Prozedurale Programmierung - TU Freiberg

https://github.com/TUBAF-IfI-LiaScript/VL_ProzeduralePrognammierung/

andre-dietrich JayTee42 SebastianZug galinarudollf DkPepper Lalelele Lorcc Anjuschenka

Inhaltsverzeichnis

1 Einführung **5**
1.1 Umfrage . . . 5
1.2 Wie arbeitet ein Rechner eigentlich? . . . 5
1.2.1 Programmierung . . . 8
1.2.2 Einordnung von C und C++ . . . 9
1.3 Erstes C++ Programm . . . 9
1.3.1 “Hello World” . . . 9
1.3.2 Ein Wort zu den Formalien . . . 10
1.3.3 Gute Kommentare . . . 10
1.3.4 Schlechte Kommentare . . . 11
1.3.5 Was tun, wenn es schief geht? . . . 12
1.3.6 Compilerfehlermeldungen . . . 12
1.3.7 Und wenn das Kompilieren gut geht? . . . 12
1.4 Warum dann C++? . . . 13
1.5 Beispiele der Woche . . . 13

Kapitel 1

Einführung

Parameter	Kursinformationen
Veranstaltung:	`Vorlesung Prozedurale Programmierung / Einführung in die Informatik`
Semester	`Wintersemester 2022/23`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	`Vorstellung des Arbeitsprozesses`
Link auf Repository:	https://github.com/TUBAF-IfI-LiaScript/VL_ProzeduraleProgrammierung/blob/master/00_Einfuehrung.md
Autoren	@author

Fragen an die heutige Veranstaltung ...

- Welche Aufgabe erfüllt eine Programmiersprache?
- Erklären Sie die Begriffe Compiler, Editor, Programm, Hochsprache!
- Was passiert beim Kompilieren eines Programmes?
- Warum sind Kommentare von zentraler Bedeutung?
- Worin unterscheiden sich ein konventionelles C++ Programm und eine Anwendung, die mit dem Arduino-Framework geschrieben wurde?

1.1 Umfrage

Hat Sie die letztwöchige Vorstellung der Ziele der Lehrveranstaltung überzeugt?

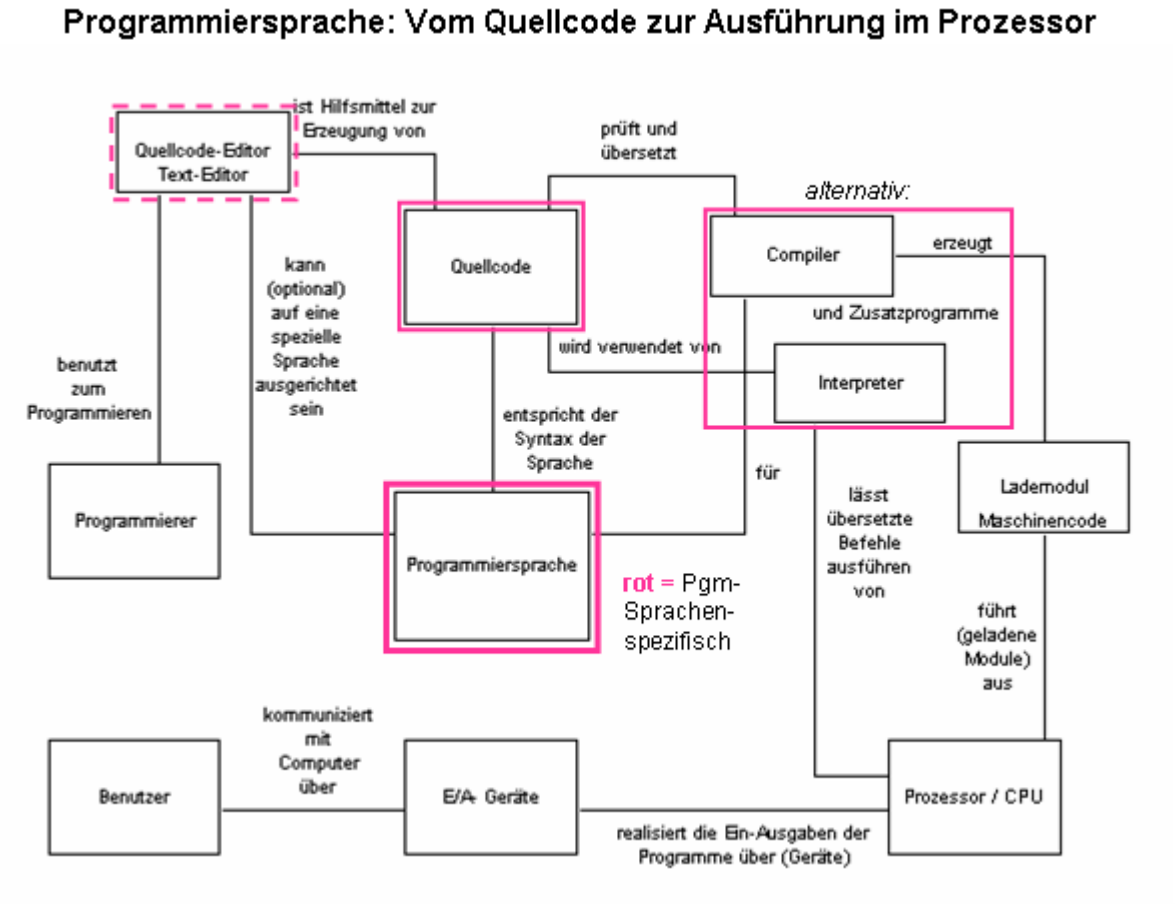
- [(ja)] Ja, ich gehe davon aus, viel nützliches zu erfahren.
- [(schau'n wir mal)] Ich bin noch nicht sicher. Fragen Sie in einigen Wochen noch mal.
- [(nein)] Nein, ich bin nur hier, weil ich muss.

