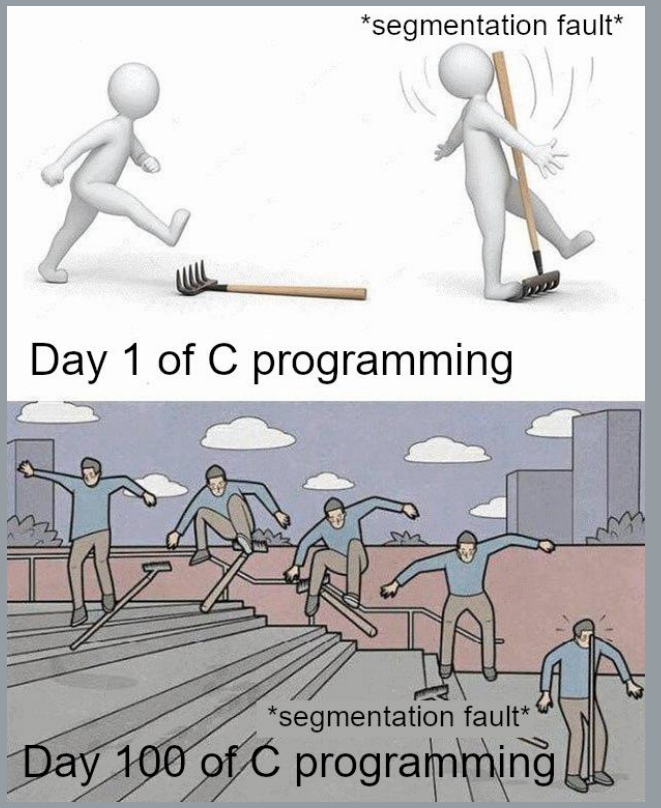
Version management with GIT
Start your GitHub journey

- Create an account
- Login and create your first Repository
- Setup VS Code with your GitHub account
- Clone your Repository and add your “Hello World” code
- Commit your changes and push

https://www.zbw-mediatalk.eu/de/2015/09/github-und-social-coding-neue-formen-der-softwareentwicklung-und-distribution-als-chance/

Preferences / Attributes of C

C style?



Preferences / Attributes of C

C has an imperative style

Imperative programming languages are:

- closest to the architecture — **assembler**
- procedural — FORTRAN, Pascal, **C**
- object oriented — Smalltalk, Java, **C++**

Use of functions/procedures:

- In C, programs are structured with the help of functions/procedures that process data (variables, arrays, structures).
- The procedures contain an ordered list of instructions to be executed and can be called at any time during execution

What is the imperative paradigm?

Preferences / Attributes of C

More characteristics of C

- **C is applicable** for common applications
- **efficient** way of putting algorithms in place
- sufficient number of **control structures**
- many **data types**
- powerful amount of **operators**
- operators are not applicable for complex objects (e.g. strings)
  exception: structures
- no instructions for input and output
- great portability (there may be other languages with higher portability)
- modular programming (modular compiling)
- **program code is very close to the architecture**

Preferences / Attributes of C

And some more characteristics of C

Names

- internal: only valid within one file
  (names of macros at the preprocessor section are internal, too)
- external: names of variables and functions that are valid for more than one file
- 31 characters are important for internal
- 6 characters can be used for external

Reserved Keywords (ISO standard: 32 keywords)

- auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while

Preferences / Attributes of C

- Q0 - What are the most important features of C?
- Q1 - What are the main preferences and characteristics of the C programming language?
- Q2 – What are the characteristics of C as a procedural programming language?

Lecture 00

PART I

- Introduction of C
- Algorithms and programs
- Programming Tools
- Variables and Constant Values
- Version management with GIT
- Preferences / Attributes of C

PART II

- **Operators, Operations and Functions**
- **Classes of Data Types**
- **Control Structures**
- **Input / Output std functions**
- **GitHub assignment 00**

Operators, Operations and Functions

What are Operators?

Operators, Operations and Functions
The associativity of operators

Definition:

- Operators are applied to Operands
- Operators and Operands form Operations
- Every Operation has a Result
- An Operation generates a Value

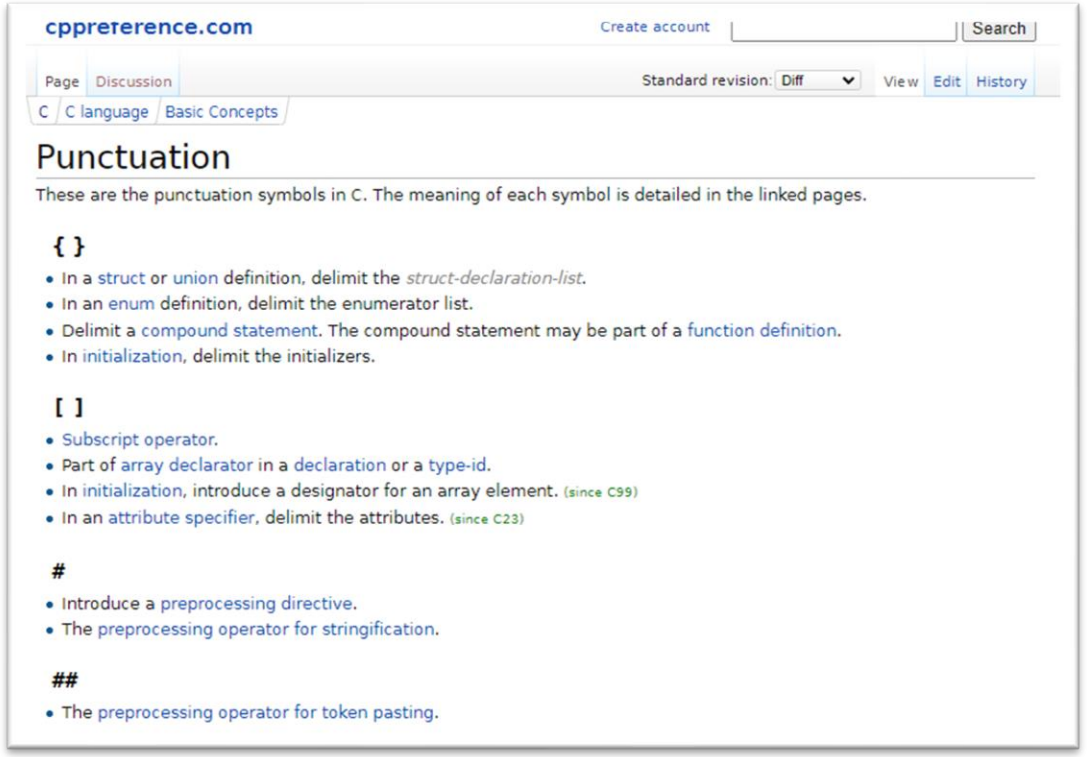
Definition:

- The associativity of an operator is a property that defines how operators of the same priority level are grouped.
- This definition is important when parenthesis are not used.

