Inhaltsverzeichnis

Kapitel 1

Einführung

Parameter	Kursinformationen
Veranstaltung:	`Vorlesung Prozedurale Programmierung`
Semester	`Wintersemester 2021/22`
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!

1.1 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.

instruction-set

Beispiel: Intel 4004-Architektur (1971)

intel

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:

instruction-set

Quelle: Intel 4004 Assembler

1.1.1 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

Beispiel

```
1 Assembler Code    |  Fortran
2                   |
3 .START ST         |  A:=2;
4 ST: MOV R1, #2    |  FOR I:=1 TO 20 LOOP
5     MOV R2, #1    |      A:=A*I;
```

```
6  M1: CMP R2, #20   |  END LOOP;
7      BGT M2        |  PRINT(A);
8      MUL R1, R2    |
9      INI R2        |
10     JMP M1        |
11 M2: JSR PRINT     |
12 .END              |
```

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:

instruction-set

1.1.2 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.

1.2 Erstes C Programm

instruction-set

Quelle: Brian_Kernighan, Programming in C: A Tutorial 1974

1.2.1 "Hello World"

```
1 // That's my first C program
2 // Karl Klammer, Oct. 2018
3
4 #include<stdio.h>
5
6 int main(void) {     // alternativ "int main()"
7     printf("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 `printf()`
6	Einsprungstelle für den Beginn des Programmes
6 - 9	Ausführungsblock der `main`-Funktion
7	Anwendung einer Funktion … `name(Parameterliste)` … hier zur Ausgabe auf dem Bildschirm
8	Definition eines Rückgabewertes für das Betriebssystem

1.2.2 Ein Wort zu den Formalien

```
1 // Karl Klammer
2 // Print Hello World drei mal
3
4 #include <stdio.h>
5
6 int main() {
7   int zahl;
8   for (zahl=0; zahl<3;  zahl++){
9         printf("Hello World! ");
10  }
```

```
11     return 0;
12 }
```

```
1 #include <stdio.h>
2 int main() {int zahl; for (zahl=0; zahl<3;  zahl++){ printf("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"

1.2.3 Gute Kommentare

1. Kommentare als Pseudocode

```
1 /* loop backwards through all elements returned by the server
2 (they should be processed chronologically)*/
3 for (i = (numElementsReturned - 1); i >= 0; i--){
4     /* process each element's data */
5     updatePattern(i, returnedElements[i]);
6 }
```

2. Kommentare zur Datei

```
1 // This is the mars rover control application
2 //
3 // Karl Klammer, Oct. 2018
4 // 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)
```

```
 */

 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

```
int main(){
  ...
  preProcessedData = filter1(rawData);
  // printf('Filter1 finished ... \n');
  // printf('Output %d \n', preProcessedData);
  result=complexCalculation(preProcessedData);
  ...
}
```

1.2.4 Schlechte Kommentare

1. Überkommentierung von Code

```
x = x + 1;  /* increment the value of x */
printf("Hello World\n"); // displays Hello world
```

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

Quelle: C Code Style Guidelines

2. "Merkwürdige Kommentare"

```
//When I wrote this, only God and I understood what I was doing
//Now, God only knows

// sometimes I believe compiler ignores all my comments

// Magic. Do not touch.

// I am not responsible of this code.

try {

} catch(e) {

} finally { // should never happen }
```

Sammlung von Kommentaren

1.2.5 Was tun, wenn es schief geht?

```
#include<stdio.h>

int main() {
  for (zahl=0; zahl<3;  zahl++){
        printf("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

1.2.6 Compilermessages

Beispiel 1

```
#include <stdio.h>

int mani() {
    printf("Hello World");
    return 0;
}
```

Beispiel 2

```
#include <stdio.h>

int main()
    printf("Hello World");
    return 0;
}
```

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

```
#include <stdio.h>

imt main() {
    printf("Hello World\n'');
    printf("Wo liegt der Fehler?")
```

```
6     return 0;
7 }
```

1.2.7 Und wenn das Kompilieren gut geht?

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

```
1 #include <stdio.h>
2
3 int main() {
4   char zahl;
5   for (zahl=250; zahl<256; zahl++){
6         printf("Hello World !");
7   }
8     return 0;
9 }
```

1.3 Warum dann C?

Zwei Varianten der Umsetzung … C vs. Python

```
1 #include <stdio.h>
2
3 int main() {
4   char zahl;
5   for (int zahl=0; zahl<3; zahl++){
6         printf("%d Hello World\n", zahl);
7   }
8     return 0;
9 }
```

```
1 import sys
2
3 for i in range(3):
4     print(i, "Hello World")
5
6 #sys.exit()
```

@Pyodide.eval

1.4 Ausblick

```
1 #include<stdio.h>
2
```

```
3 int main() {
4   printf("... \t bis \n\t\t zum \n\t\t\t");
5   printf("naechsten \n\t\t\t\t\t mal! \n");
6     return 0;
7 }
```