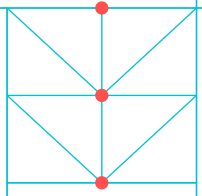
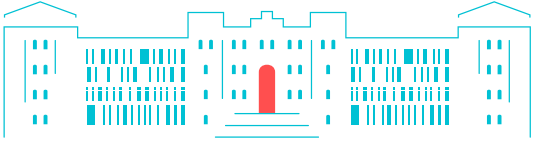
Prozedurale Programmierung für Informatiker (PPI)





TUHH
Hamburg
University of
Technology

Institute for Autonomous Cyber-Physical Systems

Prof. Dr.-Ing. Bernd-Christian Renner
with Dr.-Ing. Peter Oppermann, Wiebke Frenkel, and Johannes Göpfert

Introduction

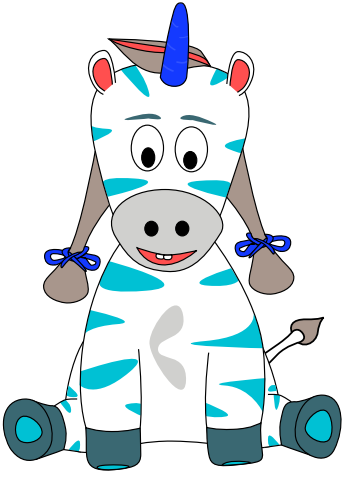
1

Learning to Walk and Talk

TUHH

Learning Goals

- Become familiar with general terms
- Understand the different paradigms of language processing
- Get acquainted with the C language
- Understanding the (basic) anatomy of a C(omputer) program



This is **Cebra**

Introduction

1

1

General Terms

Programming

Definition: (Computer) Programming

The process or activity of creating computer programs.

Programming involves

- understanding of a problem
- analysis and generation of algorithms to solve the problem
- profiling algorithms' accuracy and resource consumption
- implementation of algorithms (in a particular programming language)

Programming ≠ Implementation

Many people believe programming to be the same as implementation (coding) — this is wrong!

Algorithm

Definition: Algorithm

An ordered set of well-defined rules for the solution of a problem in a finite number of steps.

```
do
  moveForward
while not foundObstacle

if obstacle is fish
  followFish
else
  turnRandomly
```

Programming Languages

TUHH

Definition: Language

A set of characters, conventions, and rules that is used for a specific purpose.

Definition: Programming Language

An artificial language that is used to generate or to express algorithms. The language may be a high-level language or a low-level language (assembly language or machine language).

Machine Languages

Definition: Machine Language

A language that only uses the symbols 0 and 1 organized in bytes.

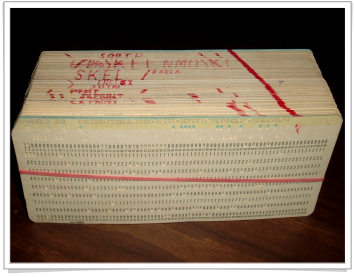
Example:

Addition of two 32-bit numbers on an x86-64

```
1100.0111 0100.0101 1111.1100 0001.0011 0000.0000 0000.0000 0000.0000
1100.0111 0100.0101 1111.1000 0101.0010 0000.0000 0000.0000 0000.0000
1000.1011 0101.0101 1111.1100
1000.1011 0100.0101 1111.1000
0000.0001 1101.0000
1000.1001 0100.0101 1111.0100
```

Problems:

Tedious, error prone, machine dependent



Source: By ArnoldReinhold - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=16041053

Assembly Languages

Definition: Assembly Language

A language that has mnemonics that directly correspond to machine language instructions.

Example:

Addition of two 32-bit numbers on an x86-64

```
1 mov    DWORD PTR [rbp-0x4],0x13
2 mov    DWORD PTR [rbp-0x8],0x52
3 mov    edx,DWORD PTR [rbp-0x4]
4 mov    eax,DWORD PTR [rbp-0x8]
5 add    eax,edx
6 mov    DWORD PTR [rbp-0xc],eax
```

Advantages:

Compact representation, more readable, high execution speed

Problems:

Still tedious, error prone, machine dependent, only for small programs

High-Level Languages

Definition: High-Level Language

A programming language that is primarily designed for, and syntactically oriented to, particular classes of problems and that is essentially independent of the structure of a specific computer or class of computers.