Priority	Operator	associativity
1	() [] -> .	left
2	! ~ ++ -- - (data type) * & sizeof	right
3	* / %	left
4	+ -	left
5	<< >>	left
6	< <= > >=	left
7	== !=	left
8	&	left
9	^	left
10	|	left
11	&&	left
12	||	left
13	? :	right
14	= += -= *= /= %= &= ^= |= <<= >>=	right
15	,	left

Operators, Operations and Functions
Punctuation Marks

- defining the syntax and structure of the code
- help separate code elements, control flow, and organize data
- essential to clear and correct communication with the compiler



https://en.cppreference.com/w/c/language/punctuators

Operators, Operations and Functions
Punctuation Marks

Semicolon (;):	Marks the end of a statement.
Comma (,): elements in	Separates variables in a declaration, function arguments, or initialization lists.
Parentheses (()): function	Enclose conditions for control structures (like if, while, etc.) and arguments.
Curly braces ({}):	Define blocks of code, like the body of functions, loops, and conditional statements.
Square brackets ([]):	Used for array indexing and declaration.
Colon (:):	Primarily used in ternary operators and switch statements.
Asterisk (*):	Indicates a pointer or is used in multiplication.
Ampersand (&):	Refers to the address of a variable.
Double slash (//):	Denotes a single-line comment.
Slash asterisk (/*...*/):	Used for multi-line comments.

Operators, Operations and Functions
C only has functions

- There aren't any procedures
- A function can call another function
- A function cannot be defined within another function
- A function can call itself
- A function has a return value
  (This is the major characteristic of a function)

Functions are defined using the following syntax:

```c
int foo(int bar) {
    /* do something */
    return bar * 2;
}

int main() {
    foo(1);
}
```

https://www.learn-c.org/en/Functions

Operators, Operations and Functions
Program main

- The main function is the MAIN function in every program
- The main function is executed first
- The main function is started at the program's start
- You need at least the main function
- Without a main function you are programming a library

Functions are defined using the following syntax:

```c
int foo(int bar) {
    /* do something */
    return bar * 2;
}

int main() {
    foo(1);
}
```

https://www.learn-c.org/en/Functions

Homework

Operators, Operations and Functions

What is the return value of the following expressions?

- `15 % 4`
- `15 / 2.0`
- `3 + 5 % 4`
- `3 * 7 % 4`

How would you order the operators and operands to be executed?
Set parenthesis around.

- `x = - 4 * ++ i - 6 % 4;`

Which of the following expressions is true which is false? With `x = 7;`

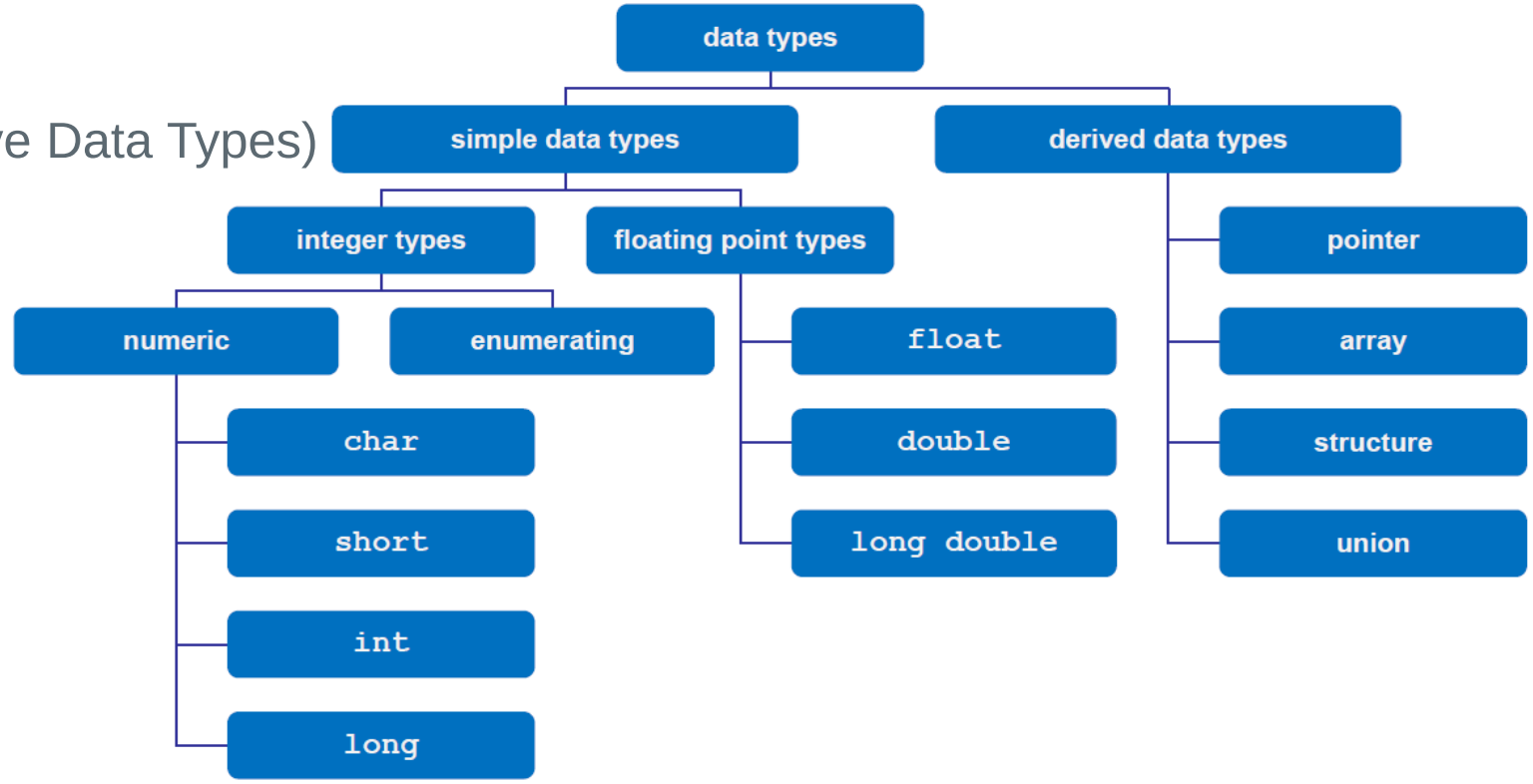
- `x < 9 && x >= -5;`
- `!x && x >= 3;`

Classes of Data Types

What Data Type Classes do you know?

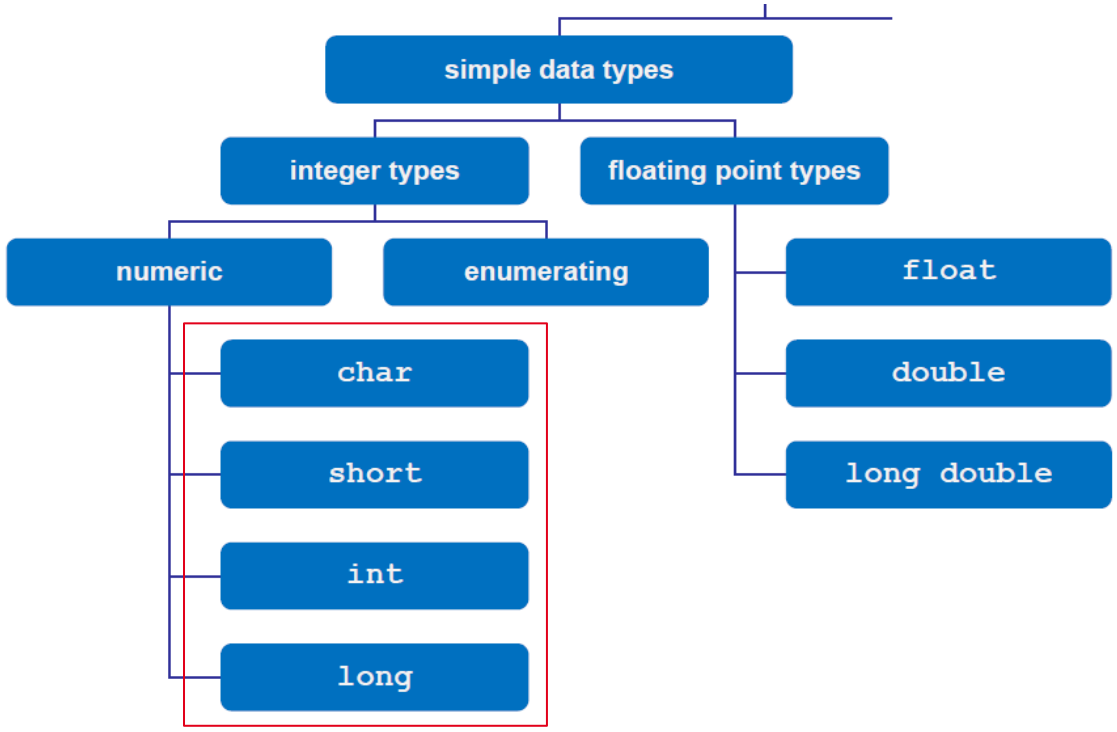
Classes of Data Types
Overview

(Primitive Data Types)



