Übung zur VL „Grundlagen der Programmierung“

1. Übung (20/21.4.)

Dr. Zubow

Heutige Themen

- “99 Bottles of Beer” Programm
- Problem des Handlungsreisenden
- Ein weiteres bekanntes Problem der Informatik – „Türme von Hanoi“
- Beweisverfahren: Vollständige Induktion
- Besprechung des 1. UE-Blattes

Programmiersprachen

- Assembler (Intel 8086)

```
; 99 Bottles of Beer program in Intel 8086 assembly language.
; Assemble to a .COM file for MS-DOS.
; ; Author: Alan deLespinasse

code segment
assume cs:code,ds:code
org 100h
start:
; Main loop
mov cx, 99                      ; bottles to start with
loopstart:
call printcx            ; print the number
mov dx,offset line1             ; print the rest of the first line
mov ah,9                        ; MS-DOS print string routine
int 21h
call printcx            ; print the number
mov dx,offset line2_3   ; rest of the 2nd and 3rd lines
mov ah,9
...
```

Programmiersprachen

- Fortran 90

```
!  F90 (Fortran 90) version of 99 bottles of beer.
!  written by Akira KIDA,  SDI00379@niftyserver.or.jp
!  Note that this source is in FIXED format.

      program ninetynine
      implicit none
      integer, parameter :: BOTTLES = 99
      integer :: i
      integer :: k
      character*7 :: btl = 'bottles'

      do i = BOTTLES, 1, -1
            k = len(btl)
            if (i == 1) k = k - 1
            print *, i, btl(1:k), ' of beer on the wall, ',
     c                    i, btl(1:k), ' of beer.'
            print *, 'Take one down, pass it around.'
            if (i == 0) exit
            print *, i, btl(1:k), ' of beer on the wall.'
      end do
      print *, 'No more bottles of beer on the wall.'
      end
```

Programmiersprachen

- Pascal

```
(* Pascal version of 99 Bottles of beer *)
program BottlesOfBeers(input, output);
var
  bottles: integer;
begin
  bottles := 99;
  repeat
    WriteLn(bottles, ' bottles of beer on the wall, ',
          bottles, ' bottles of beer.');
    Write('Take one down, pass it around, ');
    bottles := bottles-1;
    WriteLn(bottles, ' bottles of beer on the wall.');
  until (bottles = 1);
  WriteLn('1 bottle of beer on the wall, one bottle of beer.');
  WriteLn('Take one down, pass it around,'
        ' no more bottles of beer on the wall')
end.
```

Programmiersprachen

- SQL

```
remark      99 bottles of beer with Oracle SQL*Plus
remark      R.vandePol@voeding.tno.nl        < Rob van de Pol>
remark
remark      assuming that YourTable contains at least 99 rows ;-)
remark
remark      RowNum is an Oracle psuedo column indicating the sequence the rows
remark      were selected.

SELECT      TO_CHAR(100-rownum)||' bottles of beer on the wall, ' ||
                        TO_CHAR(100-rownum)||' bottles of beer,' ,
            'take one down and pass it around, ' ,
            DECODE ( TO_CHAR(99-rownum) , '0' , 'No more' , TO_CHAR(99-rownum) )||
                        ' bottles of beer,'
FROM        YourTable
WHERE       rownum < 100
```

Programmiersprachen

- LISP

```
;;; Lisp example of "99 Bottles of beer on the wall"
;;;
;;; NOTE:  Although my mailer insists on inserting
;;; (at least) one, there is no line break in the
;;; string beginning "~~  (i.e. it should all be on one line).
;;;
;;; In particular, if it breaks so that the first line
;;; ends with "...~~R" and the second line starts "~0@..."
;;; they should be put back together with a space between
;;; them.  That is, it should read "...~~R ~0@...".
;;; Or just see it here:
;;;    http://www.sover.net/~nichael/lisp99.html

(labels ((foo (x)
  (and (<= 0 x) (cons x (foo (1- x))))))
  (format t (format nil
     "~~{~~&~~@(~~%~~R ~A ~A!~~)~~:*~~&~~@(~~R ~0@*~A!~~)~~&~~
                @(~2@*~A!~~)~~&~~@(~~[~A~~:;~~:*~~R~~:*~~] ~0@*~A!~~)~~}"
                  "bottles of beer"
                  "on the wall"
                  "take one down, pass it around"
                  "no more"
                  )
 (foo 99)))
```