Example:

Addition of two 32-bit numbers (on an x86-64 or any other architecture)

```
1 int a = 19;   // declare a as an integer, initially 19
2 int b = 82;   // declare b as an integer, initially 82
3 int c;        // declare c as an integer, value unspecified
4 c = a + b;    // add a and b, store result in c
```

Paradigms of High-Level Programming Languages

- **procedural**: e.g. C, Pascal, Basic, ...
- **object-oriented**: e.g. Java, C++, Smalltalk, Eiffel, ...
- **functional**: e.g. LISP, ML, Haskell, ...
- **logical**: e.g. Prolog, Goedel, ...

Definition: Procedural Programming

A programming paradigm, classified as imperative programming, that involves implementing the behavior of a computer program as procedures (functions) that call each other. The resulting program is a series of steps that forms a hierarchy of calls to its constituent procedures.

Definition: Imperative Programming

A programming paradigm of software that uses statements that change a program's state.

Quiz 1.1

Which of the following statements regarding a high-level programming language is *wrong*?

A Execution is faster than that of a program written in machine language.	**B** It is more portable than a program written in machine language.
C It is better readible (more human understandable) than a program written in an assembly language.	**D** It is independent of a particular platform architecture (Intel, ARM, ...).

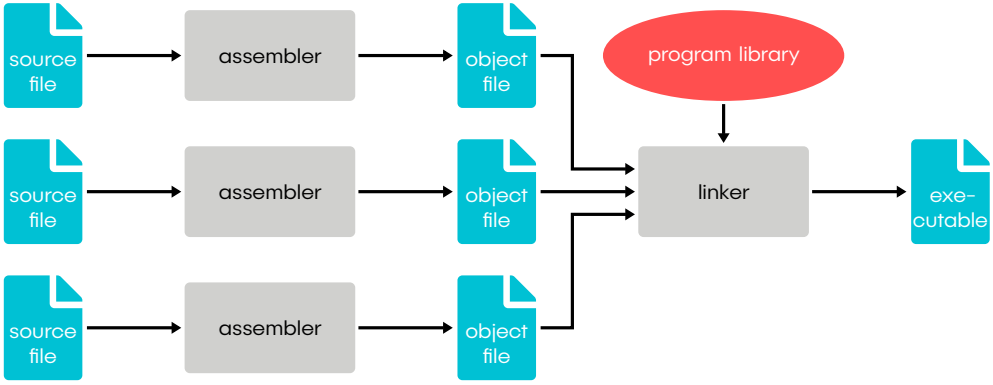
Introduction

1

2

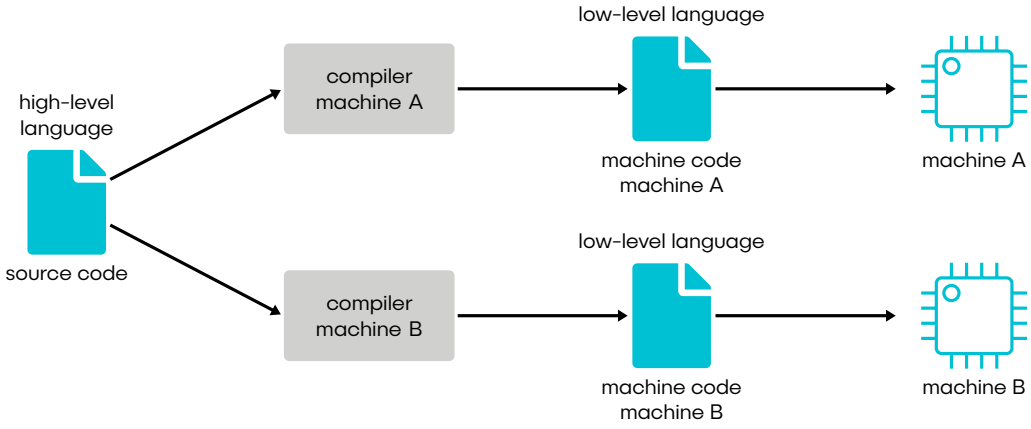
Different Paradigms of Language Processors

