Klausur zu Grundlagen der Informatik I
31. März 2004 (WS 03/04)
Prof. Dr. F. Schweiggert
Abt. Angewandte Informationsverarbeitung

Bearbeitungszeit: 120 Minuten

Beispiellösung

Name:	
Vorname:	
Matrikelnummer:	
Studiengang:	

Aufgabe	Punkte	Bewertung
1	**12**	
()	2	
(a)	2	
(b)	2	
(c)	4	
(d)	2	
2	**12**	
()	4	
(a)	4	
(b)	4	
3	**12**	
()	4	
(a)	4	
(b)	4	
4	**6**	

Aufgabe	Punkte	Bewertung
5	**14**	
6	**14**	
()	8	
(a)	6	
7	**16**	
8	**14**	
Summe	**100**	

Bitte benutzen Sie für die Lösungen den freigelassenen Platz nach der jeweiligen Angabe oder die Rückseite unter Angabe der Aufgabenummer. Nennen Sie möglichst alle Annahmen, die Sie gegebenenfalls für die Lösung einer Aufgabe treffen!

Viel Erfolg!

Aufgabe 1 **(12 Punkte)**

Sie möchten in Ihrem Heimatverzeichnis Ordnung schaffen.

(a) 2 Punkte
Im Unterverzeichnis **mails** sind jede Menge Dateien und Unterkataloge, die Sie auf Ihren PC zuhause übertragen wollen. Dazu packen Sie diese komprimiert in eine Datei mit dem Namen **mails.tgz** zusammen. Die Datei *mails.tgz* soll in Ihrem Heimatkatalog abgelegt werden.

Geben Sie die Unix-Befehlszeile an, mit der Sie dies bewerkstelligen. (Ihr Arbeitskatalog ist der Katalog **mails**)!

Lösung:

```
theseus$  tar czf ../mails.tgz *
```

(b) 2 Punkte
Sie befinden sich jetzt in Ihrem Heimatverzeichnis. Hier haben Sie einen Unterkatalog **oberon**. Legen Sie in diesem Unterkatalog einen Unterkatalog **progs** an und verschieben Sie alle Dateien mit Endung **.od** oder mit der Endung **.om** im Heimatverzeichnis in den Unterkatalog *progs*, ohne für diese Aufgabe den Heimatkatalog zu verlassen! Geben Sie die dazu notwendigen Unix-Befehlszeilen an!

Lösung:

```
theseus$ mkdir oberon/progs

theseus$ mv *.od *.om oberon/progs
```

(c) 2 Punkte
Sie haben in Ihrem Heimatkatalog zwei Dateien namens *adressen.alt* und *adressen.neu*. Was ist der Unterschied in der Wirkung der folgenden beiden Kommandozeilen:

```
theseus$ cat adressen.neu > adressen.alt

theseus$ cat  < adressen.neu >> adressen.alt
```

Lösung: Im ersten Fall wird der Inhalt der Datei adressen.alt mit dem Inhalt der Datei adressen.neu überschrieben; im zweiten Fall wird der Inhalt der Datei adressen.neu an den Inhalt der Datei adressen.alt angefügt

(d) 4 Punkte
Sie sind inzwischen in den Unterkatalog *tmp* gewechselt. Sie geben folgendes Kommando ein:

```
theseus$ ls -l
```

und erhalten als Ausgabe

Da die erste Zeile auf eine Datei „ohne“ Namen hinweist, geben Sie das Kommando

```
theseus$ ls | od -bc
```

und erhalten als Ausgabe

Offenbar hat die Datei einen Namen, der nur aus einem Blank (Leerzeichen) besteht. Diese wie auch die Datei mit dem Namen **?** wollen Sie mit einem Kommando löschen. Die anderen Dateien sollen natürlich erhalten bleiben. Geben Sie die dazu notwendige Unix-Befehlszeile an!

Lösung:

```
theseus$ \rm \  \?
```

(e) 2 Punkte
In Ihrem Unterkatalog **tmp** gibt es viele Dateien, deren Name mit einem Punkt beginnt. Geben Sie an, wie Sie diejenigen davon löschen, bei denen nach dem Punkt am Anfang noch genau ein Kleinbuchstabe folgt. Ihr Arbeitskatalog ist der Unterkatalog **tmp**.

