Programmierung 1

Einführung

Dr. Monika Schak

Woche 1

22. Oktober 2025

Vom Problem zum Programm

Programmieren heißt, ein reales Problem so genau zu beschreiben, dass ein Computer es lösen kann.

1. **Verstehen**, was das Problem eigentlich ist.
2. Das Problem in kleine, lösbare Handlungsschritte **zerlegen**.
3. Diese Schritte beschreiben den **Algorithmus**.
4. Algorithmus in einer Programmiersprache (z.B. C) **umsetzen**.

Algorithmus

- Eindeutige Beschreibung eines Verfahrens zur Lösung bestimmter Problemklassen
  - Präzise, endliche Abfolge von elementaren, ausführbaren Anweisungen, um zu beschreiben wie ein Problem allgemeingültig gelöst werden kann.
  - Muss korrekt sein, d.h. für alle Eingaben richtige Ergebnisse liefern.
- Beispiel-Algorithmus *Tee kochen*

Unterschied Algorithmus (**was** wird getan) vs. Programm (**konkrete** Umsetzung)!

Eigenschaften eines Algorithmus

- **Determinismus:** Ein Algorithmus ist eindeutig, d.h. jeder Schritt ist klar definiert.
- **Terminierung:** Ein Algorithmus ist endlich, d.h. er muss irgendwann stoppen.
- **Effektivität:** Ein Algorithmus ist ausführbar, d.h. jeder Schritt ist so einfach, dass er tatsächlich von einer Maschine ausgeführt werden kann.
- **Generalisierbarkeit:** Ein Algorithmus ist allgemeingültig, d.h. er funktioniert für eine ganze Klasse von Problemen, nicht nur für ein bestimmtes Beispiel.

Darstellung von Algorithmen

- **Natürliche Sprache**: Umgangssprachliche, alltagsnahe Erklärung in Worten.
- **Pseudocode**: Formale Beschreibung, unabhängig von einer konkreten Programmiersprache.
- **Programmablaufplan**: Graphische Darstellung als Flussdiagramm mit Symbolen und Pfeilen.
- **Struktogramm**: Formalisierte graphische Darstellung von Programmentwürfen
- **Programmiersprache**: Konkrete, maschinenverständliche Umsetzung.

Darstellung von Algorithmen - Natürliche Sprache

Tiefkühlpizza backen - Umgangssprachliche Beschreibung eines Algorithmus

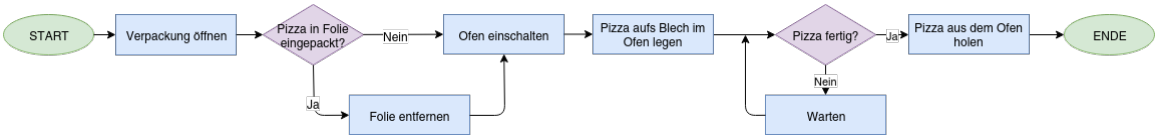
Öffne die Kartonverpackung und entnehme die Pizza. Falls die Pizza in Folie eingepackt ist, entferne die Folie. Schalte den Ofen ein, lege die Pizza in den Ofen und sobald die Pizza fertig ist, hole sie aus dem Ofen.

Darstellung von Algorithmen - Pseudocode

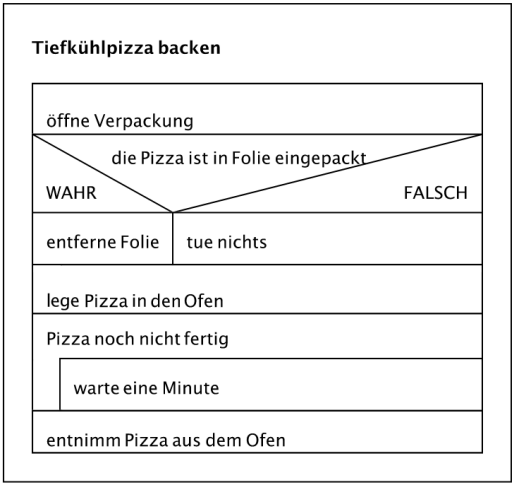
Tiefkühlpizza backen - Pseudocode

```
oeffne Verpackung
WENN Pizza in Folie eingepackt
  entferne Folie
schalte Ofen ein
lege die Pizza auf das Blech im Ofen

SOLANGE Pizza noch nicht fertig
  warte

hole die Pizza aus dem Ofen
```