Assembler



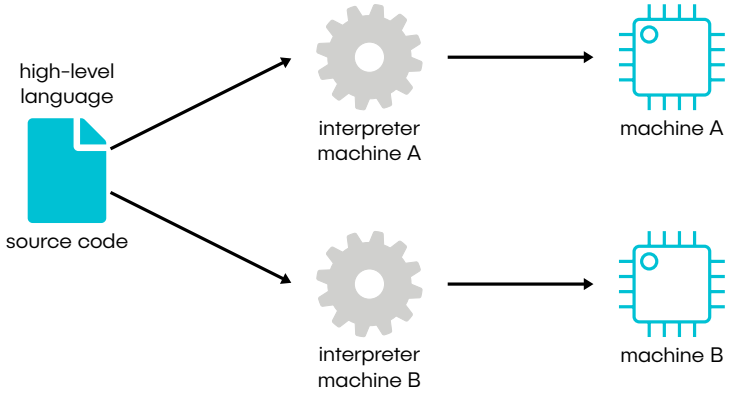
- Assembler: Tool that translates assembly into machine language code
  It reads a single assembly language source file and produces an object file containing machine instructions and information to combine several object files into a program
- Linker: Tool to combine object and library files into executable file

Compiler



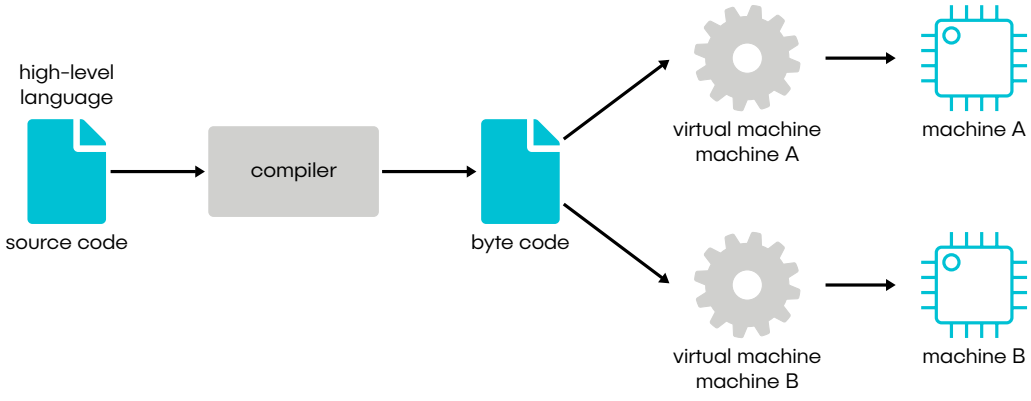
- Compiler translates code of a programming language in machine code
- It includes an assembler

Interpreter



- Interpreter implements or simulates a virtual machine
- Reads the statements of a program, analyses them, then executes them
- No compilation process, shorter development times

Hybrid (Compiled & Interpreted)



- Programming language is translated into machine-independent byte code
- The virtual machine interprets the byte code

Paradigms of Language Processors

	finding syntax errors	*execution speed*	*target machine requirements*
Compiler	compile time	high	none
Interpreter	run time	low	interpreter
Hybrid	compile time	medium	virtual machine

- There is no one-size-fits-all type of language processor
- Choice depends on specific problem domain

Examples

Compiler (C)

langproc/compiler/hello.c

- Source code compiled to binary image (via assembly)
- Binary image can be executed directly, if target platform matches
- Only changes in source code require recompilation

Interpreter (Python)

langproc/interpreter/hello.py

- Interpreter reads and executes source code in one pass
- Repeated on every program execution
- User needs interpreter

Hybrid (Java)

langproc/hybrid/Hello.java

- Compiled into Java byte code (platform-independent binary image)
- Byte code can be run on any machine by VM
- User needs VM

