Kurs 1612 "Konzepte imperativer Programmierung"

Klausur am 04.03.2000

Aufgabe 1 (6 + 5 + 3 + 4 = 18 Punkte)

In der Mathematik gibt es die sogenanten Armstrong-Zahlen, die wie folgt definiert werden:

> Eine Armstrong-Zahl ist eine Zahl, bei der die Summe der einzelnen Ziffern. die jeweils mit d. Anzahl der Ziffern der Zahl potenziert sind, gleich der Zahl ist.

z.B.: $153 = 1^3 + 5^3 + 3^3$ (3 Ziffern je hoch 3)

a) Schreiben Sie eine Funktion `countZiffen`, welche die Anzahl der Ziffern einer positiven Zahl zurückliefert (in unserem Beispiel die Zahl 3). Benutzen Sie den im Programmrahmen `ArmstrongZahlen` (siehe Seite 3) vorgegebenen Funktionskopf.

b) Schreiben Sie eine Funktion potenziere. Der Sonderfall des undefinierten Wertes 0^0 muss nicht berücksichtigt werden. Die Funktion `potenziere` soll die n-te Potenz einer Zahl berechnen. Benutzen und ergänzen Sie den vorgegebenen Funktionskopf (siehe Seite 3).

c) Die Funktion `isArmstrong` (siehe unten) gibt true zurück wenn der übergebene Parameter eine Armstrong-Zahl ist, sonst false. In dieser Funktion werden die Funktionen `countZiffen` und `potenziere` benutzt.
Bestimmen Sie, an welcher der Stellen (1) - (4) in der Funktion `isArmstrong` die Funktion `potenziere` sinnvoll angewandt wird und geben Sie die komplette Programmzeile an.

d) Schreiben Sie ein Programm, das zwei positive ganze Zahlen x und y ($x > 0$, $y <$ MAXINT, $x < y$) einliest und dann alle Armstrong-Zahlen mit $x \leq a \leq y$ ausgibt.
Hinweis: Auch wenn Sie die Teilaufgaben a) - c) nicht gelöst haben, können Sie die Funktionen voraussetzen. Nutzen Sie dazu den vorgegebenen Programmrahmen.

```
function isArmstrong (inZahl : tNatZahl) : boolean;
{prüft ob inZahl eine Armstrong-Zahl ist}
  var
  i,
  j,
  ggfArmstrong,
  AnzDerStellen
  ZwischErg,
  EndErg,
  Letzteziffer  :  tNatZahl;

begin
  ggfArmstrong := inZahl;
  AnzDerStellen := countZiff(inZahl);
  EndErg := 0;
```

Kurs 1612 “Konzepte imperativer Programmierung”

Klausur am 04.03.2000

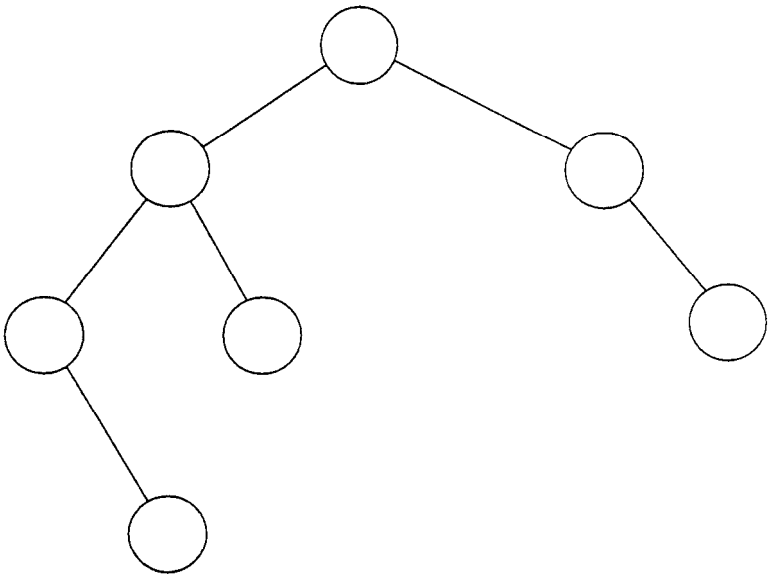
Name: _____ Matrikelnummer: _____

```
    for i := 1 to AnzDerStellen do
    begin
      (1)
      Letzteziffer := ggfArmstrong mod 10;
      (2)
      ggfArmstrong := ggfArmstrong div 10;
      (3)
      EndErg := EndErg + ZwischErg
      (4)
    end;

    if EndErg = inZahl then
      isArmstrong := true
    else
      isArmstrong := false;
  end;
```

```
program ArmstrongZahlen (input, output);

  type
    tNatZahl = 0..maxint;

  var
      ...
  function countZiff (inZahl : tNatZahl) : tNatZahl;
  {gibt die Stellenanzahl der inZahl}
      ...

  function potenziere ( ??? ): tNatZahl;
  {berechnet die n-te Potenz einer Zahl}
      ...

  function isArmstrong (inZahl : tNatZahl) : boolean;
  {prüft ob inZahl eine Armstrong-Zahl ist}
      ...
```