1.2 Wie arbeitet ein Rechner eigentlich?

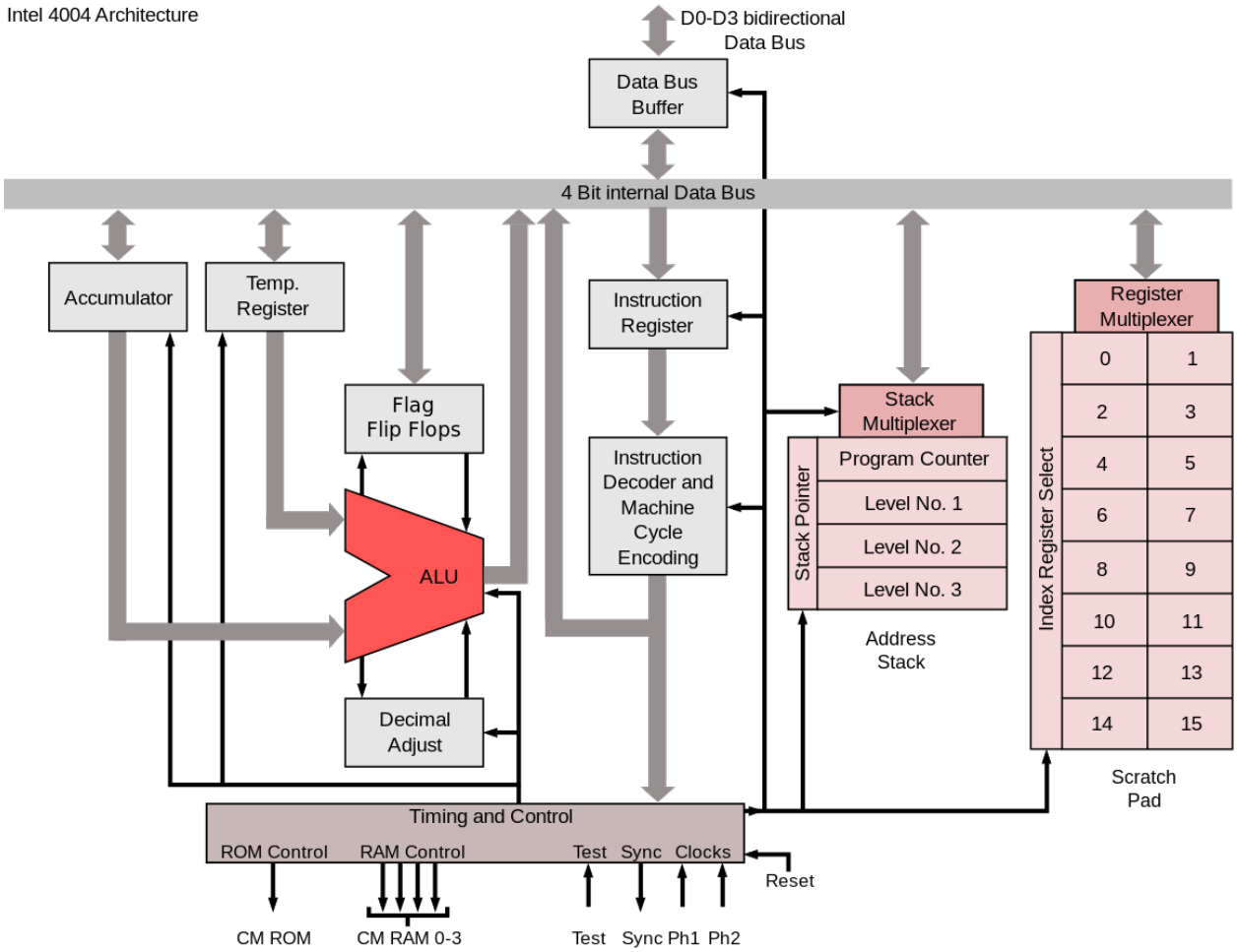
Programme sind Anweisungslisten, die vom Menschen erdacht, auf einem Rechner zur Ausführung kommen. Eine zentrale Hürde ist dabei die Kluft zwischen menschlicher Vorstellungskraft und Logik, die **implizite Annahmen und Erfahrungen** einschließt und der **"stupiden" Abarbeitung von Befehlsfolgen** in einem Rechner.