Introduction

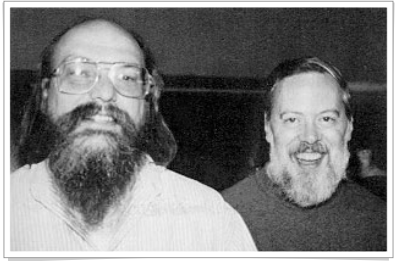
Programming in C: An Overview

1

3

The C Programming Language

- Hardware-oriented
  - directly *running on* the hardware
  - direct *access to* the hardware
- Helps to understand how a computer is working
- Higher abstraction than assembly
  more suited for humans
- More efficient than many other languages
  w.r.t. execution speed
- No complex methodology (as in OOP)
- Portable (ANSI C)



Dennis Ritchie (right), the inventor of the C programming language, with Ken Thompson

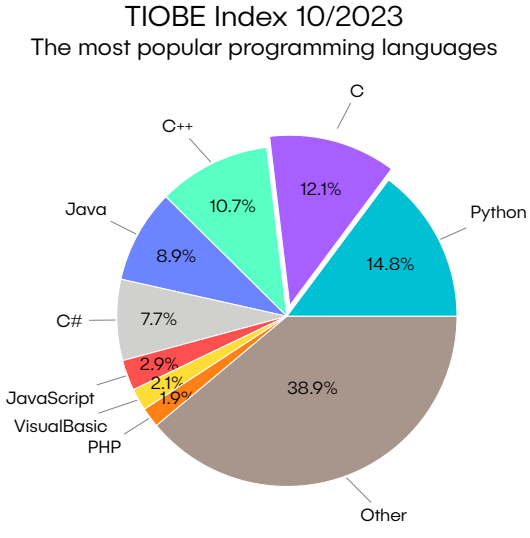
- Published by Dennis Ritchie in 1973
- Different standards
  C89/90, C95, C99, C11, C18
- Similar syntax to Java
- Imperative (no OOP)
- Modules, functions, structured data types

```c
#include <stdio.h>

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

- Potentially higher development time than Java & Co.
- More error-prone
  - less strict (than, e.g., Java)
  - no exception handling
  - pointers (no references)
  - no objects, little data encapsulation
  - no templates
  - ...

It's Old ... But *Not Outdated*

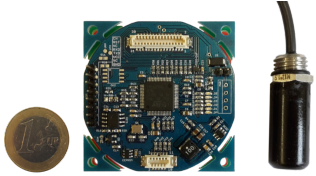
- Widely adopted in Embedded Systems
- Used for tools/components of higher languages
  Python interpreter, Java Runtime Environment, Sun's 1st Java Compiler
- Helpful in other courses
  Operating Systems, *Software for Embedded Systems*, *Autonomous Cyber-Physical Systems*, ...



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

Caught in the Wild