Programmiersprachen

- Java

```
// java version of 99 bottles of beer on the wall
// 1995 Sean Russell (ser@cs.uoregon.edu)

class bottles
{
 public static void main(String args[])
  {
  String s = "s";
  for (int beers=99; beers>-1;)
   {
   System.out.print(beers + " bottle" + s + " of beer on the wall, ");
   System.out.println(beers + " bottle" + s + " of beer, ");
   if (beers==0)
    {
    System.out.print("Go to the store, buy some more, ");
    System.out.println("99 bottles of beer on the wall.\n");
    System.exit(0);
    }
   else
    System.out.print("Take one down, pass it around, ");
   s = (--beers == 1)?"":"s";
   System.out.println(beers + " bottle" + s + " of beer on the wall.\n");
   }
  }
}
```

Problem des Handlungsreisenden

- Fragestellung,
- Optimale Lösung
  - Probleme?
- Heuristiken
  - „**Heuristik** (altgr. εὑρίσκω *heurísko* „ich finde“; *heuriskein*, „(auf-)finden“, „entdecken“) bezeichnet die Kunst, mit begrenztem Wissen und wenig Zeit zu guten Lösungen zu kommen.“, wikipedia.de

Türme von Hanoi

- Das Spiel besteht aus drei Stäben *A*, *B* und *C*, auf die mehrere gelochte Scheiben gelegt werden, alle verschieden groß.
- Zu Beginn liegen alle Scheiben auf Stab *A*, der Größe nach geordnet, mit der größten Scheibe unten und der kleinsten oben.
- Ziel des Spiels ist es, den kompletten Scheiben-Stapel von *A* nach *C* zu versetzen.
- Bei jedem Zug darf die oberste Scheibe eines beliebigen Stabes auf einen der beiden anderen Stäbe gelegt werden, vorausgesetzt, dort liegt nicht schon eine kleinere Scheibe (→ zu jedem Zeitpunkt sind die Scheiben auf jedem Feld der Größe nach geordnet).

Türme von Hanoi (2)

- Lösung über Rekursion (analog zu TSP).
- Algorithmus:
  - Funktion *bewege()* besitzt 4 Parameter
  - Mit *i* ist die Anzahl der zu verschiebenden Scheiben bezeichnet, mit *a* der Stab von dem verschoben werden soll, mit *b* der Stab, der als Zwischenziel dient und mit *c* der Stab, auf den die Scheiben verschoben werden sollen.
  - Zur Lösung des eigentlichen Problems wird *bewege* mit *i*=n, *a*=*A*, *b*=*B* und *c*=*C* aufgerufen.

Türme von Hanoi (3)

- Algorithmus:
  - Funktion *bewege* löst ein Teilproblem dadurch, dass es dieses in 3 einfachere Probleme aufteilt, sofern der zu verschiebende Turm mindestens die Höhe 1 besitzt.
  - Andernfalls macht *bewege* nichts.

```
funktion bewege (Zahl i, Stab a, Stab b, Stab c) {
        falls (i > 0) {
                bewege(i-1, a, c, b);
                verschiebe oberste Scheibe von a nach c;
                bewege(i-1, b, a, c);
        }
}
```

Türme von Hanoi (4)

- So verhält sich die Funktion bei 3 Scheiben:

```
bewege(3,1,2,3) {
        bewege(2,1,3,2) {
                bewege(1,1,2,3) {
                        bewege(0,1,3,2){};
                        verschiebe oberste Scheibe von 1 nach 3;
                        bewege(0,2,1,3){};
                };
                verschiebe oberste Scheibe von 1 nach 2;
                bewege(1,3,1,2){
                        bewege(0,3,2,1){};
                        verschiebe oberste Scheibe von 3 nach 2;
                        bewege(0,1,3,2){};
                };
        };
        verschiebe oberste Scheibe von 1 nach 3;
        ...
};
```

Beweisverfahren: Vollständige Induktion

- Wikipedia: „Vollständige Induktion […] ist eine mathematische Beweismethode, die üblicherweise eine Aussage für alle natürlichen Zahlen beweist (verallgemeinert).“
- Idee:
  - Ist bekannt,
    - dass eine bestimmte von *n* abhängige Aussage für $n = 1$ gilt und
    - dass für jede beliebige natürliche Zahl *k* aus der Gültigkeit der Aussage für $n = k$ auch die Gültigkeit für $n = k + 1$ folgt,
  - dann folgt nach dem Induktionsaxiom, dass diese Aussage für *alle* natürlichen Zahlen *n* gilt.

