Klausur zu Programmieren I

WS 95 / 96

7. Februar 1996

Name _____

Vorname _____

Matrikel-Nr. _____

Aufgabe		Punkte	Bewertung
1	a)	2	
	b)	2	
	c)	2	
	d)	2	
	e)	4	
2		20	
3		10	
4	a)	4	
	b)	4	
5		10	
6	a)	5	
	b)	5	
	c)	15	
7		15	
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

(*a*) Stellen Sie möglichst einfach fest, wieviel Unterkataloge in Ihrem Heimatkatalog sind. Sie können **nicht** davon ausgehen, daß Ihr momentaner Arbeitskatalog der Heimatkatalog ist.
Geben Sie die UNIX-Befehlszeile an.

(*b*) Sie haben ein Modula-2-Programm mit dem *vi* editiert. Nun stellen Sie fest, daß Sie statt der ganzzahligen Division (DIV-Operator) immer die reelle Division (*/*) verwendet haben. Da das Zeichen "/" wirklich nur im Zusammenhang mit ganzzahliger Division vorkommt, können Sie automatisch alle Schrägstriche durch "DIV" ersetzen.
Geben Sie das Editor-Kommando an!

(*c*) Geben Sie ein UNIX-Kommando an, das die Anzahl der Leerzeilen in der Datei *text* ausgibt. Leerzeilen (in einem Text) sind Zeilen, in denen "nichts" oder eine Folge von Blanks / Tab's enthalten sind.

(*d*) Sie haben in Ihrem Heimatkatalog eine ganze Reihe von Modula-2-Quellprogrammen (also viele **.m2**-Dateien). Leider sind die Modul-/Dateinamen sehr wenig aussagekräftig, zum Glück haben Sie aber am Modulanfang aussagekräftige Kommentare. Sie wollen feststellen, welche dieser Dateien das Programm, bei dem Sie im Kommentar "Binaeres Suchverfahren" geschrieben haben, enthält. Geben Sie dazu einen möglichst einfachen Weg an!

(*e*) Sie haben einen richtigen UNIX-Filter mit Modula-2 realisiert, das ausführbare Programm heißt *doit*. Sie wollen es nun so starten, daß die Eingabe aus der Datei *file1* genommen wird, die Ausgabe in die Datei *file2* geht und die Fehlermeldungen in die Datei *err* wandern. Geben Sie dazu zwei mögliche Aufrufsequenzen an!

Aufgabe 2

Schreiben Sie ein Modula-2-Programm namens **ModProgs**, das feststellt, ob eine der als Argument übergebenen Dateien ein Modula-2-Programm enthält. Dies sei daran zu erkennen, das in der ersten nicht-leeren Zeile am Anfang die Buchstabenfolge *MODULE* steht. (Leerzeile enthalte überhaupt kein Zeichen!)
Aufruf:

```
$ ModProgs file { file }
```

Ist ein Dateiargument keine Datei, so soll ein entsprechender Hinweis auf die Diagnose-Ausgabe gegeben werden. Die nach der UNIX-Konvention übliche Behandlung der Standardeingabe — Dateiargument ist ein Minuszeichen — muß nicht berücksichtigt werden. Folgende Prozedur können Sie als gegeben voraussetzen:

```
PROCEDURE FreadLine(f : FILE; VAR line : ARRAY OF CHAR): INTEGER;
  (* liefert -1 bei Fehler oder EOF, ansonsten die Zahl der
   * Zeichen in der gelesenen Zeile ohne 12C,
   * mit angefügtem 0C;
   *)
```

Also kann folgendes verwendet werden:

```
FROM Gegeben IMPORT FreadLine;

CONST MaxLine = 80;
TYPE Buffer = ARRAY [0..MaxLine] OF CHAR;
```

Aufgabe 3

Geben Sie die Ausgabe des folgenden Modula-2 Programms an.
Begründen Sie schrittweise die Zwischenergebnisse.

```
MODULE obscure;

   FROM InOut IMPORT WriteCard, WriteLn;

   VAR x, y : CARDINAL;

   PROCEDURE Obscure(x : CARDINAL; VAR y : CARDINAL): CARDINAL;
   BEGIN
      INC(x); INC(y);
      RETURN (x + y)
   END Obscure;

BEGIN
   x := 6; y := 8;

   IF (Obscure(x, y) < Obscure(y, x)) OR (Obscure(x, y) > Obscure(y, x)) THEN
      x := Obscure(x, y); y := Obscure(y, x)
   ELSE
      x := Obscure(y, x); y := Obscure(x, y)
   END;

   WriteCard(x, 0); WriteLn;
   WriteCard(y, 0); WriteLn
END obscure.
```

Aufgabe 4

Sie implementieren ein Grafikprogramm zur Darstellung und Verwaltung von einfachen geometrischen Objekten (in der Ebene). Zugriffsfunktionen wie Zeichnen, Einfügen, Ändern und Löschen von geometrischen Objekten sollen dabei auf einer allgemeinen Datenstruktur **Objekt** operieren. Jedes Objekt wird durch seine Geometrie beschrieben. Die Geometrie eines Objekts wird durch Angabe von kartesischen oder polaren Koordinaten spezifiziert. Die geometrischen Objekte sind *Punkte*, *Linien*, *Kreise* und zu den Koordinatenachsen parallele *Rechtecke*.