```
298     for (i = 0; i < NUM_SYMBOLS; i++) {
299       for (j = 0; j < FREQ_SYNC_NUM_MAX; j++) {
300         // multiply the sample with preamble waveforms of symbols
301         prod.sin = fp16_mul(sample, *pWaveform[i][j]++);
302         prod.cos = fp16_mul(sample, *pWaveform[i][j]++);
303
304         // update integral value using ringbuffer
305         // subtract the earliest value from the buffer and add the last one
306         integral[i][j].sin += prod.sin - ringbuffer[bufPos][i][j].sin;
307         integral[i][j].cos += prod.cos - ringbuffer[bufPos][i][j].cos;
308
309         // store new data in ringbuffer
310         ringbuffer[bufPos][i][j] = prod;
311       }
312     }
313
314     // noise calculation / n-tap filtering
315     sqSample         = fp32_mul(sample, sample);
316     noiseSum        += sqSample - noiseLP[bufPos];
317     noiseLP[bufPos]  = sqSample;
```



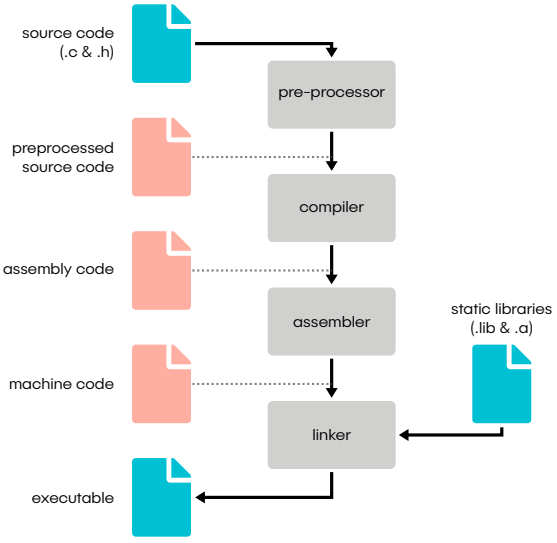
ahoi acoustic underwater modem

www.ahoi-modem.de

- **Source Code**
  Plain text files containing the program written in a programming language (e. g. C)
- **Executable (Binary)**
  File containing executable, platform-specific machine code
- **Toolchain**
  - Set of software development tools that are linked (or chained) together to produce an executable application for a processor
    For embedded systems, toolchain runs on a development system but produces binary code (executable) for a different architecture (e. g., Arduino)
  - Optionally, a toolchain may contain other tools such as a debugger
  - Various open source projects that comprise entire toolchain are available

Compilation Toolchain

1. **Pre-Processor** processes source code (inclusion of headers, macro expansions, conditional compilation, etc.)
2. **Compiler** translates human readable code into assembly language for a particular processor
3. **Assembler** translates assembly language into opcodes. Produces an object file
4. **Linker** organizes object files, necessary libraries, and other data and produces a relocatable file



Compilation

gcc (GNU Compiler Collection) takes care of all steps at once!

```
1 | $ gcc [-c|-S|-E] [-std=standard] [-Olvl] [-W...] [-Idir...] [-o outfile] infile...
```

Relevant options (for this course)

`-c|-S|-E`	Stop compilation at intermediate point
`-std=standard`	Defines the used C standard (we use C99)
`-Olvl`	optimization level (0,1,2,3,s)
`-W...`	enable specific warnings
`-Idir...`	search directory for header files
`-o outfile`	Name of the produced output file (default is *a.out* on Linux)

Typical use:

```
1 | $ gcc -std=c99 -Wall -o <outfile> <infile-1.c> [<infile-2.c> ...]
```

Makefiles

- The make utility is a handy automation tool for compiling programs efficiently
- Configuration through *Makefile*
  - based on targets
  - supports variables
- A template is provided for the lab
- Further reading:
  www.gnu.org/software/make

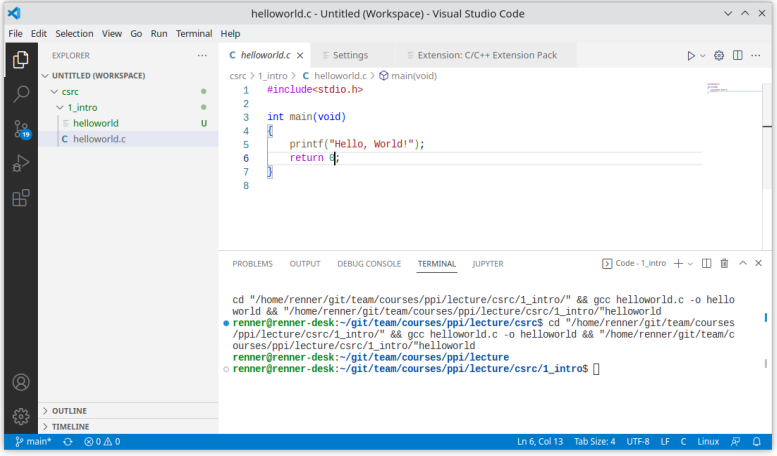
```
PROG = test
CSRCS = main.c io.c
OBJS = $(CSRCS:.c=.o)
IDIR = ./include

CC = gcc
CFLAGS = -std=c99 -Wall -Wextra -pedantic -O2 -I$(IDIR)
LDFLAGS = -lm

$(PROG): $(OBJS)
  $(CC) $(OBJS) $(LDFLAGS) -o $@

%.c.o: %.c
  $(CC) -c $(CFLAGS) $(LDFLAGS) -o $@ $<

.PHONY: clean
clean:
  rm *.o $(PROG)
```

Typical use:

