Klausur zu Grundlagen der Informatik I
17. Februar 2004 (WS 03/04)

Prof. Dr. F. Schweiggert
Abt. Angewandte Informationsverarbeitung

Bearbeitungszeit: 120 Minuten
Beispiellösung

Name:	
Vorname:	
Matrikelnummer:	
Studiengang:	

Aufgabe	Punkte	Bewertung
1	**16**	
()	2	
(a)	2	
(b)	2	
(c)	4	
(d)	2	
(e)	4	
2	**12**	
()	4	
(a)	4	
(b)	4	
3	**12**	
()	4	
(a)	4	
(b)	4	

Aufgabe	Punkte	Bewertung
4	**6**	
5	**12**	
6	**12**	
()	2	
(a)	2	
(b)	8	
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 **(16 Punkte)**

Sie möchten in Ihrem Heimatverzeichnis Ordnung schaffen.

(a) 2 Punkte
Im Unterverzeichnis **schrott** sind jede Menge Dateien und Unterkataloge, die Sie allesamt löschen wollen – das Verzeichnis **schrott** soll also komplett geleert werden. Das Verzeichnis **schrott** als solches soll aber erhalten bleiben! Sie können davon ausgehen, dass die Dateinamen nur aus Buchstaben und Ziffern bestehen!

Geben Sie die Unix-Befehlszeile an, mit der Sie dies bewerkstelligen (Ihr Arbeitskatalog ist Ihr Heimatverzeichnis – Sie dürfen dieses nicht verlassen)!

Lösung:

`theseus$` rm -r schrott/*

(b) 2 Punkte
Sie befinden sich nach wie vor in Ihrem Heimatverzeichnis. Legen Sie einen Unterkatalog **pictures** an und verschieben Sie alle Dateien mit Endung **.jpg** oder mit der Endung **.gif** im Heimatverzeichnis in diesen Unterkatalog, ohne für diese Aufgabe den Heimatkatalog zu verlassen! Geben Sie die dazu notwendigen Unix-Befehlszeilen an!

Lösung:

`theseus$` mkdir pictures

`theseus$` mv *.jpg *.gif pictures/

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

```
theseus$ cat names.new > names.old
theseus$ cat names.new | cat >> names.old
```

Lösung:

Die erste Kommandozeile bewirkt, dass der Inhalt der Datei names.new in die Datei names.old geschrieben wird. Dabei wird die Datei names.old komplett überschrieben.
Bei der zweiten Kommandozeile wird der Inhalt der Datei names.new an die Datei names.old angehängt.

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

```
theseus$ ls -a
```

und erhalten als Ausgabe

```
. .. .arcsin arcsin.od  arcsin.od.old  arcsin.om  arcsin.om-old
makefile
```

Mit der Kommandozeile

```
theseus$ rm *
```

wollen Sie auch hier aufräumen. Danach geben Sie wieder ein:

```
theseus$ ls -a
```

Welche Ausgabe erscheint nun?

Lösung:

```
theseus$ . .. .arcsin
```

(e) 2 Punkte
In Ihrem Heimatkatalog hatten Sie vor langer Zeit einen Unterkatalog **tmp** angelegt, in dem sich viele nicht mehr benötigte Dateien „angesammelt" haben – ausschließlich reguläre Dateien. Deren Namen bestehen allesamt aus einem oder zwei Buchstaben. Geben Sie an, wie Sie diejenigen löschen, deren Name mit einem Großbuchstaben beginnt! Ihr Arbeitskatalog ist der Unterkatalog **tmp**.

Lösung:

```
theseus$ rm [A-Z]*
```

(löscht alle Dateien, die mit einem Großbuchstaben beginnen)

```
theseus$ rm [A-Z] [A-Z][A-Za-z]
```

(löscht explizit alle Dateien mit einem oder zwei Buchstaben, die mit einem Großbuchstaben beginnen)

(f) 4 Punkte
Ein Programm namens **run** werde wie folgt gestartet:

```
theseus$ cat a | run 2>b >c
```

Beschreiben Sie kurz die Bedeutung der einzelnen Angaben!

Lösung:
cat a | : schreibt den Inhalt der Datei a in die Pipe (statt an Stdout)

| run : run wird gestartet und liest aus der Pipe (statt von Stdin)
$2 > b$: falls das Programm Ausgaben an Stderr schreibt, werden diese in die Datei b geschrieben
$> c$ falls das Programm Ausgaben an Stdout schreibt, werden diese in die Datei c geschrieben

Aufgabe 2 **(12 Punkte)**

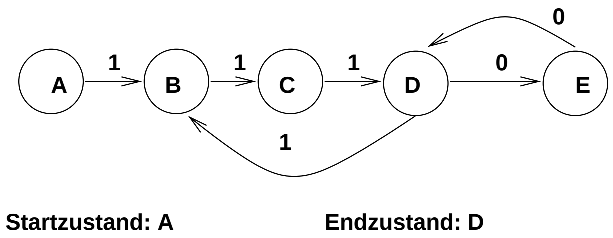
Über dem Vokabular $V = \{0, 1\}$ ist folgende Sprache definiert:

> Alle nichtleeren Zeichenfolgen über V, bei denen die Anzahl der unmittelbar nebeneinanderstehenden Einsen (**1**) ein Vielfaches von 3 ist und die Anzahl der unmittelbar nebeneinanderstehenden Nullen (**0**) ein Vielfaches von 2 ist; die Zeichenfolgen müssen mit Einsen beginnen.