Lösung:

```
theseus$ rm .[a-z]
```

Aufgabe 2 **(12 Punkte)**

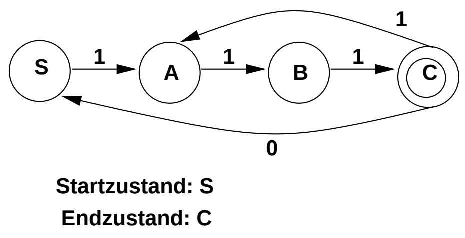
Über dem Vokabular $V = \{0, 1\}$ ist folgende Sprache definiert:

> Alle nichtleeren Zeichenfolgen über V, bei denen die Anzahl der unmittelbar nebeneinanderstehenden Einsen (**1**) ein Vielfaches von 3 ist ($l * 3, l \geq 1$); mehrere solcher Einserfolgen können vorkommen, sind dann aber durch genau eine Null getrennt.
>
> Korrekt sind also 111, 1111110111; nicht korrekt sind 1111 oder 1110

(a) 4 Punkte
Beschreiben Sie diese Sprache durch einen regulären Ausdruck! Als Metasymbole dürfen die im Skript im Zusammenhang mit Unix-Kommandos dargestellten Symbole verwendet werden!

Lösung: `(111)+(0(111)+)*`

(b) 4 Punkte
Beschreiben Sie diese Sprache mit einem grafisch dargestellten endlichen, deterministischen Automaten! Start- und Endzustand (oder Endzustände) sind zu kennzeichnen!
Lösung:



(c) 4 Punkte
Geben Sie eine Grammatik in EBNF an, die dieselbe Sprache beschreibt!
Lösung:

$$
\begin{array}{lll}
S \to 1A & A \to 1B & B \to 1C \\
C \to \epsilon & C \to 1B & C \to 0D \\
D \to 1A & &
\end{array}
$$

Aufgabe 3 **(12 Punkte)**

In der Datei *results* werden die Ergebnisse einer Klausur geführt. Die Einträge haben die Form *Matrikelnummer*:*Punktezahl*:*note* (jeder Eintrag kommt in eine neue Zeile). Für die folgenden Aufgaben habe die Datei *results* den folgenden Inhalt:

```
392244:3:
384433:81:1.3
393322:78:1.7
471133:45:4.0
472211:0:5.0
472233:86:1.0
471122:89:1.0
473311:82:1.3
473344:4:5.0
474411:74:2.0
374422:7:5.0
471144::
```

(a) 4 Punkte
Welche Ausgabe liefert folgendes Kommando?

```
egrep '^.*:$' < results
```

Lösung:

```
392244:3:
471144::
```

(b) 4 Punkte
Welche Ausgabe liefert folgendes Kommando?

```
cat results | egrep '^[^:]*:[^:]?:'
```

Lösung:

```
392244:3:
472211:0:5.0
473344:4:5.0
374422:7:5.0
471144::
```

(c) 4 Punkte
Welche Ausgabe liefert folgendes Kommando? Zusätzlich ist hier zu beschreiben, **was** diese Ausgabe beschreibt!

```
egrep ':1[.][037]$' results | wc -l
```

Lösung:

```
5
```

Alle Klausurergebnisse mit 1.0, 1.3, 1.7

Aufgabe 4 **(6 Punkte)**

Gegeben ist das folgende Programm:

```
MODULE Obscure;
  IMPORT Write;

  VAR u,v,w: INTEGER;

  PROCEDURE Magic(x: INTEGER; VAR y: INTEGER; z: INTEGER);
    VAR tmp: INTEGER;
  BEGIN
    x := y + 1; y := z + 1; z := x + 1;
  END Magic;

BEGIN
  u :=1; v:=2; w:=3;
  Magic(w,v,u);
  Write.Int(u,4); Write.Int(v,4); Write.Int(w,4);Write.Ln;
  Magic(w,u,v);
  Write.Int(u,4); Write.Int(v,4); Write.Int(w,4);Write.Ln;
  Magic(v,u,w);
  Write.Int(u,4); Write.Int(v,4); Write.Int(w,4);Write.Ln;
  Magic(v,w,u);
  Write.Int(u,4); Write.Int(v,4); Write.Int(w,4);Write.Ln;
  Magic(u,v,w);
  Write.Int(u,4); Write.Int(v,4); Write.Int(w,4);Write.Ln;
  Magic(u,w,v);
  Write.Int(u,4); Write.Int(v,4); Write.Int(w,4);Write.Ln;
END Obscure.
```

Geben Sie exakt an, was dieses Programm an die Standardausgabe schreibt!