Programmiersprachen bemühen sich diese Lücke zu schließen und werden dabei von einer Vielzahl von Tools begleitet, diesen **Transformationsprozess** unterstützen sollen.

Um das Verständnis für diesen Vorgang zu entwickeln werden zunächst die Vorgänge in einem Rechner bei der Abarbeitung von Programmen beleuchtet, um dann die Realisierung eines Programmes mit C++ zu adressieren.



Beispiel: Intel 4004-Architektur (1971)



Jeder Rechner hat einen spezifischen Satz von Befehlen, die durch “0” und “1” ausgedrückt werden, die er überhaupt abarbeiten kann.

Speicherauszug den Intel 4004:

Adresse	Speicherinhalt	OpCode	Mnemonik
0010	1101 0101	1101 DDDD	LD $5
0012	1111 0010	1111 0010	IAC

Unterstützung für die Interpretation aus dem Nutzerhandbuch, dass das *Instruction Set* beschreibt:

4004 Instruction Set

BASIC INSTRUCTIONS (* = 2 Word Instructions)

Hex Code	MNEMONIC	OPR $D_3 D_2 D_1 D_0$	OPA $D_3 D_2 D_1 D_0$	DESCRIPTION OF OPERATION
00	NOP	0 0 0 0	0 0 0 0	No operation.
1 -	*JCN	0 0 0 1	$C_1 C_2 C_3 C_4$	Jump to ROM address $A_2 A_2 A_2 A_2, A_1 A_1 A_1 A_1$ (within the same ROM that contains this JCN instruction) if condition $C_1 C_2 C_3 C_4$ is true, otherwise go to the next instruction in sequence.
- -		$A_2 A_2 A_2 A_2$	$A_1 A_1 A_1 A_1$	
2 -	*FIM	0 0 1 0	R R R 0	Fetch immediate (direct) from ROM Data $D_2 D_2 D_2 D_2 D_1 D_1 D_1 D_1$ to index register pair location RRR.
- -		$D_2 D_2 D_2 D_2$	$D_1 D_1 D_1 D_1$	