Tiefkühlpizza backen - Programmablaufplan



Tiefkühlpizza backen - Struktogramm



Programmiersprache

- Formale Sprache, mit deren Hilfe ein Programm formuliert ist.
- Wird festgelegt durch:
  - **Syntax**: Form oder „Grammatik“ der Sprache
    - Bsp: „Bitte bring mir einen Kaffee.“ vs. „Kaffee mir bitte bringen.“
  - **Semantik**: beschreibt die Bedeutung eines „Satzes“ - was gemeint ist
    - Bsp: Jemand möchte einen Kaffee haben (nicht Kuchen oder Tee).
  - **Pragmatik**: Kontext und Absicht - wann, warum und wie?
    - Bsp: Im Café zur Kellnerin angemessen, zum Polizisten bei der Verkehrskontrolle eher nicht.

Programmiersprache C

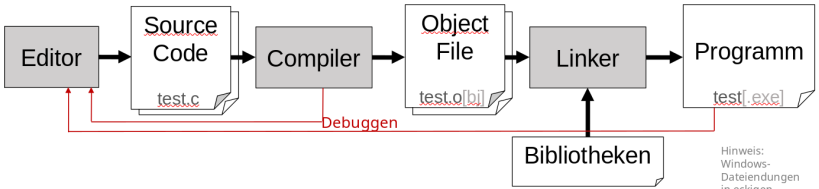
- Anfang der 70er Jahre von Dennis Ritchie entwickelt
- Weltweit eine der wichtigsten Programmiersprachen
- **Eigenschaften:** Kompiliert, imperativ, maschinennah, portabel
- Besonders gut für den Einstieg in die Programmierung geeignet, da eher technische Sprache mit wenigen Schlüsselwörtern, bei der wenig versteckt im Hintergrund passiert.

Programm und Maschine

- Anwendungsprogramme benötigen als *Grundprogramm* ein Betriebssystem (z.B. Windows oder Linux)
- Ein darauf laufendes Übersetzungsprogramm (i.d.R. Compiler) übersetzt das in Programmiersprache gegebene Quellprogramm in Maschinensprache
  - Bei C erzeugt man für jede C-Datei (*test.c*) eine Objektdatei (*test.o[bj]*)
    Unter Linux z.B.: `gcc -c test.c` erzeugt aus der Datei *test.c* die Datei *test.o*
  - Dann werden alle Objektdateien und externe Bibliotheken (Module, die Funktionalitäten bereitstellen) mit Hilfe des sog. Linkers zum ausführbaren Program zusammengebunden
    Unter Linux z.B.: `gcc -o test test.o` erzeugt das Programm *test*
  - Kurzform: `gcc -o test test.c`

Kompilieren und Linken

- Quelltext in (Text-)Dateien organisiert, Dateiendungen: `.c`, `.h`
- Daraus kompilierte Objektdateien bestehen aus Maschinenbefehlen, Dateiendung: `.o` bzw. `.obj`
- Linker bindet alles zu ausführbarem Programm zusammen, ohne Dateiendung bzw. Dateiendung `.exe`
- Beim Programmieren passieren Fehler, diese muss man finden und verbessern → sogenanntes Debugging



Programmierumgebung

- Theoretisch reichen Texteditor (z.B. vi) und Compiler (z.B. gcc)
  Unter Linux lange üblich, wird selbst heute noch genutzt

- Komfortabler sind sog. IDEs
  Beinhalten Texteditor mit Syntax Highlighting, sowie Compiler mit Linker und Debugger
  - **CLion** (alle Plattformen)
  - XCode (Mac)
  - Microsoft Visual Studio Code, Code:Blocks, o.ä. (Windows)
  - u.v.m.