Classes of Data Types
Primitive Data Types – numeric integer types

- char: Used for storing characters, typically one byte.
- short: Used for storing smaller integers, typically 16 bits.
- int: Used for storing standard integers, typically 32 bits.
- long: Used for storing larger integers, typically 32 or 64 bits.



Classes of Data Types

Primitive Data Types – numeric integer types

Data Type	Description	Size	Range
char	Stores a single character	1 byte	-128 to 127 (signed) or 0 to 255 (unsigned)
short	Stores smaller integers	2 bytes	-32,768 to 32,767 (signed)
int	Standard integer type	4 bytes	-2,147,483,648 to 2,147,483,647 (signed)
long	Stores larger integers	4 or 8 bytes	Varies by architecture

Classes of Data Types
What are Modifiers?

- Modifiers are keywords in C that are used to change or extend the properties of a data type.
- They allow you to adjust the size and behavior of primitive data types.

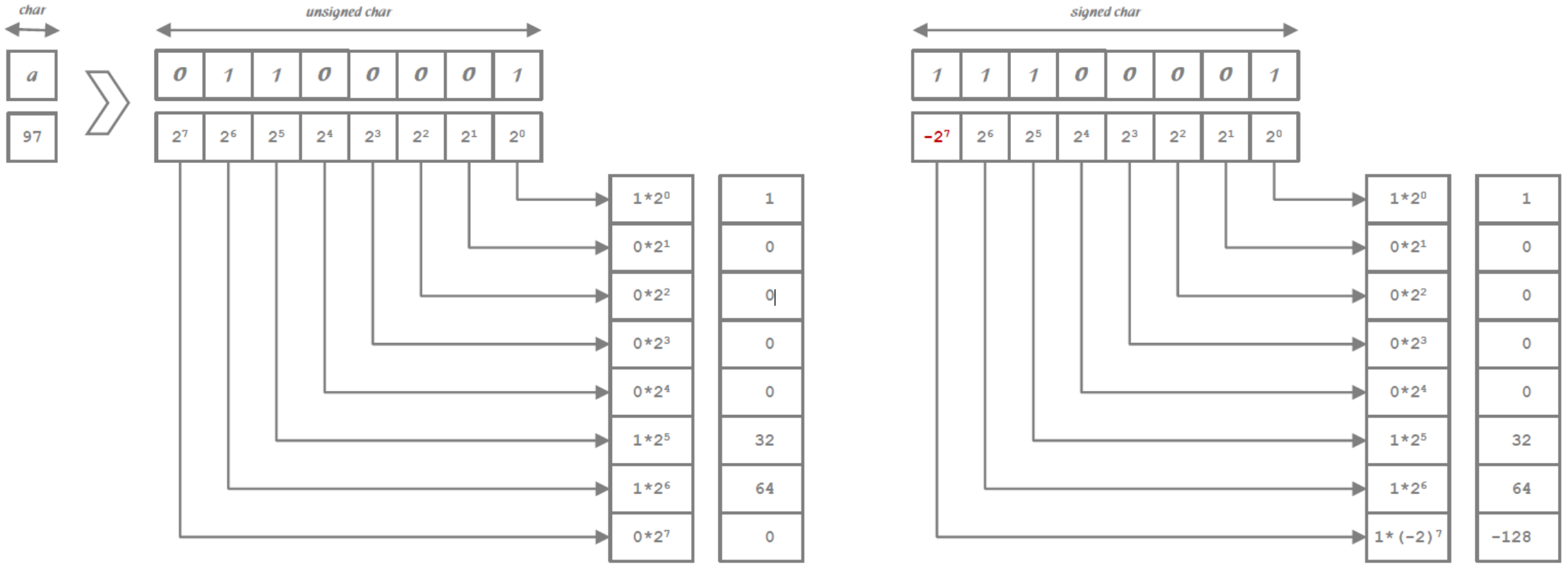
Modifier	Description	Example
signed	Allows negative values. Default for int and char.	`signed int a;`
unsigned	Allows only non-negative values, doubling the value range.	`unsigned int b;`
long	Increases size, typically to 4 or 8 bytes.	`long d;`
long long	Ensures at least 8 bytes for very large integers.	`long long e;`
long double	long double	`long double`

Classes of Data Types
Ranges of data types with modifiers

		range	
	length	min. value	max. value
signed char	1 byte	-128	+ 127
unsigned char		0	+ 255
signed short	2 bytes	- 32.768	+ 32.767
unsigned short		0	+ 65.535
signed int	2 bytes	-32.768	+32.767
	4 bytes	-2.147.483.648	+2.147.483.647
unsigned int	2 bytes	0	+65.535
	4bytes	0	+4.294.967.295
signed long	4 bytes	-2.147.483.648	+2.147.483.647
unsigned long		0	+4.294.967.295

Classes of Data Types

Example: Data type char (unsigned char)



Classes of Data Types

Example: Ranges of data types with modifiers

Transform into binary.

Assume `char` as data type

3	
15	
-119	
0	
42	

Transform into binary.

Assume `singed short` as data type with 2 bytes memory space

16866	
x	
-99	
13,5	
-42	

Transform into decimal.

Assume `singed short` as data type with 2 bytes memory space

0000 0011 0000 0000	
0000 0110 1010 1100	
1101 1010 0000 1010	
1000 0000 0000 0001	
1111 1111 1111 1111	