▪ ▪ ▪

8 -	ADD	1 0 0 0	R R R R	Add contents of register RRRR to accumulator with carry.
9 -	SUB	1 0 0 1	R R R R	Subtract contents of register RRRR to accumulator with borrow.
A -	LD	1 0 1 0	R R R R	Load contents of register RRRR to accumulator.
B -	XCH	1 0 1 1	R R R R	Exchange contents of index register RRRR and accumulator.
C -	BBL	1 1 0 0	D D D D	Branch back (down 1 level in stack) and load data DDDD to accumulator.
D -	LDM	1 1 0 1	D D D D	Load data DDDD to accumulator.
F0	CLB	1 1 1 1	0 0 0 0	Clear both. (Accumulator and carry)
F1	CLC	1 1 1 1	0 0 0 1	Clear carry.
F2	IAC	1 1 1 1	0 0 1 0	Increment accumulator.

Quelle: Intel 4004 Assembler

1.2.1 Programmierung

Möchte man so Programme schreiben?

Vorteil:

- ggf. sehr effizienter Code (Größe, Ausführungsdauer), der gut auf die Hardware abgestimmt ist

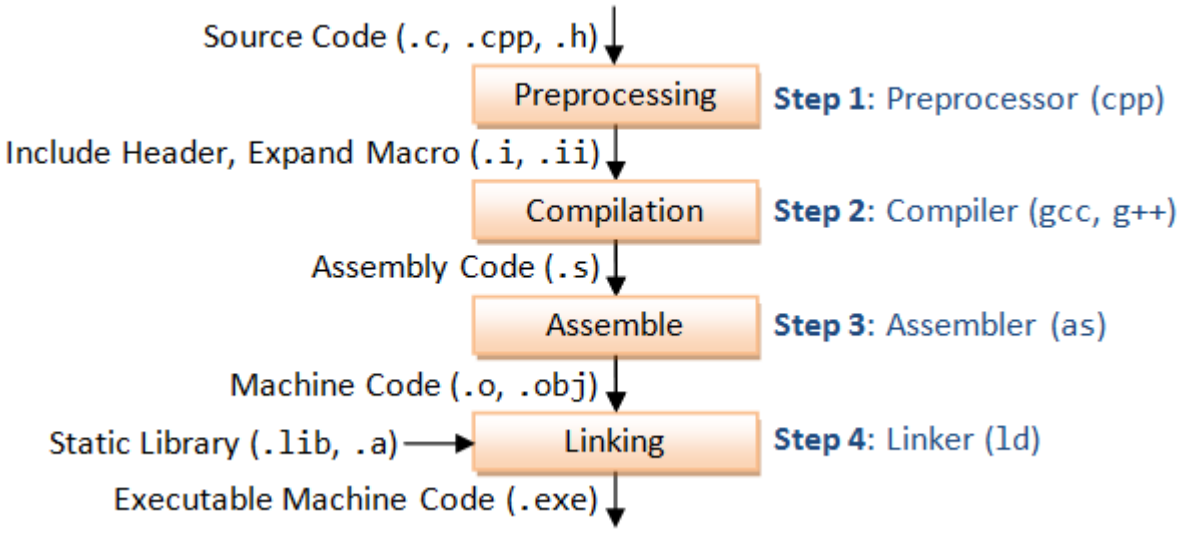
Nachteile:

- systemspezifische Realisierung
- geringer Abstraktionsgrad, bereits einfache Konstrukte benötigen viele Codezeilen
- weitgehende semantische Analysen möglich

Eine höhere Programmiersprache ist eine Programmiersprache zur Abfassung eines Computerprogramms, die in **Abstraktion und Komplexität** von der Ebene der Maschinensprachen deutlich entfernt ist. Die Befehle müssen durch **Interpreter oder Compiler** in Maschinensprache übersetzt werden.

Ein **Compiler** (auch Kompiler; von englisch für zusammentragen bzw. lateinisch compilare ‚aufhäufen') ist ein Computerprogramm, das Quellcodes einer bestimmten Programmiersprache in eine Form übersetzt, die von einem Computer (direkter) ausgeführt werden kann.