```
$ make
```

IDE: Integrated Development Environment

- (Text) editor
- File browser
- Compilation & execution via button
- Integrated terminal (command line)
- ...



Great tool for effective and efficient programming but it hides details and reduces understanding.

Example: Compile and Run a Program

```c
#include <stdio.h>

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

Terminal plain

Compilation

```
$ cd path/to/sources
$ gcc -o helloworld helloworld.c
```

Execute program

```
$ ./helloworld
```

Terminal with Make

Compilation

```
$ cd path/to/sources
$ make
```

Run program

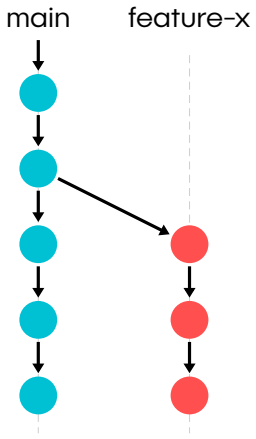
```
$ ./helloworld
```

IDE

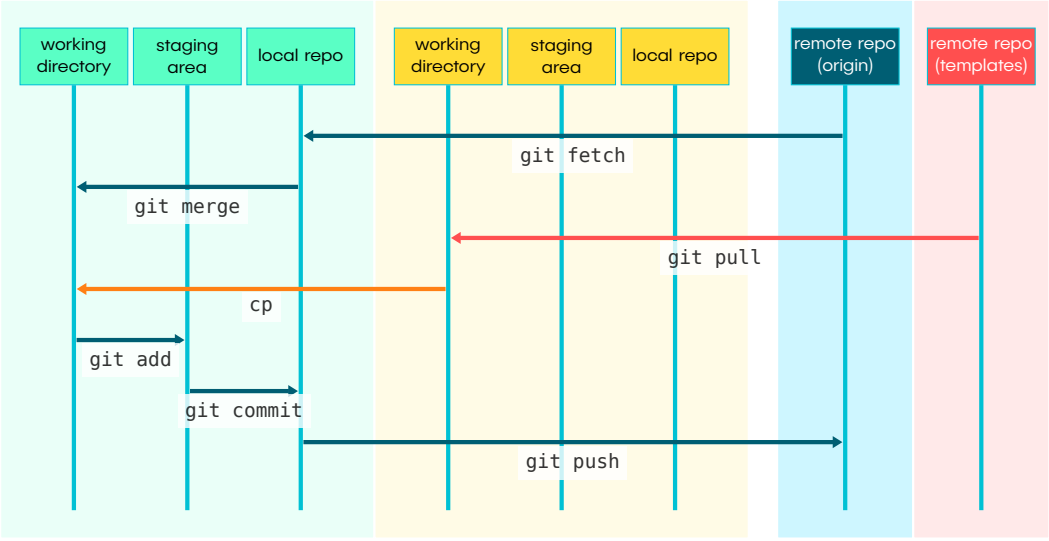
1. open file in editor
2. press button
3. find output

