Klausur zu Programmieren II

SS 97

5. Juli 1997

Name ______________________________

Vorname ______________________________

Matrikel-Nr. ______________________________

Studienfach ______________________________

Aufgabe		**Punkte**	**Bewertung**
1	a)	10	
	b)	10	
	c)	8	
2	a)	14	
	b)	10	
3		8	
4		6	
5	a)	4	
	b)	2	
	c)	4	
6	a)	6	
	b)	6	
7	a)	4	
	b)	8	
Summe		**100**	

Hinweis: Bitte benützen Sie den jeweils vorgesehenen Platz. Sollte dieser nicht ausreichen, verwenden Sie bitte die Rückseite und machen Sie eine eindeutige Markierung!

Aufgabe 1

Gegeben seien folgende Vereinbarungen:

```
TYPE
     List = POINTER TO Element;
     Element = RECORD
               cont: CARDINAL; (*stets positiv*)
               next: List;
            END;
```

(*a*) 10 Punkte
Realisieren Sie folgende Prozedur:

```
PROCEDURE sumRec(p:List):CARDINAL;
(*
 * Bildet die Summe der Listeneintraege rekursiv
 * liefert 0, falls Liste leer
 *)
```

Lösung:

```
PROCEDURE sumRec(p:List):CARDINAL;
(*Bildet die Summe der Listeneintraege rekursiv*)
BEGIN
     IF p<> NIL THEN
          RETURN (p^.cont + sumRec(p^.next));
     ELSE
          RETURN 0;
     END;
END sumRec;
```

(*b*) 10 Punkte
Realisieren Sie folgende Prozedur:

```
PROCEDURE sumRecVar(p:List; VAR summe: CARDINAL):BOOLEAN;
(*Bildet die Summe der Listeneintraege rekursiv
 *liefert FALSE, falls Liste leer, TRUE sonst
 *)
```

Lösung:

```
PROCEDURE sumRecVar(p:List; VAR sum: CARDINAL):BOOLEAN;
(*Bildet die Summe der Listeneintraege rekursiv*)
     VAR s: CARDINAL;
BEGIN
     IF p = NIL THEN
          RETURN FALSE
     ELSE
          sum := p^.cont;
          IF sumRecVar(p^.next,s) THEN
               sum := sum + s;
          END;
     RETURN TRUE
     END;
END sumRecVar;
```

(*c*) 8 Punkte
Realisieren Sie folgende Prozedur:

```
PROCEDURE sumIter(p:List):CARDINAL;
(*
 *Bildet die Summe der Listeneintraege iterativ
 * liefert 0, falls Liste leer
 *)
```

Lösung:

```
PROCEDURE sumIter(p:List):CARDINAL;
(*Bildet die Summe der Listeneintraege iterativ *)
     VAR s:CARDINAL;
BEGIN
     s := 0;
     WHILE p<> NIL DO
          s := s + p^.cont;
          p := p^.next;
     END;
     RETURN s;
END sumIter;
```

Aufgabe 2

Argumentverarbeitung
Es sei folgender Auszug aus dem Manual zum Kommando **cat** gegeben:

```
NAME
     cat - concatenate and display files

SYNOPSIS
     cat [ -nbsuvet ] [file...]

DESCRIPTION
     cat reads each file in sequence and writes it on  the  stan-
     dard output.

     If no input file is given, cat reads from the standard input
     file.
```

Zur Verfügung steht für beide Teilaufgaben folgendes Definitions-Modul:

```
DEFINITION MODULE printModule;
TYPE
     FlagSet = SET OF CHAR;

PROCEDURE printFile( file: ARRAY OF CHAR): BOOLEAN;
(*
 * printFile gibt eine als Parameter uebergebene Datei (Dateinamen) an
 * stdout aus; falls die Datei nicht existiert oder keine regulaere
 * Datei ist oder die Zugriffsrechte nicht stimmen, wird FALSE
 * zurueckgeliefert, ansonsten TRUE
 *)

PROCEDURE printStdIO();
(*
 * printStdIO hat kein Argument und gibt stdin nach stdout aus
 *)

PROCEDURE printFlags(flags: FlagSet);
END printModule.
```

(*a*) 14 Punkte

Es soll nicht das Kommando *cat* realisiert, sondern nur der Teil, der die Argumentverarbeitung erledigt:

- die zulässigen Optionen sollen erkannt und in einer Menge (vom Typ *printModule.FlagSet*) aufgesammelt werden.
- Falls kein “file” angegeben ist, soll die Funktion *printModule.printStdIO* aufgerufen werden.
- die Dateien, deren Namen als Argument übergeben werden, werden von der Funktion *printModule.printFile* ausgegeben.
- Die angegebenen Optionen werden über die Funktion *printModule.printFlags* ausgegeben.

