CARL VON OSSIETZKY universität OLDENBURG

Teil

Imperative Programmierung

Unterrichtseinheit 9

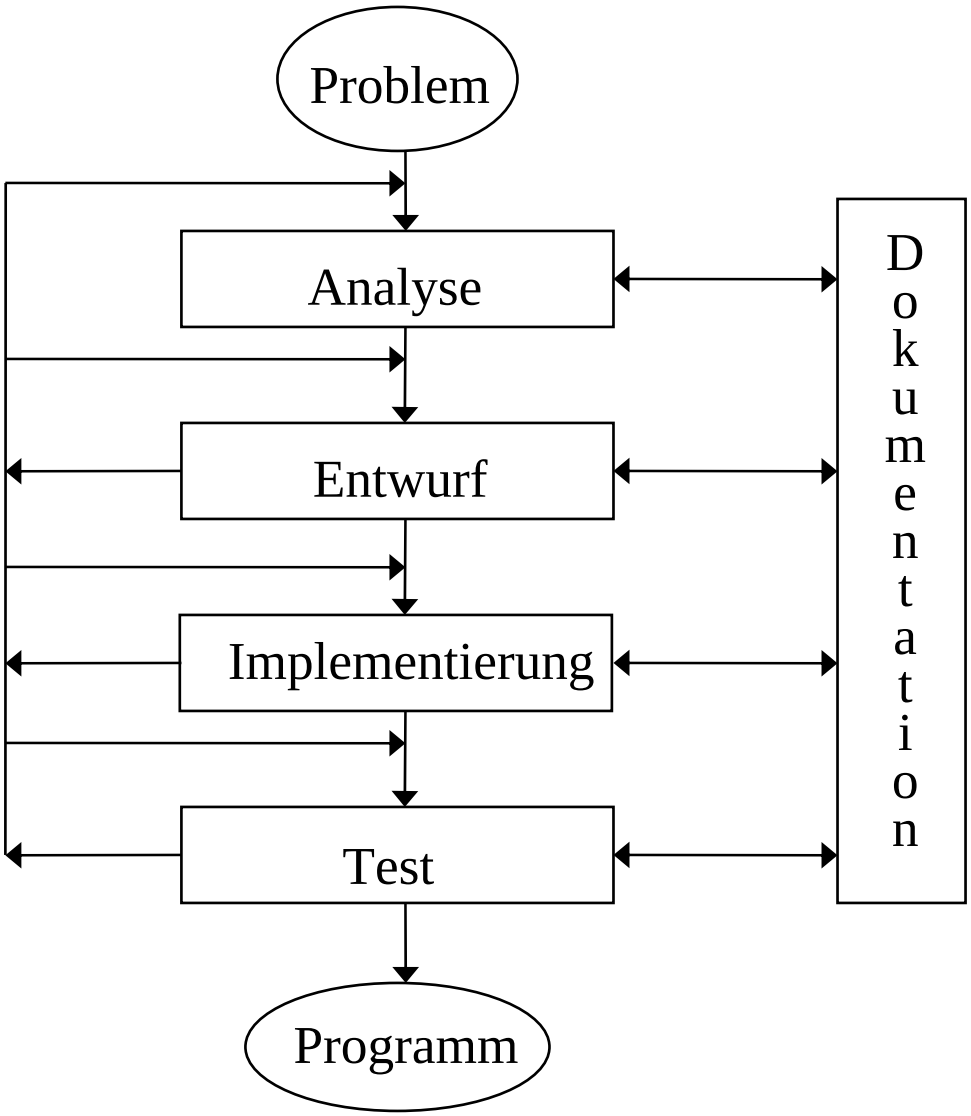
Programmentwurf (Hamster-Modell)

Dr. Dietrich Boles

Gliederung

- Entwicklungsphasen
- Problem
- Analyse
- Entwurf
- Implementierung
- Test
- Dokumentation
- Zusammenfassung

Entwicklungsphasen



Analyse:

exakte Formulierung der Aufgabe

Entwurf:

Entwicklung eines Algorithmus

Implementierung:

Codierung;

Eingabe in den Rechner;

Compilation

Test:

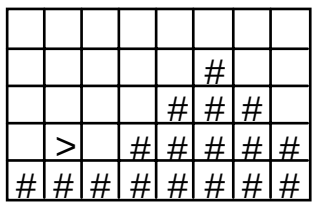
Verifikation der Korrektheit

Dokumentation:

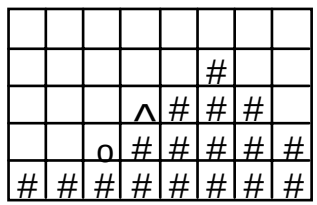
Nachlass für andere Personen

Problem

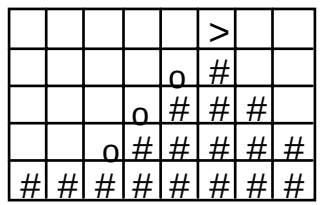
Der Hamster steht vor einem Berg unbekannter Höhe. Er soll den Gipfel erklimmen und auf dem Gipfel anhalten. Auf dem Weg zum Gipfel soll er auf jeder Stufe genau ein Korn ablegen.