Aufgabe 2 (10 + 4 = 14 Punkte)

Gegeben sei ein (nicht sortierter) binärer Baum und ein Zeiger (inRefWurzel) auf die Wurzel des Baumes. Die Funktion `countLeaf` soll die Anzahl der Blätter in dem Baum bestimmen. Wird der Funktion `countLeaf` ein Zeiger auf die Wurzel des Baumes der folgenden Abbildung übergeben, so liefert sie als Ergebnis den Wert 3. Im Fall eines leeren Baumes wird der Wert 0 zurückgegeben.



a) Ihre Aufgabe ist es, die Funktion countleaf zu implernentieren. Gehen Sie dabei von den unten gegebenen Typdefinitionen und der Funktionsdeklaration aus. Wählen Sie eine rekursive Lösung.

b) Zeichnen Sie für die **Menge** (36, 11, 18, 3, 37, 25, 8, 21) einen degenerierten und einen vollständigen Suchbaum.

```
type
  tNatZahl = 0..maxint;
  tRefBinBaum = ^tBinBaum;
  tBinBaum = record
                info  : tNatZahl;
                links,
                rechts : tRefBinBaum
                     end;

       function countLeaf(??): ??;
       { ?? }
         ...
```

Aufgabe 3 (9 + 9 = 18 Punkte)

a) Schreiben Sie eine PASCAL-Prozedur `Zahlenliste`, welche eine nicht-leere Folge von integer-zahlen mit **`readln`** einliest und in einer linearen Liste abspeichert. Die Reihenfolge der Zahlen in der Liste soll der Reihenfolge der Eingabe entsprechen, d.h. es muss jeweils am Ende der Liste angefügt werden. Die Eingabe einer Null, die nicht in der Liste abgespeichert wird, schließt die Eingabe ab. Benutzen Sie folgende Typdefinitionen und den Prozedurkopf von `Zahlenliste`:

```
type
tRefListe = ^tListe;
tListe = record
           info : integer;
           next : tRefListe
         end;

procedure Zahlenliste (var outliste : tRefListe);
{ liest integer-Zahlen von der Tastatur ein und speichert
  sie in der Reihenfolge der Eingabe in der linearen Liste
  outliste; Null beendet die Eingabe }
```

b) Schreiben Sie nun ein Programm `Zahleninversion`, das unter Benutzung der Prozedur `Zahlenliste` eine Liste mit Zahlen aufbaut und dann die Zahlen der Liste von hinten nach vorne ausgibt. Verwenden Sie für die Ausgabe die unit `Stapel`, deren Interface unten angegeben ist, wie folgt: Zuerst sind die in der Liste gespeicherten Zahlen auf dem Stapel abzulegen, dann werden die Zahlen vom Stapel geholt und ausgegeben. (Haben Sie Teil a) nicht bearbeitet, können Sie die Prozedur Zahlenliste als entsprechend vereinbart annehmen.)

```
unit Stapel;
{ ein Stapel zur Verwaltung von integer-Zahlen }

interface

  function isEmpty : boolean;
  { liefert true, wenn der Stapel leer ist, sonst false }

  procedure top (var outElem : integer);
  { liefert das oberste Stapelelement }

  procedure push (inElem : integer);
  { stapelt ein neues Element }

        procedure pop;
        { entfernt das oberste Stapelelement }
      implementation

      ... { hier nicht relevant }

      end. { Unit Stapel }
```

Kurs 1612 “Konzepte imperativer Programmierung”

Klausur am 04.03.2000

Aufgabe 4 (6 Punkte)

Gegeben sei die BNF-Grammatik G = (N, T, P, S) mit

N = {<sta <A>, <B>, <C>},
T = {a, b . ,,
P = <start> ::= <A> <B> c <B> <C> <A>,
<A> ::= a <A> | c,
<B> ::= b <B> | b,
<C> ::= <A> <C> | c } und
S = <start>

Genau eins der folgenden Worte gehört zu der von der BNF-Grammatik G erzeugten Sprache L(G). Bestimmen Sie das Wort und zeigen Sie die Ableitung.