Vorgegebener Rahmen:

```
MODULE main;
FROM printModule IMPORT printFile, printStdIO, printFlags, FlagSet;

FROM StdIO IMPORT FILE, stdout,stdin,stderr;
FROM FtdIO IMPORT FwriteString, FwriteLn;
FROM Arguments IMPORT GetFlag, GetArg, AllArgs, InitArgs, Usage;
```

Lösung:

```
MODULE main;
FROM printModule IMPORT printFile, printStdIO, printFlags, FlagSet;
FROM StdIO IMPORT FILE, stdout,stdin,stderr;
FROM FtdIO IMPORT FwriteString, FwriteLn;
FROM Arguments IMPORT GetFlag, GetArg, AllArgs, InitArgs, Usage;

VAR
     flag: CHAR;
     file: ARRAY [0..255] OF CHAR;
     flagSet: FlagSet;
BEGIN

     InitArgs(" [ -nbsuvet ] [file...] ");
     flagSet := FlagSet{};
     WHILE GetFlag(flag) DO
          CASE flag OF
          | 'n', 'b', 's', 'u', 'v', 'e', 't' : INCL(flagSet,flag);
          ELSE Usage;
          END;
     END;
     printFlags(flagSet);
     IF NOT GetArg(file) THEN
          (* stdin *)
          printStdIO();
     ELSE
          REPEAT
               IF NOT printFile(file) THEN
                    FwriteString(stderr,file);
                    FwriteString(stderr, " : Error!");
                    FwriteLn(stderr);
               END;
          UNTIL NOT GetArg(file);
     END;
END main.
```

(*b*) 10 Punkte

Realisieren Sie jetzt die Prozedur *printModule.printFlags*!

Lösung:

```
PROCEDURE printFlags(flags: FlagSet);
     VAR
          ch: CHAR;
BEGIN
     WriteString("gesetzte Flags: ");
     FOR ch := ’a’ TO ’z’ DO
          IF ch IN flags THEN
               Write(ch);
          END;
     END;
     WriteLn;
END printFlags;
```

Aufgabe 3

8 Punkte
Das Modulkonzept in Modula-2 trennt textuell in DEFINITION und IMPLEMENTATION MODULE (zwei getrennte Dateien).

Welche der folgenden Aussagen sind richtig, welche falsch?

Aussage	richtig	falsch
Um Leistungen eines Moduls nutzen zu können, muß man das IMPLEMENTATION MODULE lesen können	□	X
hidden types sind nur sinnvoll, wenn alle Operationen für diesen Typ bereitgestellt werden	X	□
Im DEFINITION MODULE muß alles importiert werden, was für die Implementierung benötigt wird	□	X
Im IMPLEMENTATION MODULE sind nur die Prozedurrümpfe zu den im DEFINITION MODULE deklarierten Prozedurköpfen enthalten	□	X
Im IMPLEMENTATION MODULE können zwischen BEGIN und END Initialisierungsanweisungen für exportierte Variable stehen	X	□
Deklarationen von Typen, Konstanten und Variablen im DEFINITION MODULE müssen im zugehörigen IMPLEMENTATION MODULE wiederholt werden	□	X
Sobald alle Definitionsmodule und das Hauptmodul realisiert sind, kann mit dem Compiler eine Prüfung der syntaktischen Korrektheit der Schnittstellen erfolgen	X	□
Wenn viele Module von verschiedenen Entwicklern erstellt werden, entstehen in den Definitionsmodulen u.U. Objekte mit gleichem Namen. Auch wenn alle Module eindeutige Namen tragen, lassen sich Namenskonflikte grundsätzlich nicht vermeiden.	□	X

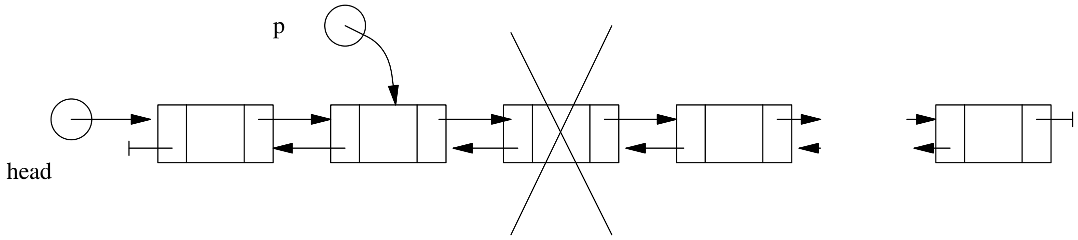
Aufgabe 4

6 Punkte

Liste:

```
TYPE
     ListPointer = POINTER TO Element;
     Element = RECORD
               cont: (*hier unwichtig *);
               next, prior: ListPointer;
            END;
```

Durch folgendes Bild sei eine Liste gegeben, der Zeiger p zeige auf ein Element, dessen (existierender!) Nachfolger gelöscht werden soll:



Geben Sie die Folge von Anweisungen zum Löschen und Entfernen aus dem Speicher an!