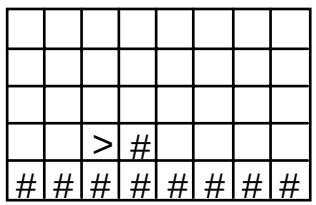
(a) zu Anfang



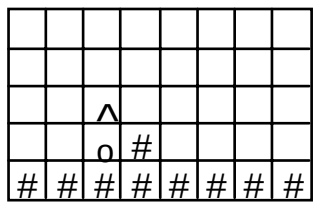
(a) zwischendurch



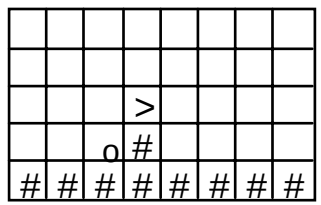
(a) am Ende



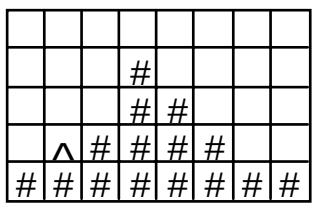
(b) zu Anfang



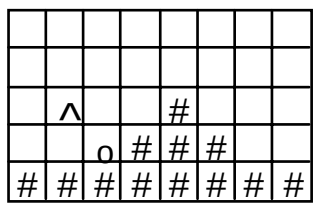
(b) zwischendurch



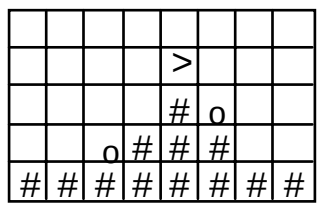
(b) am Ende



(c) zu Anfang



(d) zwischendurch



(e) am Ende

Analyse

- Anfangsgrößen / Eingabewerte
  - Position des Hamsters
  - Blickrichtung des Hamsters
  - Anzahl Körner im Maul des Hamsters
  - Körner im Territorium
  - Mauern im Territorium
- Endgrößen / Ausgabewerte
  - Position des Hamsters
  - Blickrichtung des Hamsters
  - Anzahl Körner im Maul des Hamsters
  - Körner im Territorium
- Constraints bez. des Lösungsweges

Präzisierung der Aufgabe:

Der Hamster steht mit Blickrichtung Ost vor einem regelmäßigen Berg unbekannter Höhe (ohne Überhänge). Er muss nicht unbedingt direkt vor dem Berg stehen. Die Stufen des Berges sind jeweils eine Mauer hoch. Der Hamster habe mindestens so viele Körner im Maul, wie Stufen existieren. Auf dem Territorium befinden sich keine Körner. Außer den Mauern des Berges befinden sich keine Mauern im Territorium.
Der Hamster soll den Berg erklimmen und auf dem Gipfel anhalten. Auf jeder Stufe - und nur dort! - soll er ein Korn ablegen. Auf dem Weg zum Gipfel muss er immer in Berührung zur Wand bleiben.

Entwurf (1)

- Verfahren: Schrittweise Verfeinerung / Top-Down-Entwurf
- Ziel: Komplexitätsreduktion
- Prinzip:
  - Gesamtproblem zu komplex => Aufteilung in einfachere Teilprobleme
  - Lösen der Teilprobleme:
    - Teilproblem zu komplex => Aufteilung in (noch) einfachere Teilprobleme
    - ...
    - Zusammensetzung der Lösungen der Teilprobleme zur Lösung des (übergeordneten) Teilproblems
  - Zusammensetzung der Lösungen der Teilprobleme zur Lösung des Gesamtproblems
- Vergleich: Puzzlen

Entwurf (2)

Gesamtproblem:

Der Hamster soll bis zum Berg laufen und dann den Berg erklimmen.