Das erste C-Programm

- Speichern in (Quell-)Textdatei, z.B. `helloWorld.c`
- Kompilieren (inkl. Linken) mit `gcc -o helloWorld helloWorld.c`
- Ausführen mit `./helloWorld`
  Hinweis: ausführbare Programme haben nur unter Windows die Dateiendung `.exe`

Das erste C-Programm

- Der wichtigste Teil ist der Rumpf des Hauptprogramms, also der Funktion main
- Bei Ausführung startet ein C-Programm stets mit der Funktion main
- Die Funktion printf ist vordefiniert und schreibt ihr Argument auf die Konsole
- Hier ist das Argument eine Zeichenkette (= String)
- Die Schlüsselwörter int und return legen fest, was von main zurückgegeben wird

```c
#include <stdio.h>

int main()
{
    printf("Hello World!\n");

    return 0;
}
```

Ausführung eines C-Programms

- Variante 1: Ausführen auf Betriebssystem
  - Hardware durch Betriebssystem gekapselt: Starten/Beenden des Programms über das Betriebssystem
  - Dabei Unterstützung durch Betriebssystem-Funktionen, z.B. für den Dateizugriff
- Variante 2: Ausführung direkt auf dem Prozessor
  - Spezielle Mikrocontroller, z.B. für Temperaturregelung
  - Alle Funktionen müssen im Programm vorhanden sein

Funktionen und Variablen

- Wie in Mathe werden Funktionen nicht nur mit Argumenten aus einem passendem Definitionsbereich befüttert, z.B. y = sin(x), sondern sie können auch etwas in entsprechendem Wertebereich zurückgegeben.
  - Bei `main()` ist dieser Wertebereich ganzzahlig, wofür das Schlüsselwort `int` steht
  - Die Rückgabe erfolgt über das Schlüsselwort `return`, wobei hier die ganze Zahl 0 zurückgegeben wird
- Wie in Mathe gibt es auch Variablen, die als Platzhalter dienen für eine Zahl aus einem Wetebereich (z.B. Ganzzahlen) bzw. beim Programmieren sogar für allgemeine Daten wie z.B. Zeichenketten
  - Variablen haben eine eindeutige Bezeichnung, einen eindeutigen Datentyp (Wertebereich) und zur Laufzeit einen Ort (Adresse) im Hauptspeicher, an dem der Wert der Variablen steht
  - Beispiel: `int zahl = 42;`

Sprachelemente

- In C gibt es etwas mehr als 30 Schlüsselwörter, welche man, inkl. Bedeutung und Anwendung, auswendig kennen sollte. Außerdem gibt es etliche Operatoren (z.B. +, -, *, / oder =) mit fester Bedeutung.
- Die Namen von Funktionen und Variablen sind selbstgewählt, dürfen aber nur Buchstaben aus ASCII-Zeichensatz (keine Umlaute), Ziffern und Unterstrich (keine sonstigen Sonderzeichen oder Leerzeichen) enthalten. Ziffern dürfen nicht das erste Zeichen des Bezeichners sein, Groß- und Kleinschreibung wird unterschieden.
- Kommentare sind Ergänzungen im Quelltext, die dazu dienen, selbigen möglichst verständlich zu gestalten. Sie werden durch eine spezielle Zeichenfolge eingeleitet und vom Compiler nicht übersetzt.