Vollständige Induktion - Beispiel

- Summe der ungeraden Zahlen:
  - 1=1, 1+3=4, 1+3+5=9, 1+3+5+7=16
  - Es soll eine Formel gefunden werden!
  - *Vermutung:* Die Summe aller ungeraden Zahlen von 1 bis $2n - 1$ ist gleich dem Quadrat von *n*:

$$\sum_{i=1}^{n}(2i-1) = n^2$$

Beispiel (2)

- Beweis durch vollständige Induktion über *n*:
  - Induktionsanfang (IA): $n=1$

$$\sum_{i=1}^{1}(2i-1) = 1^2$$

  - Induktionsschritt (IS): Wenn es gilt für *n* so auch für *n+1*

Wenn $\sum_{i=1}^{n}(2i-1) = n^2$, so ist $\sum_{i=1}^{n+1}(2i-1) = (n+1)^2$

Beispiel (3)

- Beweis:
  - IA: $\sum_{i=1}^{1}(2i-1)=1=1^2$
  - IS:

$$\sum_{i=1}^{n+1}(2i-1)=\sum_{i=1}^{n}(2i-1)+(2(n+1)-1)=$$

$$n^2+2(n+1)-1=n^2+2n+1=(n+1)^2$$

Hinweise: 1. binomische Formel

Besprechung des 1. UE-Blattes

- Aufgabe 1
  - Ges.: ist ein Algorithmus (= Folge von Rangierschritten)
- Aufgabe 2
  - Problem des Handlungsreisenden
  - Ges.: günstigster Pfad mit Hilfe einer Heuristik
- Aufgabe 3
  - Anforderungsanalyse
- Abgabe: 3. Mai, 8.00

Anforderungsanalyse

- Anforderungen an ein zu konstruierende System:
  - Funktionale Anforderungen
    - Ziele, Aufgaben, Aktionen
  - Nichtfunktionale Anforderungen
- Anwendungsfälle

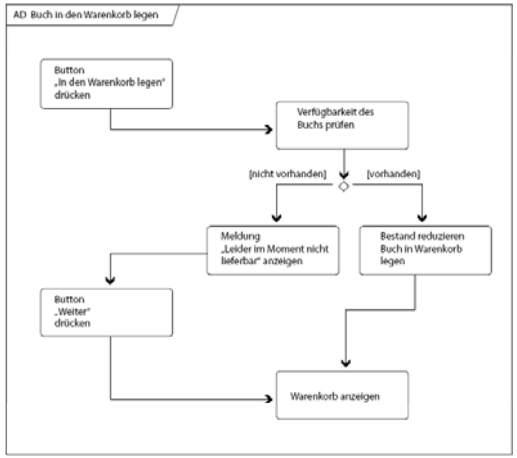
  … beschreibt, wie ein Anwender eines seiner Ziele durch das Verwenden des Systems erreicht

Anwendungsfälle

Anwendungsfall	UC123
Name	Artikel in den Warenkorb legen
Ziel	Der Benutzer entscheidet sich, ein Buch seiner Bestellung hinzuzufügen.
Akteur	Kunde
Auslöser	Button „In den Warenkorb legen“ auf der Seite Buchdetails
Vorbedingung	keine
Nachbedingung (Erfolg)	Artikel ist im Warenkorb verzeichnet
Nachbedingung (Misserfolg)	Artikel ist nicht im Warenkorb verzeichnet Benutzer ist über den Grund informiert
Standardablauf	1. Der Benutzer klickt auf der Seite „Buchdetails“ den Button „In den Warenkorb legen“
	2. Das System überprüft, ob das Buch noch verfügbar ist und fügt dann dem Warenkorb eine neue Position mit dem Buch und Stückzahl 1 hinzu. Das System reduziert den Bestand um 1.
	3. Der Benutzer kann jetzt auf der Seite „Warenkorb editieren“ das Ergebnis sehen und ggfs. die Stückzahl verändern
Alternativabläufe	Das Buch ist nicht mehr verfügbar …
	Das Buch ist bereits im Warenkorb …

Anwendungsfälle (2)

- Aktivitätendiagramm der UML



Fragen ???