Die Geometrie der einzelnen Objekte ist in folgender Tabelle definiert:

Geometrisches Objekt	**Koordinaten** *(kartesisch oder polar)*
Punkt	Koordinaten des Punktes
Linie	Anfangs- und Endpunkt-Koordinaten
Rechteck	Koordinaten zweier gegenüberligender Punkte
Kreis	Mittelpunkt-Koordinaten und Radius

(*a*) Definieren Sie einen geeigneten Datentyp **Koordinaten** in Modula-2. Abszisse und Ordinate bzw. Radius und Polarwinkel seien reellwertig. Dazu notwendige Hilfsdeklarationen sind mit anzugeben!

(*b*) Definieren Sie unter Verwendung von Teilaufgabe (a) einen geeigneten Datentyp **Objekt** in Modula-2. Dazu notwendige Hilfsdeklarationen sind mit anzugeben!

Aufgabe 5

Eine (fiktive) UNIX-Shell erlaube folgende Kommandozeilen-Syntax:

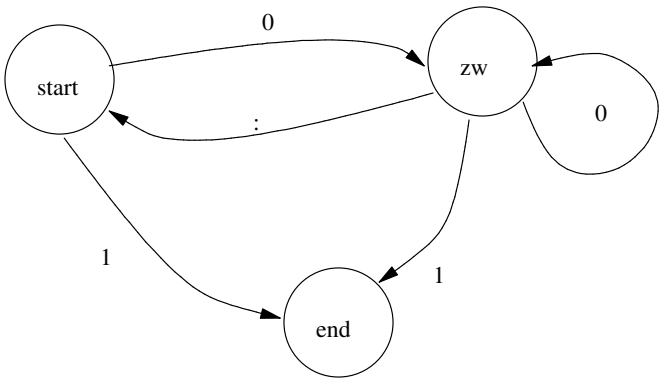
```
CmdLine   ::= [ Cmd [ ('>' File | '>>' File) ] ] '12C'
Cmd ::= Word { Word }
File      ::= Word
```

Die in einfache Hochkommas eingeschlossenen Symbole sind Terminalsymbole. Das Symbol *Word* ist zwar aus einzelnen Zeichen aufgebaut, soll hier aber nicht weiter erläutert werden. Betrachten Sie also *Word* als Terminalsymbol.

Geben Sie einen endlichen Automaten in grafischer Darstellung an (vgl. Aufgabe 6), der diese Sprache erkennt.

Aufgabe 6

Gegeben sei der folgende endliche Automat:



Die Symbole (Beschriftungen von Kanten) sind als Terminalsymbole zu verstehen!

a) Beschreiben Sie die von diesem Automaten definierte Sprache in EBNF.

b) Beschreiben Sie die von diesem Automaten definierte Sprache mit einem regulären Ausdruck.

c) Ergänzen Sie das folgende Programm wie in den Kommentaren angewiesen, so daß es feststellt, ob eine gegebene Zeichenfolge zur durch obigen Automaten definierten Sprache gehört:

```
MODULE parser;
FROM InOut IMPORT Read, Done;
FROM SysExit IMPORT Exit;

PROCEDURE getline(VAR buffer: ARRAY OF CHAR);
(*liefert die zu pruefende Zeichenfolge von stdin in
  den Parameter 'buffer'
---> NICHT zu implementieren, kann als gegeben verwendet werden
*)
END getline;

PROCEDURE parse(buffer: ARRAY OF CHAR): BOOLEAN;
(*
--> diese Prozedur ist zu implementieren!!!
Anm.: die Zeichenfolge, die zu pruefen ist, ist im Parameter 'buffer'
      durch 0C abgeschlossen
*)









END parse;

VAR buffer: ARRAY[0..80] OF CHAR;
BEGIN
     getline(buffer);
     IF parse(buffer) THEN Exit(0) ELSE Exit(1) END;
END parser.
```

Aufgabe 7

Gegeben seien folgende Vereinbarungen:

```
CONST
     Length = 20;
TYPE
     Word = ARRAY [0..Length] OF CHAR;
     Message =
          RECORD
               name: Word;
               sum: CARDINAL;
          END;
```

Schreiben Sie die Prozedur

```
PROCEDURE parSum(VAR m: Message);
```

die für die Komponente *name* eine Prüfzahl berechnet und in der Komponente *sum* ablegt. Die Prüfzahl wird aus den Zeichen der Komponente *name* errechnet (wie, ist freigestellt – es müssen nur alle Zeichen in die Berechnung eingehen) und muß in dem Intervall [0..255] liegen.