Lösung: Allgemeine Informatik II

SS 2000

18. Juli 2000

Name ______________________

Vorname ______________________

Matr.Nr. ______________________

Studiengang ______________________

Aufgabe		Punkte	Bewertung
1		8	
2		8	
3		8	
4		8	
5	a)	8	
	b)	8	
6	a)	2	
	b)	4	
	c)	6	
7	a)	3	
	b)	2	
	c)	5	
Summe		**70**	

☞ Bearbeitungszeit: 120 Min.

Viel Erfolg!

Aufgabe 1 (8 Punkte)

Argumentverarbeitung:

Schreiben Sie eine Prozedur **MyUsage**, die als Argument den üblichen Usage-String sowie den Exit-Code erhält, ersteren zusammen mit dem Kommandonamen auf die Diagnoseausgabe ausgibt und mit dem zweiten Argument als Exit-Status terminiert.

Die Funktionen *Init* und *Usage* aus dem Modul UnixArguments dürfen hier **nicht** verwendet werden!

Lösung:

```
PROCEDURE MyUsage(string: ARRAY OF CHAR; code: INTEGER);
  VAR cmd: ARRAY 20 OF CHAR;
BEGIN
  Write.StringS(Streams.stderr, "Usage: ");
  AV.GetCommandName(cmd);
  Write.StringS(Streams.stderr, cmd);
  Write.CharS(Streams.stderr," ");
  Write.StringS(Streams.stderr, string);
  Write.LnS(Streams.stderr);
  Process.Exit(code);
END MyUsage;
```

Aufgabe 2 (8 Punkte)

Schreiben Sie eine Prozedur **Sum**, die als Argument einen String, bestehend aus durch Leerraum getrennten (mehrstelligen) Zahlen, erhält und als Rückgabewert die Summe dieser Zahlen liefert.

Hinweise:

1) Das Modul Strings enthält die Funktion
PROCEDURE Open(VAR stream: Streams.Stream; VAR string: ARRAY OF CHAR);
2) In der Datenstruktur *Streams.Stream* gibt es eine Komponente *count* (also für **s** vom Typ *Streams.Stream* ist es **s.count**), die für die letzte Leseoperation auf 0 gesetzt wird, wenn nicht gelesen wurde, die ansonsten auf einen Wert > 0 gesetzt wird.
3) Sie können davon ausgehen, dass die Module *Streams, Strings, Read* importiert sind.

Lösung:

```
PROCEDURE Sum(str: ARRAY OF CHAR):INTEGER;
   VAR summe, i: INTEGER;
       s: Streams.Stream;
BEGIN
   summe := 0;
   Strings.Open(s,str);
   Read.IntS(s,i);
   WHILE s.count > 0 DO
     summe := summe + i;
     Read.IntS(s,i);
   END;
   RETURN summe;
END Sum;
```


Aufgabe 3 (8 Punkte)

File-I/O:

Schreiben Sie ein Oberon-Programm **MyHead**, das die ersten Zeilen einer Datei ausgibt (Nachbildung des Kommandos *head*).

Es soll wie folgt aufgerufen werden können.‘:

```
MyHead [ -number ] file
```

Sollte keine Zahl *number* angegeben sein, so ist die Voreinstellung **10** zu nehmen!

Hinweis: Alles, was Sie hierzu brauchen, ist in den Beispielen im Skript enthalten (Frage ist nur, wo?)!

```
MODULE MyHead;
  IMPORT Strings, ASCII, UnixFiles, Streams, Read, Write, AV := UnixArguments;
  VAR
     ch: CHAR;
     stream: Streams.Stream;
     number: INTEGER;
     line, file: ARRAY 256 OF CHAR;
BEGIN
   number := 10;
   IF AV.GetFlag(ch) THEN
      AV.UngetOpt;
      AV.Fetch(stream);
      Read.IntS(stream,number);
      IF Streams.Close(stream) THEN END;
   END;
   IF AV.GetArg(file) & UnixFiles.Open(stream, file, 0, 1, NIL) THEN
     Read.LineS(stream, line);
     WHILE ~stream.eof & (number > 0) DO
       DEC(number);
       Write.Line(line);
       Read.LineS(stream, line);
     END;
     IF Streams.Close(stream) THEN END;
   END;

END MyHead.
```

Aufgabe 4 (8 Punkte)

Rekursion

Gegeben sei folgendes Programm:

```
MODULE Rekursi;
  IMPORT Write;
  VAR x,y: INTEGER;

  PROCEDURE Rek(i: INTEGER; VAR j: INTEGER): INTEGER;
  BEGIN
     Write.String("i = "); Write.Int(i,0); Write.Ln;
     Write.String("j = "); Write.Int(j,0); Write.Ln;
     Write.String("----------"); Write.Ln;
     IF i > j THEN
     RETURN Rek(i-j,j)
     ELSIF i < j THEN
     j := j -i;
     RETURN Rek(i,j)
     ELSE
     RETURN i
     END;
  END Rek;
BEGIN

  x := 28; y := 16;
  x := Rek(x,y);
  Write.String("----------"); Write.Ln;
  Write.String("x = "); Write.Int(x,0); Write.Ln;
  Write.String("y = "); Write.Int(y,0); Write.Ln;
END Rekursi.
```

Gegeben Sie an, was dieses Programm als Ausgabe produziert!