Teilprobleme:

- Der Hamster soll bis zum Berg laufen.
- Der Hamster soll den Berg erklimmen.

```
// der Hamster soll bis zum Berg laufen und dann den
// Berg erklimmen
void main() {
    laufeZumBerg();
    erklimmeDenBerg();
}

// der Hamster soll bis zum Berg laufen
void laufeZumBerg() {}

// der Hamster soll den Berg erklimmen
void erklimmeDenBerg() {}
```

Entwurf (3)

Teilproblem 1: Der Hamster soll bis zum Berg laufen.

```
// der Hamster soll bis zum Berg laufen
void laufeZumBerg() {
    while (vornFrei()) {
        vor();
    }
    gib();
}
```

Teilproblem 2: Der Hamster soll den Berg erklimmen.

```
// der Hamster soll den Berg erklimmen
void erklimmeDenBerg() {
    while (!gipfelErreicht()) {
        erklimmeEineStufe();
    }
}

void erklimmeEineStufe() {}
boolean gipfelErreicht() {}
```

Teilproblem 2.1

Teilproblem 2.2

Entwurf (4)

Teilproblem 2.1: Der Hamster soll eine Stufe erklimmen.

```
void erklimmeEineStufe() {
    linksUm(); vor();
    rechtsUm(); vor();
    gib();
}

void rechtsUm() {
    linksUm(); linksUm(); linksUm();
}
```

Teilproblem 2.2: Ist der Gipfel erreicht?

```
boolean gipfelErreicht() {
    return vornFrei();
}
```

Implementierung (1)

- Codierung in die Entwurfsphase integriert
- Restaufgaben:
  - Eingabe des Programms in den Rechner
  - Compilation
- Anmerkungen / Hinweise:
  - kein Spaghetti-Code!
  - Entwurf mit Bleistift und Papier!
  - Auf übersichtliche Strukturierung achten!
  - Kommentare verwenden!
  - Prozeduren / Funktionen verwenden!
  - Aussagekräftige Bezeichner wählen!

Implementierung (2)

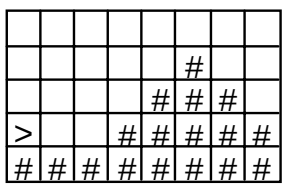
korrekte aber schlechte Lösung:

```
void main() {  while
(vornFrei()) vor(); gib(); while (!vornFrei()) {
      linksUm(); vor(); r(); vor(); gib(); }
} void r() { linksUm();linksUm();linksUm();}
```

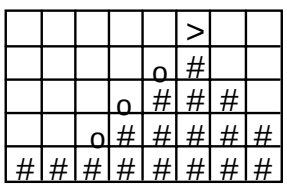
Test

- Verifikation der Korrektheit und Vollständigkeit
- Konstruktion einer Testmenge:
  - disjunkte Zerlegung der Menge der zulässigen Anfangsgrößen in typische Klassen
  - Vertreter jeder Klasse wählen
  - Grenzwerte beachten
  - erwartetes Ergebnis notieren
  - Erfahrung und Intuition notwendig!
- Ausführung des Programms mit allen Elementen der Testmenge:
  - Ergebnisse (Lösungsweg/Endgrößen) überprüfen
  - Testläufe/Ergebnisse protokollieren

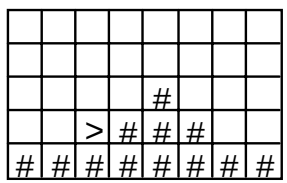
Demo



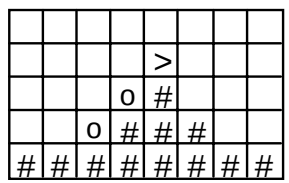
(a) zu Anfang



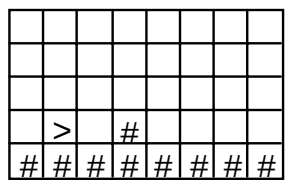
(a) am Ende



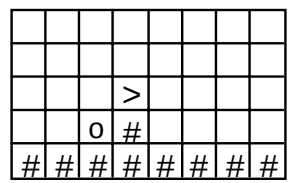
(b) zu Anfang



(b) am Ende



(c) zu Anfang



(c) am Ende

Dokumentation

Zur Dokumentation gehören:

- eine exakte Problemstellung
- eine verständliche Beschreibung des entwickelten Algorithmus
- der Programmcode
- die gewählte Testmenge mit Protokollen der durchgeführten Testläufe
- eine Beschreibung von aufgetretenen Problemen
- alternative Lösungsansätze

Zusammenfassung

- Bei der Entwicklung von Programmen werden mehrere Phasen durchlaufen
  - Analyse: Präzisieren der Aufgabenstellung
  - Entwurf: Entwicklung des Lösungsalgorithmus
  - Implementierung: Eingabe des Sourcecodes in den Rechner
  - Test: Suchen und Beheben von Fehlern
  - Dokumentation: verständliche Beschreibung des Programms sowie des Entwicklungsweges

- Entwurfsverfahren (bei kleinen Problemen!)
  - Top-Down-Entwurf / prozedurale Zerlegung
    - schrittweise Zerlegung des Problems in Teilprobleme
    - Lösung genügend kleiner Teilprobleme durch Prozeduren bzw. Funktionen