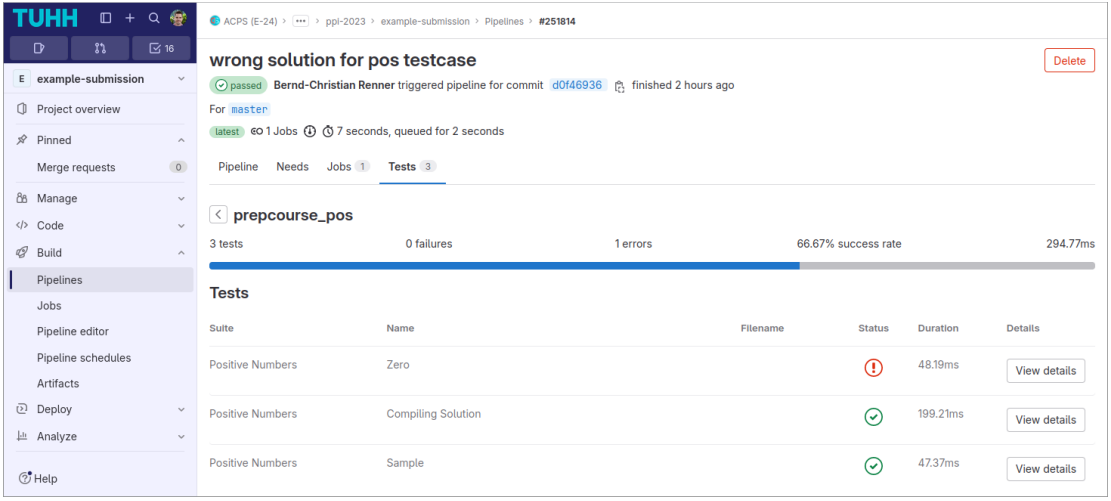
Version Control

- C files are plain text files
- Keeping track of changes is highly beneficial
  - providing a history of what you did
    e.g., when a feature was introduced
  - add the ability to review changes
    to understand what was done
  - find and fix bugs that were introduced earlier
    by inspecting the change history
- This (and other) functionality is offered by *Version Control*
  - TUHH hosts its own Gitlab at
    https://collaborating.tuhh.de
  - accessible for all members of TUHH
- We have a tutorial for you
  full documentation at https://git-scm.com/docs



PPI Git Workflow with Templates Submodule



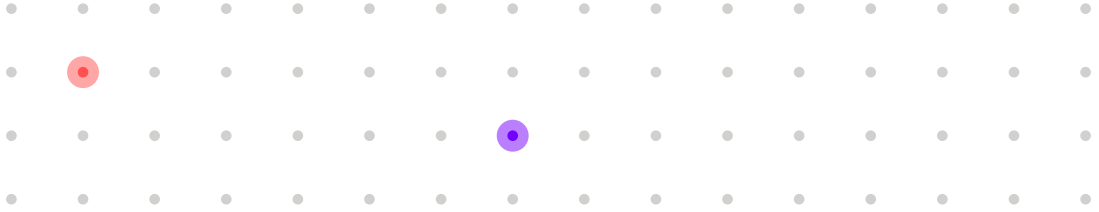

TUHH
16
E example-submission
Project overview
Pinned
Merge requests 0
Manage
Code
Build
Pipelines
Jobs
Pipeline editor
Pipeline schedules
Artifacts
Deploy
Analyze
Help
ACPS (E-24) › ••• › ppi-2023 › example-submission › Pipelines › #251814
wrong solution for pos testcase
Delete
passed Bernd-Christian Renner triggered pipeline for commit d0f46936 finished 2 hours ago
For master
latest 1 Jobs 7 seconds, queued for 2 seconds
Pipeline Needs Jobs 1 Tests 3
prepcourse_pos
3 tests 0 failures 1 errors 66.67% success rate 294.77ms
Tests
Suite Name Filename Status Duration Details
Positive Numbers Zero 48.19ms View details
Positive Numbers Compiling Solution 199.21ms View details
Positive Numbers Sample 47.37ms View details

Introduction

1

4

Hello World! Again.



Anatomy of a C Program

```c
/**
 * This would be the place for a file's summary comment
 */

#include <stdio.h>           // here, we include the header of a library

int main(void)               // this is, were our program starts
{
  printf("Hello again!");    // we use printf to write to the console
  return 0;                  // and we tell about successfull execution by returning 0
}
```

Summary

- A C file is a simple text file with file extension *.c*
- Pre-processor directives start with a **#**
- C programs start in the *function* `main`
  there must be exactly one function `main` in your program
- We can use comments, that are ignored by the compiler
- The **`return`** value is used to indicate how the program exited
  more on that in the course Operating Systems

Program

Definition: Program (simplified)

A **program** consists of a sequence of statements.

- During program execution the statements are executed one by one
- Order of execution: top down (in principle)
- Some statements can change this order
- Statements can change the values of variables
- Small number of different statements

Statements

Definition: Statement

A **statement** specifies an action to be performed.