Ergebnis:

```
i = 28
j = 16
----------
i = 12
j = 16
----------
i = 12
j = 4
----------
i = 8
j = 4
----------
i = 4
j = 4
----------
----------
x = 4
y = 4
```


Aufgabe 5 (16 Punkte)

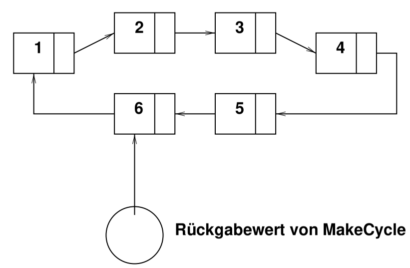
Listen

Ene mene muh - und weg bist du

N Personen stellen sich im Kreis auf. Beginnend bei einer Person scheidet die um M Positionen weiter stehende Person aus (z.B. im Uhrzeigersinn, was hier aber unwichtig ist, es sollte nur immer die gleiche Orientierung sein). Nach dem Ausscheiden wird die entstehende Lücke wieder geschlossen. Das geht so lange, bis nur mehr eine Person im Spiel ist. Die Frage ist nun, welche Person als letztes übrig bleibt.

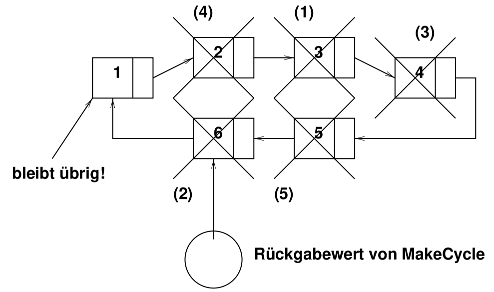
Dazu werden die Personen von eins an nummeriert. Der Kreis wird über eine **zyklische Liste** implementiert:

N = 6;



Beginnend bei der zuletzt eingefügten Person (höchste Nummer) wird nun zyklische jede M-te Person aus der Liste entfernt, es bleibt am Schluss eine Person übrig:

N = 6; M=3



Typvereinbarungen:

```
TYPE
   Node = POINTER TO NodeRec;
   NodeRec = RECORD
                key: INTEGER; (*Nummer der Person*)
                next: Node;
             END;
```

(*a*) 8 Punkte

Implementieren Sie eine Prozedur *MakeCircle*, die als Argument die Anzahl N der Personen erhält und einen Zeiger auf die letzte eingefügte Person zurückliefert (die Personen sollen aufsteigend nummeriert sein und die zuletzt eingefügte Person trägt die höchste Nummer).

Lösung:

```
PROCEDURE MakeCycle(n: INTEGER): Node;
    VAR i: INTEGER; p, list: Node;
BEGIN
   IF (n<1) THEN RETURN NIL; END;
   i := 1; NEW(p); list := p; p.key := i;
   INC(i);
   WHILE (i <= n ) DO
     NEW(p.next);
     p := p.next; p.key := i; INC(i);
   END;
   p.next := list;
   RETURN p
END MakeCycle;
```


(*b*) 8 Punkte

Implementieren Sie die Prozedur *EneMeneMuh*, die als Argument einen Zeiger auf den "Anfang" des Kreises (geliefert von MakeCircle) sowie eine Zahl *m* erhält und folgendes realisiert: die *m*-te Person nach dem Anfang wird gelöscht (aus der Liste entfernt und nicht nur markiert!) und der Zeiger bleibt auf der Person **vor** der gelöschten stehen, danach wird die nächste *m*-te Person gelöscht usw., bis schließlich nur noch eine Person übrig ist - deren Nummer soll als Ergebnis der Prozedur geliefert werden.

