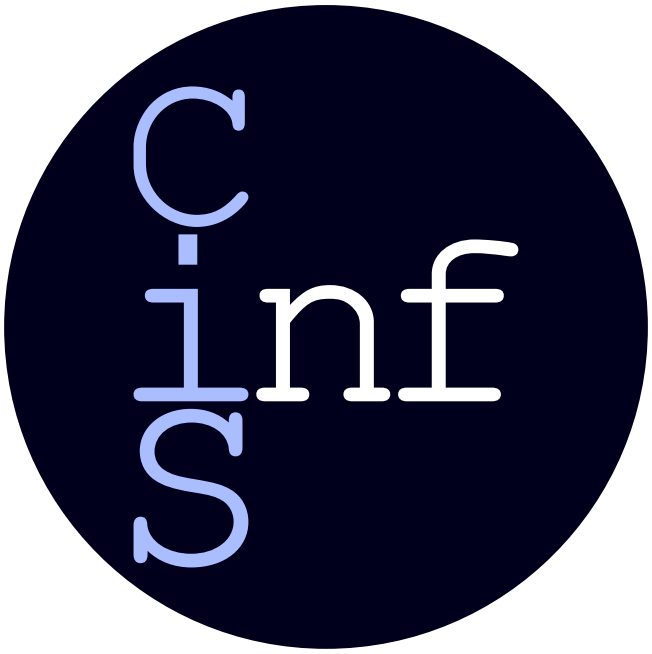

Softwareentwicklung 1

Alan Kniep, Livi Franke

6. Februar 2021

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Inhaltsverzeichnis

1 Einführung **1**

2 Binärsystem **2**

3 Imperative Programmierung **3**
3.1 Einleitung 3
3.1.1 Von Neumann-Rechner 3
3.1.2 Typen von Programmiersprachen 3
3.2 Literale, Operatoren und Ausdrücke 5
3.2.1 Operatoren 5
3.2.2 Variablen 6
3.3 Prozeduren 8

4 Objektorientierte Programmierung **9**
4.1 Kernbegriffe 9

1. Einführung

Software:
Software ist eine Familie von **ausführbaren Computerprogrammen**, damit verbundenen **Ressouren** (z.B. Bilder oder Schriftarten) sowie **Anwenderdatenbeständen**, die zur **Lösung einer festgelegten Aufgabe** eine bestimmte **Menge von Funktionen** anbietet und auf **vorgegebenen Hardwarestrukturen zur Ausführung** bringt. Software ist of sehr umfangreich und besteht neben **Code** auch aus **Nutzerdokumentation**, **Anforderungs-** und **Architekturbeschreibung**.

Herausvorderung/Größenordnung:

- Zeitaufwand
- Problemstellung
- Teamgröße
- Komplexität
- Lange Betriebszeiten
- ...

Algorithmus

- besteht aus einer endlich Menge an Einzelschritten
- dient zur Lösung eines bestimmten Problems
- Handlungsvorschrift für die Lösung
- können in Programmiersprachen beschrieben werden

Kontrollstrukturen

- Sequenz
- Aufruf zusammengesetzter Befehle
- Zählschleife
- Fallunterscheidung
- Bedingte Schleife

2. Binärsystem

Zahlen im Binärsystem werden zur Basis 2 geschrieben, d.h. dass wir im Kontrast zum Dezimalsystem folgende Ziffernbenennung an bestimmten Positionen haben:

Position	Name_{10}	Zifferwert_{10}	Name_2	Zifferwert_2
$\underline{1}$	Einer	$(0 \sim 9) * 10^0$	Einer	$(0 \sim 1) * 2^0$
$\underline{1}0$	Zehner	$(0 \sim 9) * 10^1$	Zweier	$(0 \sim 1) * 2^1$
$\underline{1}00$	Hunderter	$(0 \sim 9) * 10^2$	Vierer	$(0 \sim 1) * 2^2$
$\underline{1}\ 000$	Tausender	$(0 \sim 9) * 10^3$	Achter	$(0 \sim 1) * 2^3$
$\underline{1}0\ 000$	Zehntausender	$(0 \sim 9) * 10^4$	Sechzehner	$(0 \sim 1) * 2^4$
...	...	...	...	...

Im Binärsystem kann man die gleichen Rechenoperationen machen wie auch im Dezimalsystem, nur unterscheidet sich teilweise die Herangehensweise.

3. Imperative Programmierung

3.1 Einleitung

3.1.1 Von Neumann-Rechner

Die Struktur eines von Neumann-Rechners ist unabhängig von dem bearbeiteten Problem und besteht aus 5 Komponenten.