Stufen des Compile-Vorganges:



1.2.2 Einordnung von C und C++

- Adressiert Hochsprachenaspekte und Hardwarenähe -> Hohe Geschwindigkeit bei geringer Programmgröße
- Imperative Programmiersprache

  imperative (befehlsorientierte) Programmiersprachen: Ein Programm besteht aus einer Folge von Befehlen an den Computer. Das Programm beschreibt den Lösungsweg für ein Problem (C, Python, Java, LabView, Matlab, …).

  deklarative Programiersprachen: Ein Programm beschreibt die allgemeinen Eigenschaften von Objekten und ihre Beziehungen untereinander. Das Programm beschreibt zunächst nur das Wissen zur Lösung des Problems (Prolog, Haskal, SQL, …).
- Wenige Schlüsselwörter als Sprachumfang

  Schlüsselwort Reserviertes Wort, das der Compiler verwendet, um ein Programm zu parsen (z.B. if, def oder while). Schlüsselwörter dürfen nicht als Name für eine Variable gewählt werden
- Große Mächtigkeit

  Je “höher” und komfortabler die Sprache, desto mehr ist der Programmierer daran gebunden, die in ihr vorgesehenen Wege zu beschreiten.

1.3 Erstes C++ Programm

1.3.1 “Hello World”

```
1 // That's my first C program
2 // Karl Klammer, Oct. 2022
3
4 #include <iostream>
5
6 int main() {
7     std::cout << "Hello World!";
8     return 0;
9 }
```

Zeile	Bedeutung
1 - 2	Kommentar (wird vom Präprozessor entfernt)
4	Voraussetzung für das Einbinden von Befehlen der Standardbibliothek hier `std:cout()`
6	Einsprungstelle für den Beginn des Programmes
6 - 9	Ausführungsblock der `main`-Funktion

Zeile	Bedeutung
7	Anwendung eines Operators << hier zur Ausgabe auf dem Bildschirm
8	Definition eines Rückgabewertes für das Betriebssystem

Halt! Unsere C++ Arduino Programme sahen doch ganz anders aus?

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

@AVR8js.sketch

Noch mal Halt! Das klappt ja offenbar alles im Browserfenster, aber wenn ich ein Programm auf meinem Rechner kompilieren möchte, was ist dann zu tuen?

1.3.2 Ein Wort zu den Formalien

```
// Karl Klammer
// Print Hello World drei mal

#include <iostream>

int main() {
  int zahl;
  for (zahl=0; zahl<3;  zahl++){
        std::cout << "Hello World! ";
  }
    return 0;
}
```

```
#include <iostream>
int main() {int zahl; for (zahl=0; zahl<3;  zahl++){ std::cout << "Hello World! ";} return 0;}
```

- Das *systematische Einrücken* verbessert die Lesbarkeit und senkt damit die Fehleranfälligkeit Ihres Codes!
- Wählen sie *selbsterklärende Variablen- und Funktionsnamen*!
- Nutzen Sie ein *Versionsmanagmentsystem*, wenn Sie ihren Code entwickeln!
- Kommentieren Sie Ihr Vorgehen trotz “Good code is self-documenting”

1.3.3 Gute Kommentare

1. Kommentare als Pseudocode

```
/* loop backwards through all elements returned by the server
(they should be processed chronologically)*/
for (i = (numElementsReturned - 1); i >= 0; i--){
    /* process each element's data */
    updatePattern(i, returnedElements[i]);
}
```

2. Kommentare zur Datei

```
// This is the mars rover control application
//
// Karl Klammer, Oct. 2018
// Version 109.1.12
```

```
5
6 int main(){...}
```

3. Beschreibung eines Algorithmus

```
1 /* Function:  approx_pi
2  * --------------------
3  * computes an approximation of pi using:
4  *    pi/6 = 1/2 + (1/2 x 3/4) 1/5 (1/2)^3  + (1/2 x 3/4 x 5/6) 1/7 (1/2)^5 +
5  *
6  *  n: number of terms in the series to sum
7  *
8  *  returns: the approximate value of pi obtained by suming the first n terms
9  *           in the above series
10 *           returns zero on error (if n is non-positive)
11 */
12
13 double approx_pi(int n);
```

In realen Projekten werden Sie für diese Aufgaben Dokumentationstools verwenden, die die Generierung von Webseite, Handbüchern auf der Basis eigener Schlüsselworte in den Kommentaren unterstützen -> doxygen.