Schlüsselwörter in C

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

Grau gefärbte Schlüsselwörter sind z.T. seltener gebräuchlich und nicht klausurrelevant.
Blau gefärbte Schlüsselwörter bezeichnen nur verschiedene Datentypen (z.B. int)
* Inkludiert man in der C-datei stdbool.h, bekommt man noch folgende Pseudo-Schlüsselwörter bool, true, false

Variablen

```
int i = 23, j;
int zahl = 42;
```

Bezeichner	Datentyp	Speicheradr	Speicherinhalt
i	int	48000	23
j	int	48001	?
zahl	int	48002	42

- Dienen dazu, um Daten an der jeweiligen Adresse im Hauptspeicher abzuspeichern.
- Müssen vor Gebrauch eingeführt, d.h. deklariert, werden. Im Beispiel werden drei Variablen **deklariert** und dann mit Werten **initialisert**, d.h. mit einem Startwert versehen. Wird eine Variable nicht initialisiert, befindet sie sich in undefiniertem Zustand (d.h. der Wert ist Speichermüll).
- Schlüsselwort `int` besagt, dass es sich um ganze Zahlen handelt, der Datentyp der Variable ist damit `int`.
- Am Ende einer Deklaration steht ein Semikolon.

Zuweisungen

- Enden auch mit Semikolon
- Modifizieren die Werte von Variablen
- Beispiele:
  - `zahl = 42;`
    Variable `zahl` erhält den Wert 42, am zugehörigen Speicherplatz liegt 42 als Bitmuster `0010 1010`
  - `j = i + zahl;`
    Werte von `i` und `zahl` werden ermittelt, addiert und der Variable `j` zugewiesen
  - `j = j + 1; //erhoehe j um 1`
    In dieser Zuweisung greift `j` auf der rechten Seite des Zuweisungsoperators auf den Wert vor der Zuweisung zu, woraufhin 1 addiert wird

Beispiel

```
int n1, n2, n3;

n1 = 5;
n2 = n1 * n1;
n3 = n2 * n2;
```

Wert der Variablen

n1	n2	n3
?	?	?

Arithmetische Ausdrücke

Auf der rechten Seite von Zuweisungen können Ausdrücke stehen

- Konstanten: Werte, die keine Variablen sind, z.B. 4711
- Zwei Ausdrücke mit einem Operator: `+`, `-`, `*`, `/`
  Vorsicht bei Division (`/`): Sind beide Werte ganzzahlig (`int`), handelt es sich um eine ganzzahlige Division (Nachkommazahl wird nicht berücksichtigt), wird das Ergebnis einer Integer-Variablen zugewiesen, ist es auch eine ganzzahlige Division.
  Modulo-Operation (`%`): Gibt den Rest bei ganzzahliger Division.
- Geklammerter Ausdruck, z.B. $2 * (17 + 4)$

Beispiele:

```
zahl = 2 * (17 + 4) - 7 / 3;
zahl = 23 / 5;
zahl = 23 % 5;
wert = zahl + 27;
```

Beispiel-Programm

```
#include <stdio.h>
#include <stdbool.h>

int main() {
  int i = 23, j;
  int zahl = 42;

  j = i + zahl;  // i = 23, zahl = 42
  j = j + 1; // erhoehe j um 1
  printf("j hat Wert %d\n", j);

  zahl = 2 * (17 + 4) - 7 / 3;
  printf("zahl hat Wert %d\n", zahl);

  return 0;
}
```

- Funktion `printf()` schreibt Argument(e) auf die Konsole. Das erste Argument ist immer eine Zeichenkette, optional sind weitere Argumente durch Komma getrennt möglich. Die weiteren Argumente sind Ausdrücke, deren Werte man ausgeben möchte.
- Erlaubt eine formatierte Ausgabe: Platzhalter `%d` steht für eine ganze Zahl (`int`) als Dezimalzahl. Für jeden weiteren Ausdruck ist je ein Platzhalter nötig.

Sequenz

- Jede Zeile im Rumpf des Hauptprogrammes ist eine Anweisung, die stets mit einem Semikolon beendet wird
- Es können mehrere Zuweisungen, Deklarationen oder Ausgaben hintereinander stehen - sind damit eine Sequenz
- Zu jedem Zeitpunkt wird nur genau eine Anweisung ausgeführt, und zwar genau eine
- Reihenfolge ist wie im Quelltext angegeben
- Nach der letzten Anweisung endet das Programm, meistens: `return 0;`