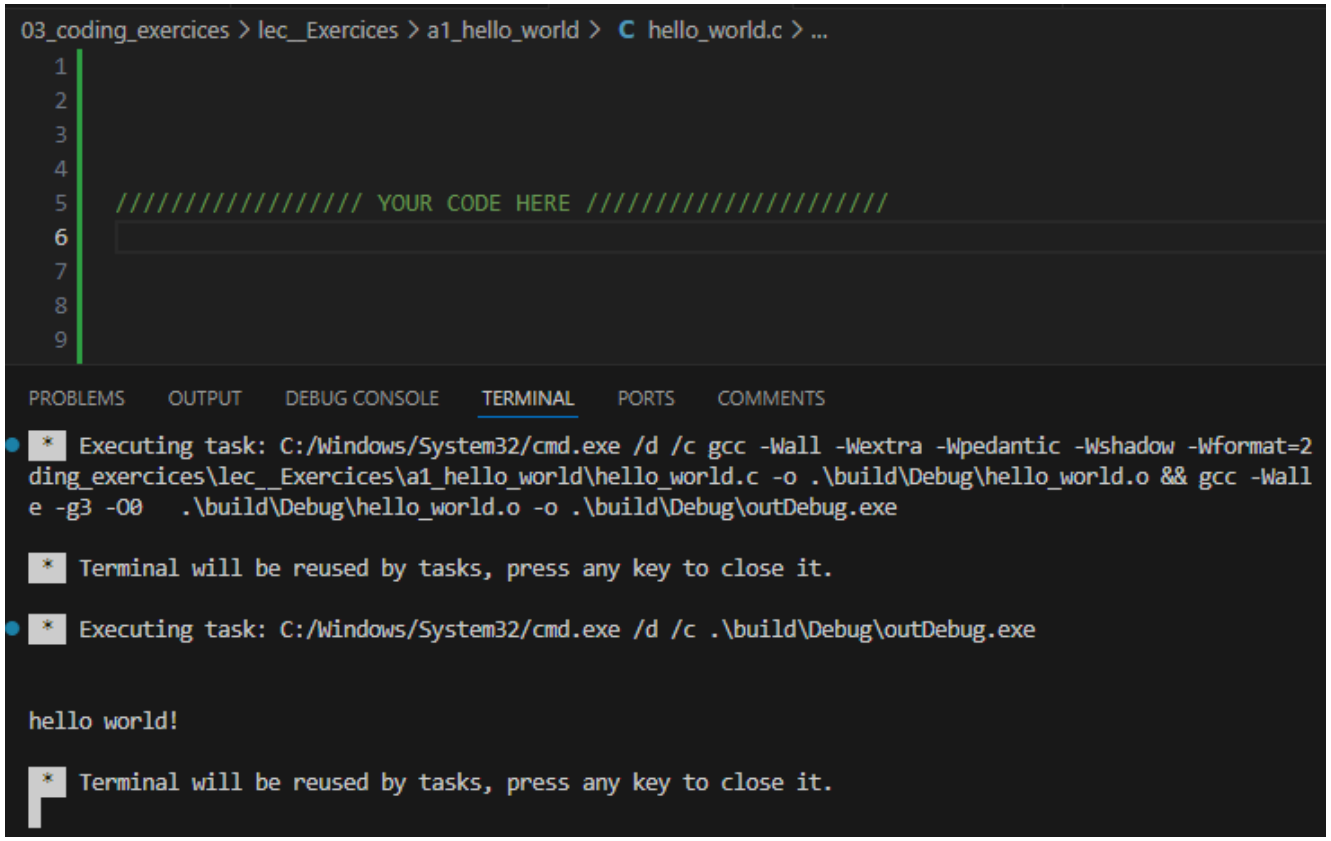
Classes of Data Types
What is enum?

- enum (short for enumeration) is a user-defined data type in C that consists of a set of named integer constants.
- It allows you to create a variable that can hold a set of predefined constants, enhancing the readability and maintainability of the code

```c
enum Days {
    SUNDAY,    // 0
    MONDAY,    // 1
    TUESDAY,   // 2
    WEDNESDAY, // 3
    THURSDAY,  // 4
    FRIDAY,    // 5
    SATURDAY   // 6
};
```

```c
int main() {
    enum Days today;

    today = TUESDAY;

    if (today == TUESDAY) {
        printf("Today is Tuesday!\n");
    } else {
        printf("Today is not Tuesday.\n");
    }

    return 0;
}
```

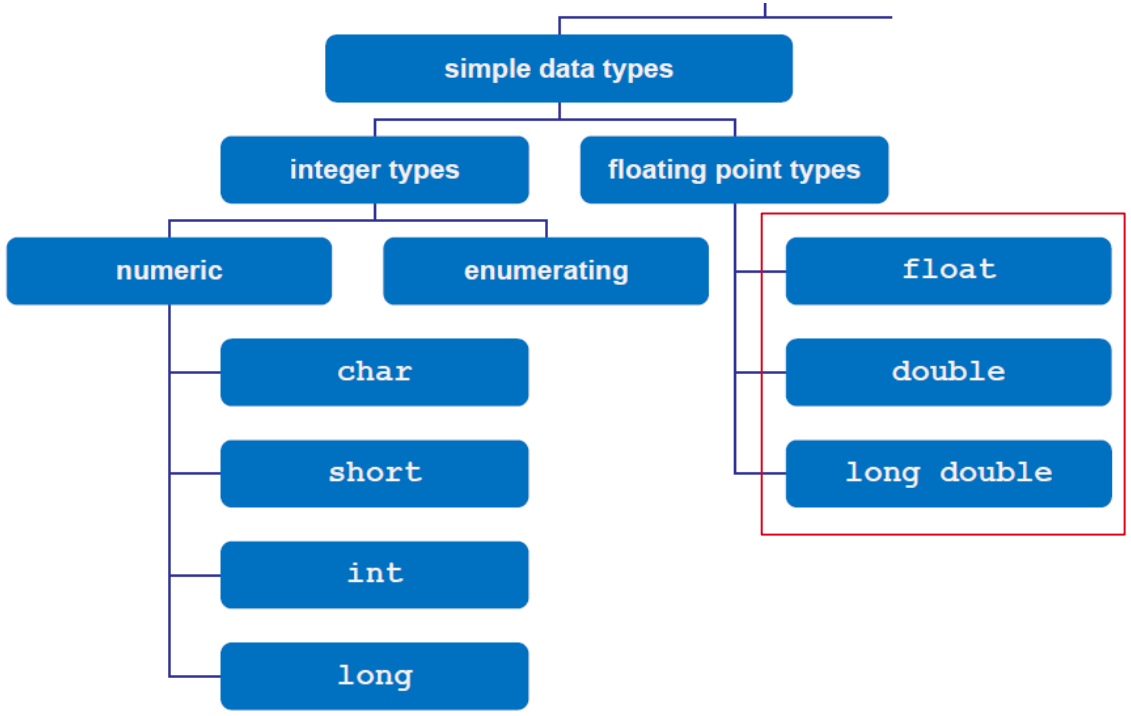
Let's code


03_coding_exercices > lec__Exercices > 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_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.

Classes of Data Types
Primitive Data Types

- float: Used for storing single-precision floating-point numbers.
- double: Used for storing double-precision floating-point numbers.
- long double: Used for storing extended-precision floating-point numbers.

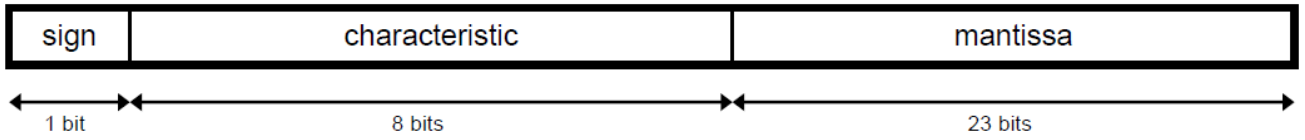
```
long double pi = 3.14159265358979323846264338327 9L;
```



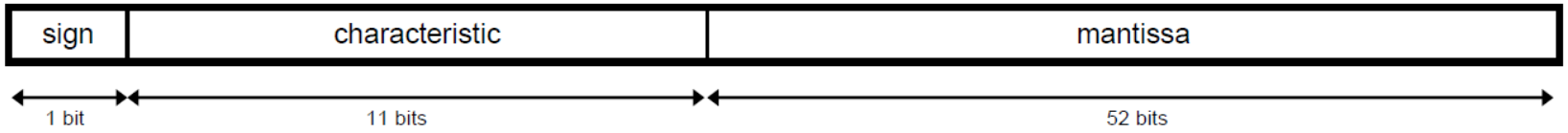
Classes of Data Types
Primitive Data Types – floating point types