Lösung:

```
   1   2   3
   3   2   3
   4   2   3
   4   2   5
   4   6   5
   4   6   7
```

Aufgabe 5 **(14 Punkte)**

Palindrome sind Wörter und Sätze, die vor- und rückwärts gelesen identisch sind. Das klassische Beispiel:

Ein Neger mit Gazelle zagt im Regen nie

Schreiben Sie eine Bool'sche Prozedur *IsPalindrom*, die bestimmt, ob ein als Parameter übergebener String ein Palindrom ist oder nicht. Leerzeichen sind nicht enthalten; damit die Wortanfänge sichtbar sind, ist der Anfangsbuchstabe eines jeden Wortes groß. Groß- / Kleinschreibung ist nicht signifikant, d.h. beispielsweise, dass “E” zu “e” passt.

Beispiel für die Verwendung:

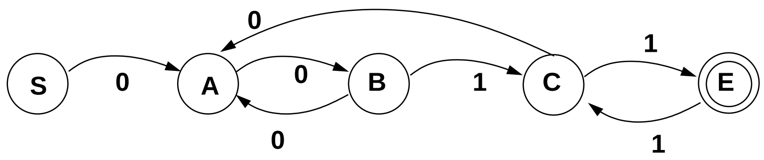
```
IF IsPalindrom("EinNegerMitGazelleZagtImRegenNie") THEN ... END;
```

Lösung:

```
PROCEDURE IsPalindrom (s : ARRAY OF CHAR) : BOOLEAN;
VAR i, j : INTEGER;
BEGIN
  i := 0;
  WHILE (i < LEN (s)) & (s[i] # 0X) DO
    INC (i);
  END;
  DEC (i);
  (* Jetzt steht i auf dem letzten Zeichen oder auf -1 im Falle
     eines leere Strings. Dann wird die Schleife uebersprungen. *)
  j := 0;
  WHILE j < i DO
    IF CAP(s[j]) # CAP(s[i]) THEN RETURN FALSE; END;
    INC (j); DEC (i);
  END;
  RETURN TRUE;
END IsPalindrom;
```

Aufgabe 6 **(14 Punkte)**

Eine Sprache über dem Vokabular $\{0, 1\}$ sei durch folgenden Automaten definiert:



Startzustand ist **S**, Endzustand ist **E**.

(a) 8 Punkte

Schreiben Sie eine Bool'sche Prozedur *Parse*, die als Argument einen String erhält und bestimmt, ob dieser zu der durch diesen Automaten definierten Sprache gehört.

Lösung:

```
PROCEDURE Parse(s: ARRAY OF CHAR):BOOLEAN;
   VAR i: INTEGER;
       state: CHAR;
BEGIN
 state := "S"; i := 0;
 WHILE (i < LEN(s)) & (s[i] # 0X) DO
   CASE state OF
   | "S" : IF s[i] = "0" THEN state := "A" ELSE RETURN FALSE END;
   | "A" : IF s[i] = "0" THEN state := "B" ELSE RETURN FALSE END;
   | "B" : IF s[i] = "0" THEN state := "A" ELSE state := "C" END;
   | "C" : IF s[i] = "0" THEN state := "A" ELSE state := "E" END;
   | "E" : IF s[i] = "1" THEN state := "C" ELSE RETURN FALSE END;
   ELSE RETURN FALSE
   END;
   INC(i);
   END;
   IF state = "E" THEN RETURN TRUE ELSE RETURN FALSE END;
END Parse;
```

(b) 6 Punkte

Auch wenn Sie die Teilaufgabe a) nicht gelöst haben, so können Sie für das folgende diese Prozedur benutzen.

Schreiben Sie ein Oberon-Programm, das über die Standardeingabe zeilenweise Zeichenfolgen erhält und die Zeilennummern der Zeilen ausgibt, die einen korrekten Satz obiger Sprache darstellen.