- typ. concluded by a semicolon ;
  depends on the statement
- may contain other statements
- examples
  - expressions
  - labels
  - blocks
  - selections (conditionals)
  - iterations (repetition)
  - compounds (blocks)
  - jumps

Example

```
1 ...
2 do {
3   scanf("%d", &x);
4 } while (x < 0 || x > 10);
5 printf("You typed: %d\n", x);
6 ...
```

Wait and see

We will learn what all of this is and means throughout the course.

Syntax

Definition: Syntax

The rules that state how words and phrases must be used in a language.

Definition: Syntax Error

A **syntax error** is a violation of the programming language's grammar (syntax).

- For compiled languages, syntax errors are found and reported at compile-time. The compiler can only produce an executable, if all syntax errors in the source code have been corrected.
- An interpreter may behave like a compiler (with the difference that no executable is created), yet it may also execute a syntactically incorrect program until it shall execute a syntactically wrong statement or expression (when it aborts execution with an error message).

Semantics

Definition: Semantics

The meaning of words, phrases, systems.

Definition: Semantic Error

A **semantic error** is an invalid *program logic* that produces incorrect results during execution. The syntax of the source code is valid, but the algorithm being employed is not. A semantic error is also called a "logic error."

Style

Definition: (Code) Style

- The quality of being elegant and made to a high standard.
- A set of rules or guidelines used when writing the source code for a computer program to improve readability and to avoid the introduction of errors.

Definition: Style Error

A **style error** is an error that only affects style (visual representation, conventions, ...).

(Code) Style

Make your programs readable, where appropriate, by

- using a single statement per line only
  unless they are very short or there is another good reason
- indenting code blocks
  will be handled later
- inserting blank lines and comments
- adding white spaces
- abiding by a style guide

Remarks

- Tabs are evil (use blanks)!
- The compiler ignores unnecessary white spaces.
- Style your programs while writing them, *not* afterwards!

Comments

Definition: Comment

A *comment* is a programmer-readable explanation or annotation in the source code of a computer program.

Use cases

- Explain your code, only if it is not self-explanatory
- Sketch a file's and/or function's semantic content
- Add auto-documentation tags and fields
- Divide a C file or function into sections
  will be handy later

Types of comments

- Multi-line comments are delimited by `/*` and `*/`
- Single-line comments are started by `//` until the end of the line

You can mix comment types within a C file, but you should be consistent!

Never ever ...

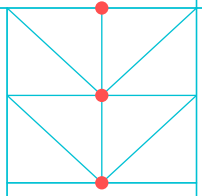
Never ever use comments to repeat the obvious or decipher your code!

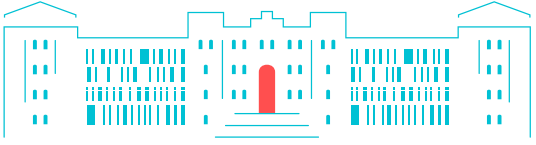
```
double
calc(double r, unsigned h, double ovt, unsigned ovh)
{
  /*
   * Function to calculate pay of an employee
   * Parameters:
   *     r: the hourly pay rate (as decimal)
   *     h: the work hours (as integer >= 0)
   *   ovt: the overtime pay rate (as decimal)
   *   ovh: the overtime work hours (as integer >= 0)
   *
   * Return:
   *   the total amount of pay
   */

  // calculate pay for regular working hours
  double t = r * h;

  // apply overtime rate
  t += ovt * ovh;

  // return total pay
  return t;
}
```

```
double
calc_pay(double hourly_rate,
         unsigned hours,
         double overtime_hourly_rate,
         unsigned overtime_hours)
{
  double pay;
  pay = hours * hourly_rate;
  pay += overtime_hours * overtime_hourly_rate;
  return pay;
}
```

Source: adapted from https://towardsdatascience.com/why-good-codes-dont-need-comments-92f58de19ad2

Prozedurale Programmierung für Informatiker (PPI)





TUHH
Hamburg
University of
Technology

Institute for Autonomous Cyber-Physical Systems

Prof. Dr.-Ing. Bernd-Christian Renner
with Dr.-Ing. Peter Oppermann, Wiebke Frenkel, and Johannes Göpfert