4. Debugging

```
1 int main(){
2   ...
3   preProcessedData = filter1(rawData);
4   // printf('Filter1 finished ... \n');
5   // printf('Output %d \n', preProcessedData);
6   result=complexCalculation(preProcessedData);
7   ...
8 }
```

1.3.4 Schlechte Kommentare

1. Überkommentierung von Code

```
1 x = x + 1;  /* increment the value of x */
2 std::cout << "Hello World! "; // displays Hello world
```

"... over-commenting your code can be as bad as under-commenting it"

Quelle: C Code Style Guidelines

2. "Merkwürdige Kommentare"

```
1 //When I wrote this, only God and I understood what I was doing
2 //Now, God only knows
3
4 // sometimes I believe compiler ignores all my comments
5
6 // Magic. Do not touch.
7 Hello World !Hello World !Hello World !Hello World !Hello World !Hello World !Hello World
      !Hello World !Hello Wor
8 // I am not responsible of this code.
9
10 try {
11
12 } catch(e) {
13
14 } finally { // should never happen }
```

Sammlung von Kommentaren

1.3.5 Was tun, wenn es schief geht?

```
// Karl Klammer
// Print Hello World drei mal

#include <iostream>

int main() {
  for (zahl=0; zahl<3;  zahl++){
        std::cout << "Hello World! "
  }
    return 0;
```

Methodisches Vorgehen:

- ** RUHE BEWAHREN **
- Lesen der Fehlermeldung
- Verstehen der Fehlermeldung / Aufstellen von Hypothesen
- Systematische Evaluation der Thesen
- Seien Sie im Austausch mit anderen (Kommilitonen, Forenbesucher, usw.) konkret

1.3.6 Compilerfehlermeldungen

Beispiel 1

```
#include <iostream>

int mani() {
    std::cout << "Hello World!";
    return 0;
}
```

Beispiel 2

```
#include <iostream>

int main()
    std::cout << "Hello World!";
    return 0;
}
```

> Manchmal muss man sehr genau hinschauen, um zu verstehen, warum ein Programm nicht funktioniert. Versuchen Sie es!

```
#include <iostream>

imt main() {
    std::cout << "Hello World!'';
    std::cout << "Wo liegt der Fehler?";
    return 0;
}
```

1.3.7 Und wenn das Kompilieren gut geht?

... dann bedeutet es noch immer nicht, dass Ihr Programm wie erwartet funktioniert.

```
#include <iostream>

int main() {
  char zahl;
  for (zahl=250; zahl<256; zahl++){
        std::cout << "Hello World!";
  }
    return 0;
}
```

Hinweis: Die Datentypen werden wir in der nächsten Woche besprechen.

1.4 Warum dann C++?

Zwei Varianten der Umsetzung … C++ vs. Python

```cpp
#include <iostream>

int main() {
  char zahl;
  for (int zahl=0; zahl<3; zahl++){
        std::cout << "Hello World! " << zahl << "\n";
  }
    return 0;
}
```

```python
for i in range(3):
  print("Hallo World ", i)
```

1.5 Beispiele der Woche

Gültiger C++ Code

```cpp
#include <iostream>

imt main() {
  int i = 5;
  int j = 4;
  i = i + j + 2;
    std::cout << "Hello World! ";
  std::cout << i << "!";
    return 0;
}
```

Umfrage: Welche Ausgabe erwarten Sie für folgendes Code-Schnippselchen?

- [(Hello World5)] `Hello World7`
- [(Hello World11)] `Hello World11`
- [(Hello World 11!)] `Hello World 11!`
- [(Hello World 11 !)] `Hello World 11 !`
- [(Hello World 5 !)] `Hello World 5 !`

Algorithmisches Denken

Aufgabe: Ändern Sie den Code so ab, dass das die LED zwei mal mit 1 Hz blinkt und dann ausgeschaltet bleibt.

```cpp
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

@AVR8js.sketch