Lösung:

```
MODULE MyParser;
   IMPORT Read, Write, Streams;
   CONST length = 256;
   VAR z: INTEGER;
       line: ARRAY length OF CHAR;

   PROCEDURE Parse(s: ARRAY OF CHAR):BOOLEAN;
   (* siehe Teilaufgabe a) *)

BEGIN

   Read.Line(line); z := 1;
   WHILE ~ Streams.stdin.eof  DO
      IF Parse(line) THEN
         Write.Int(z, 4); Write.Ln;
      END;
      INC(z); Read.Line(line);
   END;

END MyParser.
```

Aufgabe 7 **(16 Punkte)**

Zur Darstellung sehr großer Dezimalzahlen kann man ein `ARRAY OF CHAR` verwenden; die Einerstelle kommt auf Indexposition 0, die 10-er Stelle auf Indexposition 1, usw.

Sei a ein solches Array und z eine Dezimalziffer (repräsentiert als CHAR), so könnte die Berechnung von $a := a * z$ durch eine Prozedur `Mul` realisiert werden:

```
PROCEDURE Mul(VAR a: ARRAY OF CHAR; z: CHAR);
```

Realisieren Sie diese Prozedur! Sie können davon ausgehen,

- dass der aktuelle Parameter a den üblichen Konventionen (Null-Byte) folgt und
- dass das Ergebnis auf jeden Fall in a dargestellt werden kann.

Lösung:

```
PROCEDURE Mul(VAR a: ARRAY OF CHAR; z: CHAR);
  VAR i, p, n,  carry: INTEGER;

BEGIN
  i := 0; carry := 0;
  n := ORD(z)-ORD("0");
  WHILE (i < LEN(a)) & (a[i] # 0X) DO
     p := (ORD(a[i]) -ORD("0")) * n + carry;
     carry := p DIV 10; p := p MOD 10;
     a[i] := CHR(ORD("0")+p);
     INC(i);
  END;
  IF (a[i] = 0X) & (carry # 0) THEN
     a[i] := CHR(ORD("0")+carry);
     a[i+1] := 0X;
  END;
 END Mul;
```

Aufgabe 8 **(14 Punkte)**

Argumentverarbeitung:

Die Aufrufstruktur des zu schreibenden Oberon-Programms sei wie folgt definiert:

```
progname [-n number] [-o outfile] [-cz] file ...
```

Schreiben Sie ein komplettes Oberon-Programm mit der in der Vorlesung behandelten Argumentverarbeitung, welches bei korrektem Aufruf den Wert aller Optionen und die Namen der angegebenen Dateien ausgibt! Kommandozeile angegebenen Optionen, deren Werte und die Zahl der angegebenen Dateien ausgibt.

Beispiel: Für `$ cmd -n 33 -o a.out -c a "b c"` wird folgende Ausgabe produziert:

```
Options:
number = 33
outfile = a.out
c = TRUE
z = FALSE
FILES:
a
b c
```

Lösung nächste Seite

Lösung:

```
MODULE Arg;
  IMPORT Read, Write, Streams, Args := UnixArguments;
  CONST ArgLength = 256;
  TYPE  ArgName = ARRAY ArgLength OF CHAR;
  VAR   flag: CHAR; number: INTEGER; c,z : BOOLEAN;
        arg, outfile: ArgName; args: Streams.Stream;
  BEGIN
  Args.Init("[-n number] [-o outfile] [-cz] file ...");
  c := FALSE; z := FALSE;
  WHILE Args.GetFlag(flag) DO
     CASE flag OF
     | "n" : Args.Fetch(args);
             Read.IntS(args, number);
             Write.String("number = ");
             Write.Int(number, 4); Write.Ln;
     | "o" : Args.FetchString(outfile);
             Write.String("outfile = ");
             Write.String(outfile); Write.Ln;
     | "c":  c := TRUE;
     | "z":  z := TRUE;
     ELSE
             Args.Usage;
     END;
  END;
  Write.String("c = ");
  IF c THEN Write.String("TRUE") ELSE Write.String("FALSE") END;
  Write.Ln;
  Write.String("z = ");
  IF z THEN Write.String("TRUE") ELSE Write.String("FALSE") END;
  Write.Ln;
  Write.String("FILES: "); Write.Ln;
  WHILE Args.GetArg(arg) DO
        Write.String(arg); Write.Ln;
  END;
END Arg.
```