- ❑ aaacbcbbaac
- ❑ cbbbcbcca
- ❑ acbcbccac

Aufgabe 5 (9 + 8 = 17 Punkte)

Gegeben sei das folgende Programm Quadrat, das die quadratische Funktion

$y = \frac{1}{2} x^2 + \frac{1}{2} x + 1$ $(x > 0)$ implementiert.

```
program Quadrat ;
const
x = ?;
var
i,
Y : integer;

{P ≡ x> 0}

begin
  i := 1;
  Y := 2;
  while i < x do
  begin
    i := i + 1;
    Y := y + i;
  end
end.
{Q ≡ y= 1/2 x^2 + 1/2 x + 1}
```

a) Zeigen Sie, daß

$$INV \equiv 2y = i^2 + i + 2$$

eine Invariante der while-Schleife des Programms `Quadrat` ist. Geben Sie die verwendeten Hoare-Regeln explizit an (siehe Anhang).

b) Genau eines der vier angegebenen Prädikate `A, B, C` oder `D` ist eine geeignete Invarinate `INV '` der while-Schleife.

$$A \equiv INV \wedge i < x$$

$$B \equiv INV \wedge i \leq x$$
$$C \equiv INV \wedge i = x$$
$$D \equiv INV \wedge i > x$$

Wählen Sie das richtige Prädikat aus und beweisen Sie die partielle Korrektheit des Programms `Quadrat.` Dabei können Sie die Invarianteneigenschaften von INV' als bewiesen voraussetzen. Geben Sie die benutzten Hoare-Regeln explizit an.

Aufgabe 6 (10 Punkte)

Zeigen Sie, dass das Programm „Quadrat“ aus Aufgabe 5 terminiert, indem Sie eine Funktion τ angeben und beweisen, dass τ eine Terminierungsfunktion der while-Schleife ist.

(Beachten Sie, dass eine Invariante, mit der sich die Terminierung einer Schleife beweisen lässt, oft nicht identisch mit einer geeigneten Invariante ist, mit der sich die partielle Korrektheit zeigen lässt.)

Aufgabe 7 (6 + 3 + 5 + 3 = 17 Punkte)

Die unten aufgeführte PASCAL-Funktion MaxZif f erF ermittelt die größte vorkommende Ziffer in der Dezimaldarstellung einer nicht-negativen ganzen Zahl.

```
type
tNatZahl = 0..maxint;
tNullBisNeun = 0..9;

  function MaxZifferF (inZahl : tNatZahl) : tNullBisNeun;
  { ermittelt die groesste Ziffer in der Dezimaldarstellung von
  inZahl }

  var
  Maxi,
  Ziffer : tNullBisNeun
  Rest : tNatZahl;

1 begin
2   Rest := inZahl;
3   Maxi := 0;
4   while Rest > 0 do
5   begin
6     Ziffer := Rest mod 10;
7     if Ziffer > Maxi then
8       Maxi := Ziffer;
9     Rest := inZahl div 10
10  end; { while }
11  MaxZifferF := Maxi
12end; { MaxZifferF }
```

a) Erstellen Sie für die Funktion MaxZ i f f erF den kompaktifizierten Kontrollflußgraphen.

b) Ermitteln Sie Testfälle für einen kontrollflußorientierten Test der Funktion MaxZif f erF, die eine 100%ige *Zweigüberdeckung* sicherstellen. Bestimmen Sie zunächst die Pfade im Kontrollflußgraphen und leiten Sie daraus die Testfälle ab. Geben Sie für jeden Testfall ein Testdatum an.

c) Ermitteln Sie nun Testfälle für den *boundary-interior Pfadtest* der Funktion MaxZif f erF. Bestimmen Sie zunächst die hierfür noch fehlenden Pfade im Kontrollflußgraphen und geben Sie für jeden Pfad ein Testdatum an. (Beachten Sie die if-Anweisung.)

d) Mit einem der Tests wird ein Programmfehler entdeckt. Geben Sie ein Testdatum an, dessen Eingabe zur Aufdeckung des Fehlers führt, und schildern Sie die Auswirkung des Fehlers! Wie ist die fehlerhafte Anweisung in der Funktion MaxZi f f erF abzuändern, damit die Funktion korrekt arbeitet?