Lösung:

```
PROCEDURE EneMeneMuh(p: Node; m: INTEGER): INTEGER;
    VAR i: INTEGER; list: Node;
BEGIN
   ASSERT(p#NIL);
   WHILE ( p # p.next ) DO
     i := 1;
     WHILE (i < m) DO
       p := p.next; INC(i);
     END;
     Write.Int(p.next.key, 0); Write.Ln;
     p.next := p.next.next;
   END;
   RETURN p.key
 END EneMeneMuh;
```

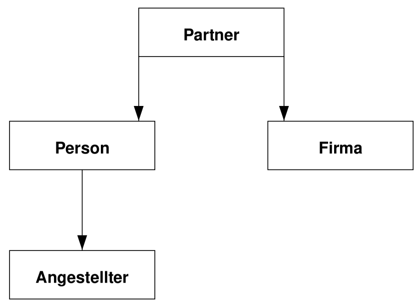
Aufgabe 6 (12 Punkte)

Typerweiterung

Gegeben seien folgende Typdefinitionen:

```
TYPE
   Partner         = POINTER TO PartnerRec;
   PartnerRec      = RECORD
                        name, adresse: ARRAY 80 OF CHAR
                     END;
   Person          = POINTER TO PersonRec;
   PersonRec       = RECORD (PartnerRec)
                        vorname: ARRAY 20 OF CHAR
                     END;
   Angestellter    = POINTER TO AngestelltenRec;
   AngestelltenRec = RECORD (PersonRec)
                        abteilung: ARRAY 16 OF CHAR
                     END;
   Firma           = POINTER TO FirmenRec;
   FirmenRec       = RECORD (PartnerRec)
                        chef: Angestellter
                     END;
```

(*a*) 2 Punkte
Zeichnen Sie ein Diagramm, das die Typ-Hierarchie widerspiegelt.



(*b*) 4 Punkte
Bitte kreuzen Sie jeweils das Zutreffende an:

Aussage	Richtig	Falsch
Angestellter ist eine Erweiterung von *Partner*	X	
Firma ist eine Erweiterung von *Person*		X
Person ist eine direkte Erweiterung von *Partner*	X	
Partner ist eine Erweiterung von *Firma*		X

Hinzu kommen folgende Deklarationen:

```
VAR partner: Partner;
    person: Person;
    angestellter: Angestellter;
    firma: Firma;

PROCEDURE Assign(VAR ptr: Partner; partner: Partner);
BEGIN
   ptr := partner;
END Assign;
```

(*c*) 6 Punkte
Geben Sie bei folgenden Teilaufgaben jeweils an, ob der jeweilige Programmtext A) vom Compiler akzeptiert wird und ob er B) zur Laufzeit fehlerfrei durchläuft. Begründen Sie Ihre Angaben jeweils kurz durch die Angaben der statischen und dynamischen Typen von *ptr* und *partner* innerhalb von *Assign*.

(*c1*) `NEW(partner); NEW(angestellter); Assign(partner, angestellter);`

A) Compiler akzeptiert B) kein Fehler zur Laufzeit

`ptr` ist statisch wie dynamisch vom Typ `Partner` (keine Projektion!), `angestellter` ist vom Typ `Angestellter`, dynamisch vom Typ `Partner` (Projektion!) -

(*c2*) `NEW(firma); NEW(person); Assign(firma, person);`

A) Compiler akzeptiert B) Fehler zur Laufzeit

`ptr` ist dynamisch vom Typ `Firma`, `person` ist statisch vom Typ `Person`, dynamisch vom Typ `Partner` (Projektion)

(*c3*) `NEW(angestellter); NEW(person); Assign(angestellter, person);`

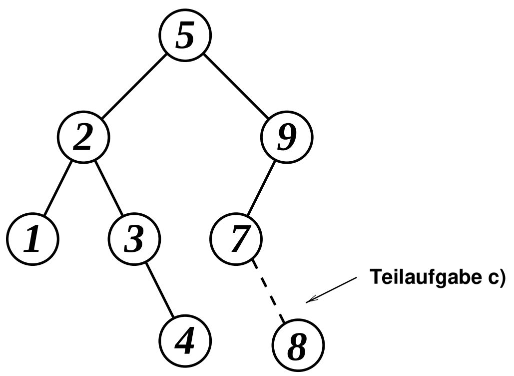
A) Compiler akzeptiert B) Fehler zur Laufzeit

`ptr` ist dynamisch vom Typ `Angestellter`, `person` ist statisch vom Typ `Person`, dynamisch vom Typ `Partner`

Aufgabe 7 (10 Punkte)

Binärbäume

Gegeben sei folgender binärer Baum, bei dem ganze Zahlen als Schlüssel dienen:



(*a*) 3 Punkte

Bitte kreuzen Sie das jeweils zutreffende an:

Aussage	Richtig	Falsch
Der Baum ist bezüglich der Höhe ausgeglichen.	X	
Der Baum ist nach dem Gewicht ausgeglichen.		X
Der Baum ist nach Inorder sortiert.	X	

(*b*) 2 Punkte

Zeichnen Sie auf, wie der vorgegebene Baum aussehen würde, wenn ein Element mit dem Schlüssel 8 hinzukommen würde. (Dabei soll die Ordnung im Baum erhalten werden. Er muß aber nicht nach Höhe oder Gewicht ausgeglichen bleiben, falls er das vorher war!)

siehe oben

(*c*) 5 Punkte

Geben Sie die Schlüssel des Baumes in Preorder an.

5 2 1 3 4 9 7 (8)