Lösung:

```
q := p^.next;
p^.next := q^.next;
IF q^.next <> NIL THEN
     q^.next^.prior : = p;
END;
DISPOSE(q);
```

Aufgabe 5

(*a*) 4 Punkte

Welche Klasse von Problemstellungen lassen sich mit der Technik des "backtracking" bearbeiten? Nennen Sie zwei Beispiele!

sog. Labyrinthprobleme: man kennt Schritte, die in Richtung Lösung führen können; Lösungsverfahren mit dem Prinzip 'trial / error'.
Beispiele: PROLOG-Interpreter, n-Damen-Problem

(*b*) 2 Punkte

Warum wird bei "backtracking" sehr häufig mit rekursiven Prozeduren gearbeitet?

Nach Beendigung der rekursiven Aufrufe kehren die Inkarnationen der Prozeduren auf dem Aufrufweg zurück ('auf der Spur zurück')

(*c*) 4 Punkte

Wie kann bei "backtracking"-Algorithmen vermieden werden, daß bereits untersuchte Lösungswege erneut betrachtet werden? Nennen Sie zwei Alternativen!

durch Markierung der bereits untersuchten Alternativen oder durch Zugrundelegen einer Ordnung über die Alternativen

Aufgabe 6

(*a*) 6 Punkte
Was versteht man unter ‘‘makefiles’’ und was enthalten Sie prinzipiell?

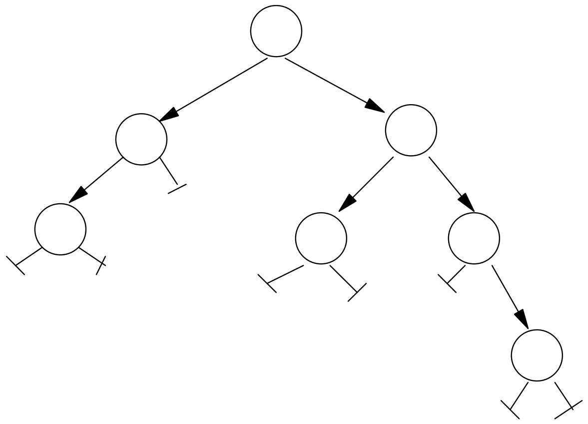
‘makefiles’ enthalten Beschreibungen von Abhängigkeiten von Eingabedateien zu Ergebnisdateien sowie die dazu notwendigerweise auszuführenden Kommando, interpretiert vom Kommando make.

(*b*) 6 Punkte
Was leistet das Kommando ‘‘make’’?

es werden zum einen die Kommandos zur Erzeugung der Ergebnisdateien ausgeführt; falls aufgrund der Zeitstempel der Eingabedateien und der bisherigen Ergebnisdateien auf Basis der beschriebenen Abhängigkeiten keine Notwendigkeit der erneuten Generierung besteht, wird dies auch nicht durchgeführt.

Aufgabe 7

(*a*) 4 Punkte
Gegeben sei der folgende Binärbaum.



Kreuzen Sie bei jeder Aussage an:

	ja	nein
Der Baum ist nach Höhe ausgeglichen	X	□
Der Baum ist nach Gewicht ausgeglichen	□	X

(*b*) 8 Punkte
Was bestimmt folgende Prozedur hinsichtlich oben gegebenem Binärbaum (allgemein bei Binärbäumen und speziell in diesem Fall)?

```
PROCEDURE obscure(wurzel: Baum): CARDINAL;
BEGIN
(* ``links'' / ``rechts'' sind die Zeiger auf den
 * linken / rechten Teilbaum
 *)
  IF wurzel <> NIL THEN
     WITH wurzel^ DO
      IF (links = NIL) & (rechts = NIL) THEN
              RETURN 1
      ELSE
              RETURN obscure(links) + obscure(rechts);
      END
     END;
   ELSE
      RETURN 0
   END;
END obscure;
```

Lösung: Es wird die Anzahl der Blätter ermittelt, hier: 3