Data Type	Description	Size	Range
float	Single-precision floating-point numbers	4 bytes	Approximately 3.4E-38 to 3.4E+38
double	Double-precision floating-point numbers	8 bytes	Approximately 1.7E-308 to 1.7E+308
long double	long double	long double	long double

`float`



`double`



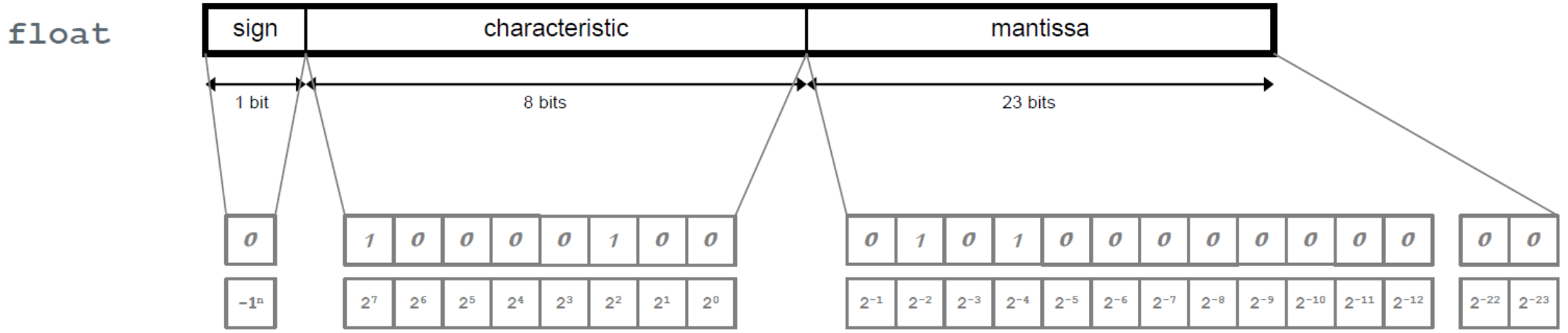
Classes of Data Types
Introduction to Characteristic and Mantissa in Floating-Point Representation

Characteristic (Exponent)

- is the exponent part that shifts the decimal point of the mantissa

Mantissa (Significand)

- represents the significant digits of a floating-point number



Classes of Data Types
Calculation Formula for Floating Point Numbers

$$value = (-1)^{sign} \bullet (1+Mantissa) \bullet 2^{(characteristic - BIAS)}$$

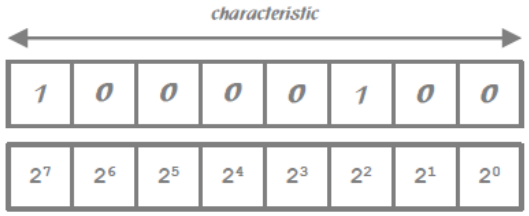
Characteristic and BIAS

Take the characteristic as an unsigned decimal number with eight bits.

The BIAS is depending on the data type:

- float: BIAS is `127`
- double: BIAS is `1023`

Using the BIAS the characteristic represents values for the exponent that covers a range from `-127` up to `+128` (for float) and `-1023` up to `+1024` (for double).



The binary number of the characteristic is `132`.

Taking the BIAS into account the value for the exponent is:

exponent = `132 - 127 = 5`

Classes of Data Types

Calculation Formula for Floating Point Numbers

$$value = (-1)^{sign} \bullet (1+Mantissa) \bullet 2^{(characteristic - BIAS)}$$

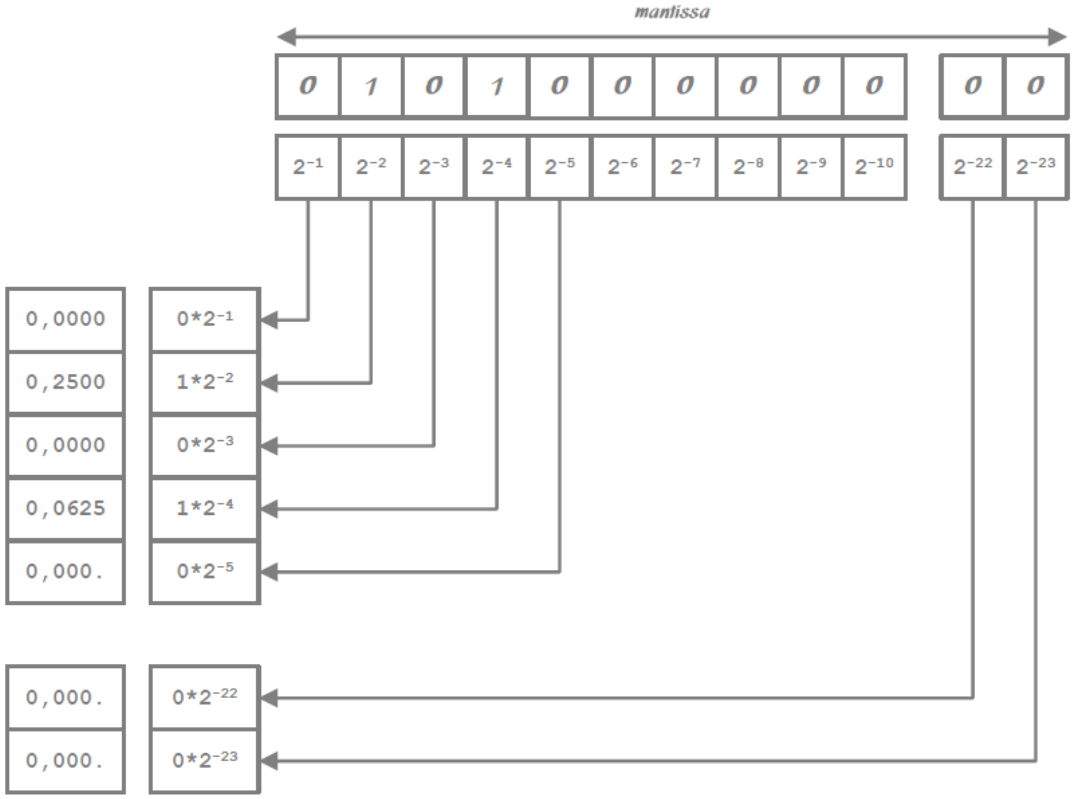
Mantissa

The calculation of the mantissa's value is a straight forward process.

```
  0,0000
+ 0,2500
+ 0,0625
= 0,3125
```

The mantissa only represents the fraction and the decimal (value `1`) part is implicitly stored.

The factor for the formula is `1,3125`.



Classes of Data Types

Calculation Formula for Floating Point Numbers

$$value = (-1)^{sign} \bullet (1+Mantissa) \bullet 2^{(characteristic - BIAS)}$$

Sign Bit and final Calculation

The sign bit in the data type float as well as double is a „real“ sign bit. If 0 the number is positive, if 1 the number is negative.

Let’s summarize the parts:

```
Characteristic  = 132
Exponent        = 5
Mantissa        = 0,3125
```

Finally, the value of the float number is calculated:

$(-1)^0 * (1+0,3125) * 2^5 = 42.0$

Classes of Data Types

Calculation Formula for Floating Point Numbers

$$value = (-1)^{sign} \bullet (1+Mantissa) \bullet 2^{(characteristic - BIAS)}$$

Range

The **range** is mainly determined by the **exponent** and therefore by the **characteristic**.

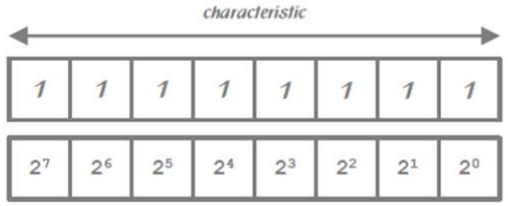
The characteristic hosts 8 bit in the data type float and 11 bit for the data type double.