- **ALU** (Arithmetic Logic Unit) oder auch **Rechenwerk** ist für die Ausführung von logischen Verknüpfungen und Rechenoperationen zuständig.
- **Control Unit** oder auch **Steuerwerk**, interpretiert die Anweisungen des Programms und hält sich dabei an den von-Neumann-Zyklus (Fetch, Decode, Fetch Operands, Execute, Write Back). Das Stauerwerk steuert alle anderen Funktionseinheiten des Prozessors.
- **Bus-System** über das Bussystem kommunizieren die einzelnen Komponenten, es werden bspw. Befehle und Daten aus dem Speicher in die CPU übertragen und die Ergebnisse zurück in den Speicher geschrieben.
- **RAM** oder auch **Arbeitsspeicher/Speicherwerk** speichert Daten und Programme und macht sie dem Rechenwerk zugänglich.
- **I/O Unit** oder auch **Eingabe-/Ausgabewerk** dient der Steuerung von Eingaben des Benutzers (z.B. Tastatur, Maus) und der Ausgabe von Daten (z.B. Monitor).

Das Steuer- und das Rechenwerk bilden zusammen den **Prozessor**. Der Hauptspeicher ist in Zellen gleicher Größe unterteilt, welche durchgehend adressierbar sind.
Ein Programm ist als Folge von Befehlen aufgebaut, welche nacheinander (sequentiell) ausgeführt werden, es sei denn es wird von dieser Reihenfolge, durch sogenannte **Sprungbefehle** abgewichen. Fallunterscheidungen, Wiederholungen und Aufrufe von zusammengesetzten Befehlen greifen ebenfalls in die sequentielle Abfolge der Befehle ein.

3.1.2 Typen von Programmiersprachen

Es gibt drei Arten der Übersetzung von Programmiersprachen.

Compilersprachen	Anweisungen werden einmal kompiliert, also in Maschinensprache übersetzt und anschließend in der ausgeführt.
Interpretersprachen	Ein Interpreter übersetzt Anweisungen sobald sie ausgeführt werden sollen.
Hybridsprache	Anweisungen werden in eine Zwischensprache übersetzt, die gut für Interpretation geeignet ist.

Java ist eine Hybridsprache, bei der der Quellcode zunächst in **Java Bytecode** kompiliert wird und anschließend an die **Java Virtual Machine (JVM)**, den Interpreter übergeben wird, welcher den Bytecode quasi wie Maschinencode liest.

Der Vorteil daran ist, dass der Bytecode auf jedem System funktioniert, wenn es eine entsprechende JVM für das System gibt. C-Programme bspw. müssen hingegen für jede Systemarchitektur neu kompiliert werden.

Die Imperative Programmierung baut auf dem von-Neumann Konzept auf, dabei sind Programme Folgen von Anweisungen. Die **Ausführungsreihenfolge** der Befehle wird durch die textuelle Reihenfolge und Sprunganweisungen festgelegt. Dabei bestimmen höhere Programmkonstrukte (zusammengefasste Anweisungsfolgen) die Ausführungsreihenfolge.

3.2 Literale, Operatoren und Ausdrücke

Literale sind feststehende Werte, welche direkt im Quellcode stehen und denen ein bestimmter Typ zugewiesen ist. So sind bspw. ganze Zahlen vom Typ `int`.

Operatoren meint Operatorzeichen (z.B. "`+`") und die Operation, welche dahinter steht (z.B. "Addieren").

Ausdruck ist ein anderes Wort für Term und meint eine Abfolge von Operanden und Operatoren, welche berechenbar sind. Das sind häufig arithmetische und logische Ausdrücke (z.B. "`4 + 5`", "`true && false`").

3.2.1 Operatoren

Position von Operatoren

Infix	Operator steht zwischen zwei Operanden	z.B. `9 - 6`
Präfix	Operator steht vor seinem Operanden	z.B. `-1`
Postfix	Operator steht nach seinem Operanden	z.B. `3!`

Stelligkeit von Operanden

Unär (einstellig)	z.B. `9!`
Binär (zweistellig)	z.B. `9-6`
Ternär/triadisch (dreistellig)	**if** `Operand1` **then** `Operand2` **else** `Operand3`

Präzedenz von Operatoren: Drück aus, wie stark Operanden von einem Operator "gebunden" werden:

arithmetische Funktionen

Operator	Funktion	Präzedenzgruppe
`*`	Multipliaktion	2
`/`	Division	2
`%`	Modulo	2
`+`	Addition	3
`-`	Subtraktion	3
`++`	Inkrement	1
`--`	Dekrement	1

boolesche Funktionen

Operator	Funktion	Präzedenzgruppe
`!`	logisches nicht	1
`&&`	logisches und	10
`||`	logisches oder	11
`<`	kleiner als	5
`<=`	kleiner gleich	5
`>`	größer als	5
`>=`	größer gleich	5
`==`	Gleichheit	6
`!=`	Ungleichheit	6

3.2.2 Variablen

Variablen sind eine Abstraktion eines physischen Speicherplatzes und werden durch einen Namen (Bezeichner) angesprochen/benutzt. Die **Belegung** ist der aktuelle Inhalt (z.B. die Zahl die in der Variablen gespeichert wird).