The highest number of the characteristic is also representing the highest exponent for a float number.

float	characteristic = 255	exponent = 127
double	characteristic = 2.048	exponent = 1.023

The number e.g. in `float` is between

$$-(2^{127}) \ldots 2^{127} \rightarrow -(10^{38}) \ldots 10^{38}$$



Taking a simplified approach, let's agree that

$$2^{10} = 1.024 \approx 1.000 = 10^3$$

$$2^{127} = \underbrace{2^{10} * 2^{10} * 2^{10} * 2^{10} * \ldots * 2^{10} * 2^{10}}_{12 \text{ times}} * 2^7$$

$$\rightarrow 2^{127} \approx \underbrace{10^3 * 10^3 * 10^3 * 10^3 * \ldots 10^3 * 10^3}_{12 \text{ times}} * 10^2$$

$$\rightarrow 2^{127} \approx 10^{38}$$

Classes of Data Types
Calculation Formula for Floating Point Numbers

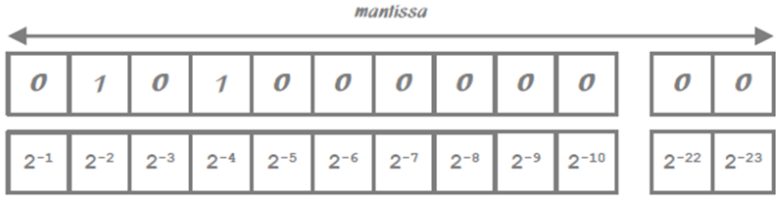
$$value = (-1)^{sign} \bullet (1+Mantissa) \bullet 2^{(characteristic - BIAS)}$$

Accuracy

The **mantissa** is that element that controls the **accuracy** of the floating point number. The more bit are spent for the mantissa the more precise is the value hosted by the variable.

In float there are 23 bits available for the mantissa, double uses 52 bits.

The least significant bit has the value 2^{-23} (here data type `float`).



Taking the same simplified approach:

$$2^{10} = 1.024 \approx 1.000 = 10^3$$

$$2^{-23} = 2^{-10} * 2^{-10} * 2^{-3}$$

$$\rightarrow \quad 2^{-23} \approx 10^{-3} * 10^{-3} * 10^{-1}$$

$$\rightarrow \quad 2^{-23} \approx 10^{-7}$$

The accuracy of float is 10^{-7}, 7 decimal digits.

Classes of Data Types

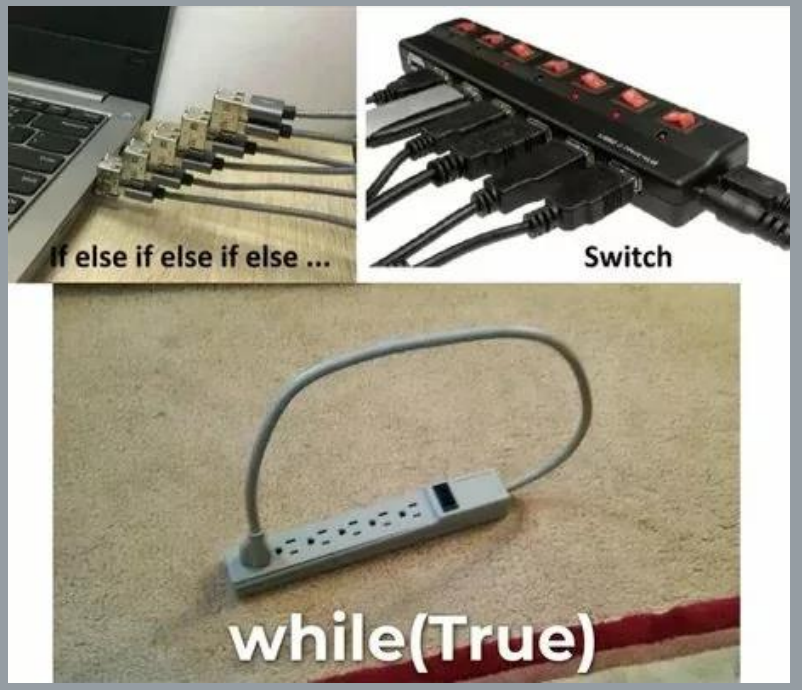
Calculation Formula for Floating Point Numbers

$$value = (-1)^{sign} \bullet (1+Mantissa) \bullet 2^{(characteristic - BIAS)}$$

Convert the following numbers from decimal into binary using the float format.

```
4711,0

-1226,3

1,2

-7,0

17,38
```

Control Structures



Control Structures
Sequence

```
statement;
```

A statement ends with a semicolon.

Multiple statements form a sequence.

Example Program Code

```
#include <stdio.h>

void main (void)
{
   printf ("hello world !\n");
   return (0);
}
```

Control Structures
Selection

```
if (condition) {if_statement} else {else_statement}
```

The simple selection checks a condition.

The if clause is mandatory. The else clause is optional.

Each clause is included in braces. Braces are not mandatory if the clause exists of only one statement.

Example Program Code

```
void main (void)
{
   int x=0, y=1;
   if (x < y)
   {
         y = y - x;     // executed if condition is true
   }
      else
   {
         x = x - y;     // executed if condition is false
   }
}
```

Control Structures
Multiple Selection

```
switch (value) {case n: statement; break;}
```

The multiple selection with case check a value.

The selection exit is based on case statements.

Each case statement ends with a break.

The last case of a multiple selection is the default case. The default case is executed if no other case statement is valid.

Example program code

```
int main (void)
{
   int a=2;

   switch(a) {
   case 1:  printf("a is one\n");
            break;
   case 2:  printf("a is two\n");
            break;
   case 3:  printf("a is three\n");
            break;
   default: printf("a is any value\n");
            break;

  }

   return (0);

}
```

Control Structures
Iteration

```
for (initialization; condition; iteration) {statement;}
```

`for` is a head controlled loop.

Initialization, condition and iteration statements are part of the loop's head.

The loop is executed as long as the condition is true.

The condition is checked first.

The body of the loop is a block and enclosed by braces.

Example Program Code

```
int main (void)
{
   int a;

   for (a=50; a>0; a--)
   {
      printf("a has the value %d\n", a);
   }

   return (0);
}
```

Control Structures
Iteration

```
do {statement;} while (condition);
```

`do-while` is a foot controlled loop.

The function's body condition is executed at least once.

The loop is executed as long as the condition is true.

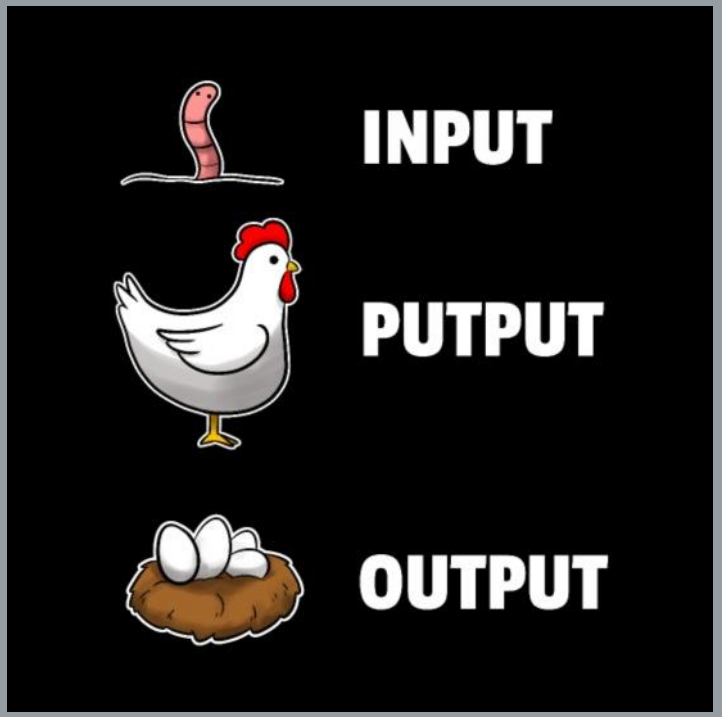
The body of the loop is a block and enclosed by braces.

An iteration statement must be part of the body.

Example Program Code

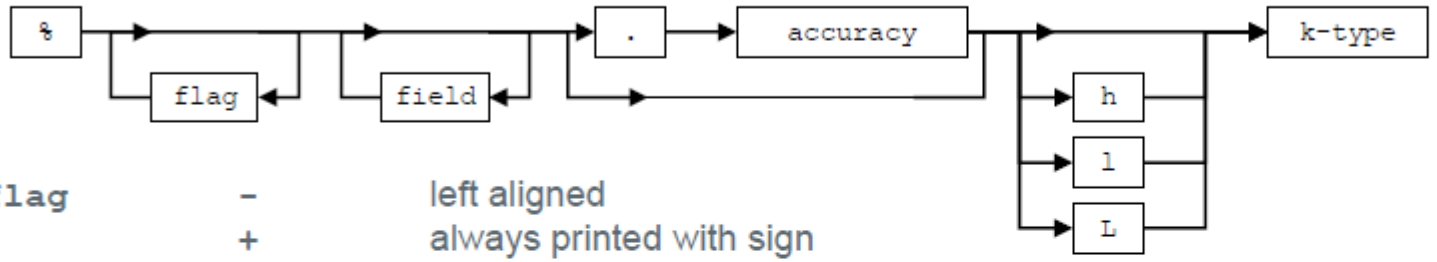
```
int main (void)
{
   int a=0;
   do
   {
      printf("a has the value %d\n", a);
      a++;                  // iteration statement
   } while (a<50);
   return (0);
}
```

Input / Output std functions



Input / Output std functions
Formatted Output

```
int printf (const char * format, …);
```



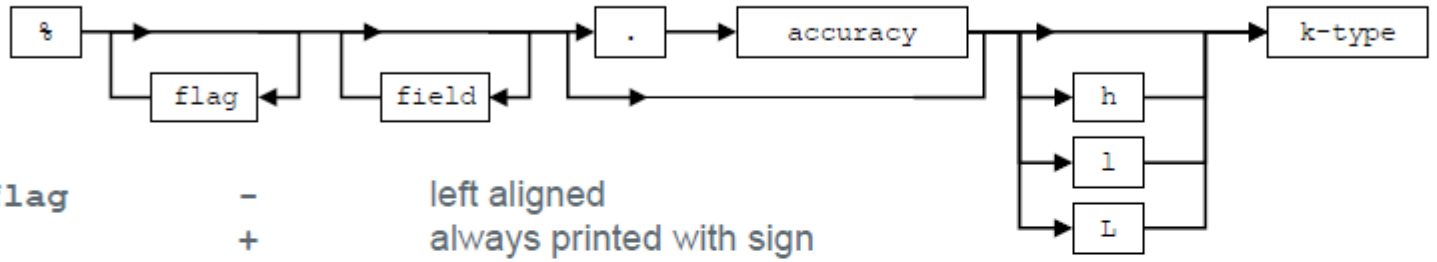
flag	-	left aligned
	+	always printed with sign
	blank	blank
	0	number printed with leading zeros
	#	number printed with system indicator octal with leading zero, hex with 0x or 0X

field	minimal length of the number field
accuracy	minimal number of digits

https://cplusplus.com/reference/cstdio/printf/

Input / Output std functions
Formatted Output

```
int printf (const char * format, …);
```



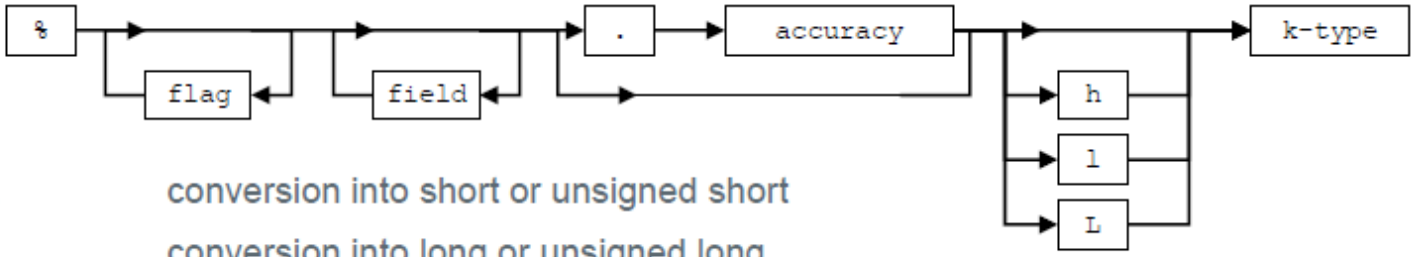
flag	-	left aligned
	+	always printed with sign
	blank	blank
	0	number printed with leading zeros
	#	number printed with system indicator octal with leading zero, hex with 0x or 0X

field	minimal length of the number field
accuracy	minimal number of digits

https://cplusplus.com/reference/cstdio/printf/

Input / Output std functions
Formatted Output

```
int printf (const char * format, …);
```



`h`	conversion into short or unsigned short
`l`	conversion into long or unsigned long
`L`	formatted output of floating point numbers

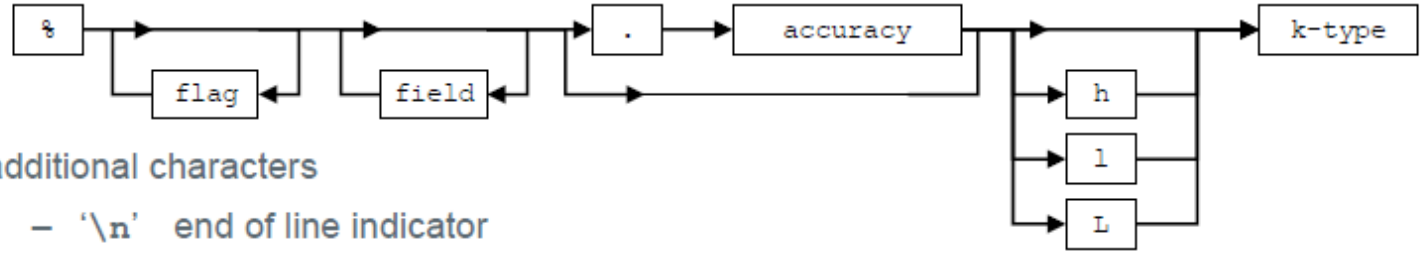
`k-type` character for the format of the argument

`d,i`	int	decimal number
`o`	int	octal number (without sign character)
`x`	int	hexadecimal number with small characters
`X`	int	hexadecimal number with capital characters
`u`	int	decimal number without sign character
`c`	int	single character (unsigned char)
`f`	float	floating point number
`%`	-	to print the character '%'

https://cplusplus.com/reference/cstdio/printf/

Input / Output std functions
Formatted Output

```
int printf (const char * format, …);
```