Deklaration und Initialisierung

Vor der Verwendung müssen Variablen deklariert werden, dies geschieht durch die Angabe des **Typs** (z.B. `int`) und der Vergabe eines **Namen** über einen **Bezeichner**.
Durch die reine Deklaration ist die Belegung der Variablen **undifiniert**. Durch die **Initialisierung** wird die Variable erstmals mit einem gültigen Wert befüllt.

Deklaration	Deklaration und Initialisierung
`int i;`	`int i = 42;`
`boolean b;`	`boolean b;`
	`b = true;`

Variablenwerte sind, wie der Name sagt, variabel (Ausnahme sind bspw. der Typ `const` in JavaScript). D.h. die Belegung einer Variable kann überschrieben werden.

```
1  int i;
2  i = 12;
3  i = 15;
4  i = i - 8;
```

Zuweisung

Es ist außerdem möglich, das Ergebnis eines ausgewerteten Ausdrucks einer anderen Variable zuzuweisen. Hierbei muss der Typ der Zielvariable zu dem des Ausdrucks passen.
Syntax:

Linke Seite (LHS)	Mitte	Rechte Seite (RHS)
Bezeichner	Zuweisungsoperator	Ausdruck (Term)
`i`	`=`	`i * 35 + 42 - 100`

3.3 Prozeduren

Prozeduren sind eine Anweisungsfolge in Kombination mit einem Namen und Parametern, also Speicheradressen von Speicherzellen von Eingabedaten. Sie kann, ähnlich wie ein zusammengesetzter Befehl aufgerufen werden, wobei Werte als Parameter mitgegeben werden können.
Prozeduren können genutzt werden, um Anweisungsfolgen treffend zu bennen und die Lesbarkeit des Codes zu erhöhen.

Eine Prozedur hat einen **Rückgabetyp**, sollte dieser nicht `void` sein, so muss diese einen **Rückgabewert** mit `return` zurück geben, der nach Auswertung der Prozedur in einem Ausdruck außerhalb verwendet werden kann.

Wird eine Prozedur aufgerufen, so wird in einer sequenziellen, imperativen Sprache der Programmablauf unterbrochen bzw. zur Prozedur gesprungen und anschließend wieder in den regulären Programmablauf zurückgesprungen. Werden in Prozeduren andere Prozeduren aufgerufen (oder ruft sich die Prozedur selbst auf), so entsteht eine Aufrufskette.

```
1  // Prozedurdefinition; a, b, c sind formale Parameter
2  int minimumAusDrei(int a, int b, int c)
3  {
4      if(a < b && a < c)
5      {
6          // gibt einen Wert zur"uck und stoppt die Prozedur
7          return a;
8      }
9      if(b < c)
10     {
11         return b;
12     }
13     return c;
14 }
15 ...
16 // Prozeduraufruf mit aktuellen Parametern, speichert Ergebnis in
       Variable
17 int result = minimumAusDrei(7, -13, 3);
```

Die Anzahl der **aktuellen Parameter** beim Aufruf muss gleich der Anzahl der **formalen Parameter** sein, als auch **typkompatibel** sein. Außerdem ist die Position (Reihenfolge) der aktuellen Parameter relevant.

4. Objektorientierte Programmierung

4.1 Kernbegriffe

Quelltext (source code)	Programmtext, auch Klassendefinition (class definition)
Objekt (object)	Ein belibiges Exemplar einer Klasse
Exemplar (instance)	Ein bestimmtes Objekt einer Klasse
interener Zustand	Belegungen der Felder eines Exemplars
Klasse (class)	Definiert für ihre Exemplare, welche Methoden an ihnen aufrufbar sind
Methode (method)	Schnittstelle einer Klasse, aufrufbar an Exemplaren
sondierende Methode	Gibt Variablenwerte zurück ohne sie zu verändern
verändernde Methode	Verändert Variablen
öffentliche Methode	Mit `public` deklarierte Methode; außerhalb der Klasse aufrufbar
private Methode	Mit `private` deklarierte Methode; nur innerhalb der Klasse aufrufbar
Kopf (header)	Signatur einer Methode
Signatur	Name der Methode und Reihenfolge der Typen der Parameter
Rumpf (body)	Anweisungen und Deklarationen einer Methode
Anweisungen	Definieren Aktionen eines Programms
Punktnotation (dot notation)	`objekt.methode([parameter])`
Übersetzen (compile)	Quelltext mit hilfe eines Compilers in maschinenausführbare Form überführen
Compiler	Programm, welches Quelltext übersetzt