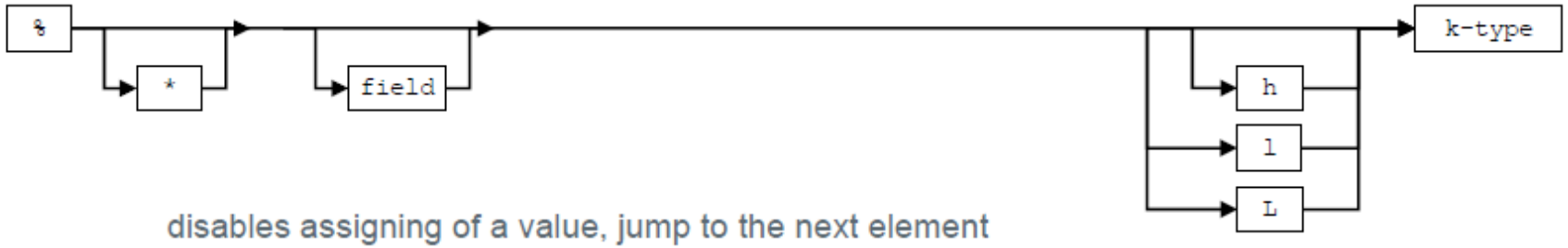
additional characters

- '\n' end of line indicator
- '\t' tab stop
- '\v' vertical tab stop
- '\r' carriage return
- '\f' line feed / new line
- '\b' bell
- '\\' print a \
- '\'' print a '
- '\"' print a "

https://cplusplus.com/reference/cstdio/printf/

Input / Output std functions
Formatted Input

```
int scanf(const char * format, …);
```



`*`	disables assigning of a value, jump to the next element
`field`	maximum length of the field
`h`	with k-type d, i, n it's a pointer to a short variable with k-type o, u, x it's a pointer to a unsigned short variable
`l`	with k-type d, i, n it's a pointer to a long variable with k-type o, u, x it's a pointer to a unsigned long variable
`L`	formatted input of a floating point number

https://cplusplus.com/reference/cstdio/scanf/

Input / Output std functions
Formatted Input

```
int scanf(const char * format, …);
```

k-type	character for the format of the argument	
d	int *	decimal number
l	int *	integer number
u	unsigned *	decimal number without sign character
o	unsigned *	octal number
x	unsigned *	hexadecimal number
c	char *	single character
n	int *	no input, the number of read characters by scanf will be assigned to the pointer's address

https://cplusplus.com/reference/cstdio/scanf/

Input / Output std functions
Unformatted Input

```
int getc (FILE * stream);
```

The `getc` function reads a simple character from the current stream position and advances the stream position to the next character. The `getchar ()` function is identical to `getc (stdin)`. The return value of `getc ()` is the order number of the entered character. A return value of `EOF` indicates an error of end-of-file condition.

The difference between the `getc ()` and `getchar ()` function is that the `getc ()` function can be implemented so that its arguments can be evaluated multiple times. The stream argument to `getc ()` should not be an expression with side effects.

```
int getchar (void);
```

Same description as with `getc ()`.

https://cplusplus.com/reference/cstdio/getchar/

99

Input / Output std functions
Unformatted Input

```
int getc (FILE * stream);
```

Example

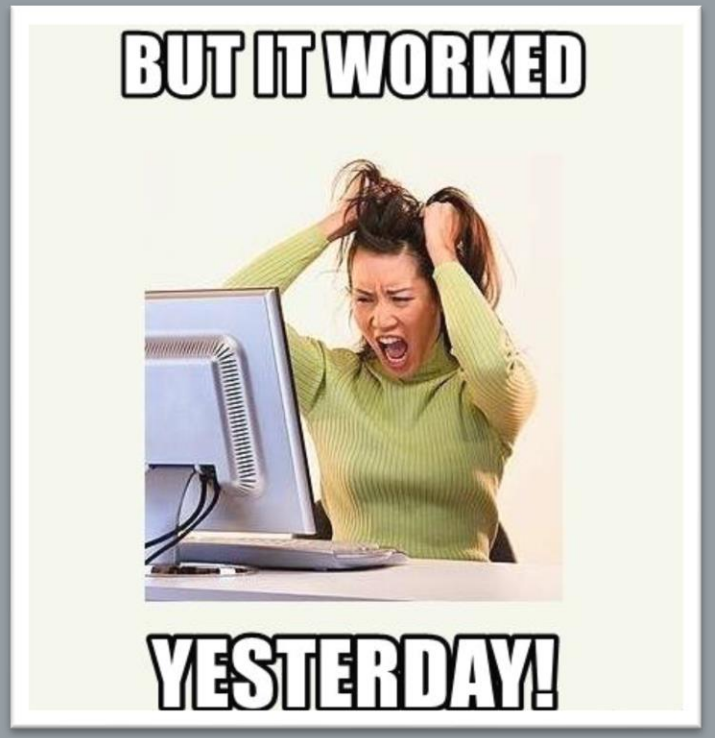
```
#include <stdio.h>
#define LINE 80
int main (void)
{
    char buffer [LINE+1];
    int i, ch;
    printf ("Please enter a string.\n");
    for (i=0; (i<LINE) && ((ch=getchar()) != EOF) && (ch!='\n'); i++)
        buffer [i] = ch;
    buffer [i] = '\0';  /*a string should always end with '\0' */
    printf ("The string is \"%s\"\n", buffer);
    getchar ();
    return (0);
}
```

Screen output

```
Please enter a string.
Hello world!
The string is "Hello world!"
```

https://cplusplus.com/reference/cstdio/getchar/

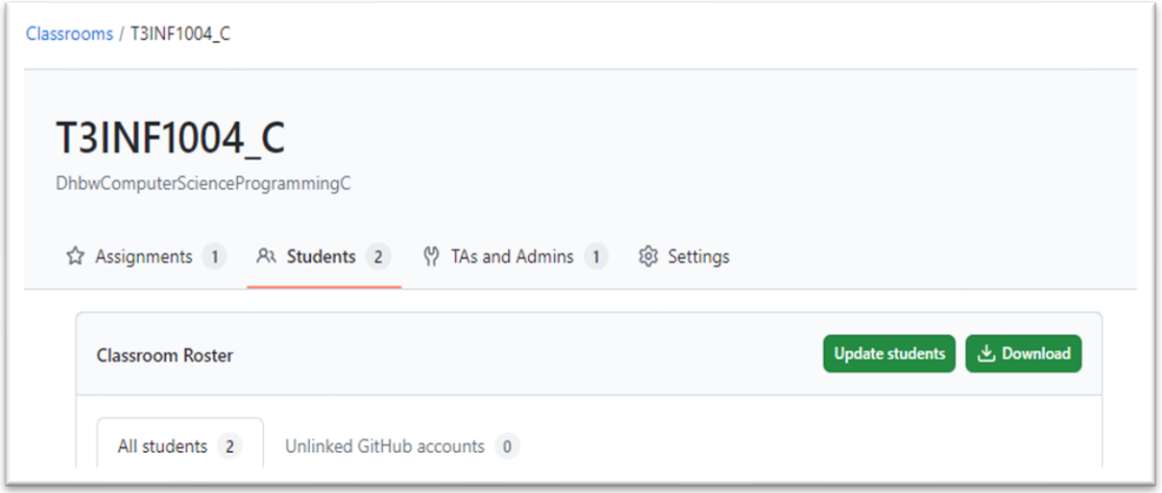
BACK TO GIT AGAIN



Live demo

Your Classroom for C coding Assignments

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



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