(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(111|00)*
```

(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 reguläre Grammatik in EBNF an, die dieselbe Sprache beschreibt!
Lösung:

A → 1B B → 1C C → 1D
D → 1B D → 0E E → 0D
D → ϵ

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:

```
471122:90:1.0
471133:45:4.0
471144::
472211:0:5.0
472233:86:1.0
472244:3:
473311:82:1.3
473322:78:1.7
473344:4:5.0
474411:74:2.0
574422:9:5.0
474433:81:1.3
```

(a) 4 Punkte
Welche Ausgabe liefert folgendes Kommando?

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

Lösung:

```
472211:0:5.0
472244:3:
473344:4:5.0
574422:9:5.0
```

(b) 4 Punkte
Welche Ausgabe liefert folgendes Kommando?

```
cat results | egrep '^[^:]*:[89].:'
```

Lösung:

```
471122:90:1.0
472233:86:1.0
473311:82:1.3
474433:81:1.3
```

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

```
egrep ':[12]\.0$' results | wc -l
```

Lösung:
`3`

Anzahl der Studenten, die in der Klausur die Noten $1,0$ oder $2,0$ erreicht haben.

Aufgabe 4 **(6 Punkte)**

Gegeben ist das folgende Programm:

```
MODULE Obscure;
  IMPORT Write;

  VAR u,v,w: CHAR;

  PROCEDURE Magic(x: CHAR; VAR y: CHAR; z: CHAR);
    VAR tmp: CHAR;
  BEGIN
    tmp := x; x:=y; y:=z; z:=y
  END Magic;

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

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

Lösung:

```
113
113
313
313
333
333
```

Aufgabe 5 **(12 Punkte)**

Schreiben Sie eine Oberon-Prozedur **Swap**, die als Argument einen String bekommt und darin das erste mit dem letzten Zeichen, das zweite mit dem vorletzten Zeichen, ..., vertauscht.

Beispiel für die Verwendung:

```
   string := "E-Technik";
   Swap(string);
   Write.String(string); Write.Ln;
   (*liefert: "kinhceT-E"*)
```

Lösung:

```
PROCEDURE Swap(VAR s: ARRAY OF CHAR);

 VAR i,j: INTEGER; tmp: CHAR;

 BEGIN
   i := 0;
   WHILE (i < LEN(s)) & (s[i] # 0X) DO
     INC(i);
   END;
   IF i<LEN(s) THEN DEC(i) END;
   j := 0;
   WHILE (j <= i DIV 2) DO
     tmp := s[j];
     s[j] := s[i-j];
     s[i-j] := tmp;
     INC(j);
   END;

END Swap;
```

Aufgabe 6 **(12 Punkte)**

Sie sollen ein Oberon-Programm schreiben, das von der Standardeingabe einen Text liest und darin die Häufigkeiten der verschiedenen Kleinbuchstaben ermittelt.

(a) 2 Punkte
Die Buchstaben „a“ …„z“ sollen zunächst auf das Intervall 0 … 25 abgebildet werden, also „a“ $\rightarrow$ 0, „b“ $\rightarrow$ 1, …
Die Variable ch (vom Typ CHAR) enthalte einen Kleinbuchstaben und der Variablen i ist der korrespondierende Index zuzuweisen. Geben Sie die entsprechende Oberon-Anweisung an:

```
i := ORD(ch)-ORD("a");
```

(b) 2 Punkte
Damit können die Häufigkeiten der Kleinbuchstaben wie folgt gezählt werden: In einem Integer-Array mit 26 Elementen ist der Index die dem Kleinbuchstaben nach Teilaufgabe a) zugeordnete Zahl, der Wert ist die festgestellte Häufigkeit.

Für die Ausgabe muss aus dem Index aber wieder der zughörige Kleinbuchstabe (als *CHAR*) ermittelt werden. Sei also *i* eine Zahl zwischen 0 und 25; geben Sie an, wie daraus der Buchstabe bestimmt werden kann:

```
ch := CHR(ORD("a") + i);
```

(c) 8 Punkte
Das zu erstellende Programm liest den Text von der Standardeingabe, stellt für jedes Zeichen fest, ob es ein Kleinbuchstabe ist oder nicht; wenn ja, so wird der Zähler für diesen Buchstaben um 1 erhöht. Die Ausgabe soll eine Tabelle der Form `<buchstabe> : <haeufigkeit>` sein.

Der Programmrahmen sei wie folgt vorgegeben:

```
MODULE Count;
  IMPORT Write, Read, Streams;
  CONST
     numberOfLetters = 26;
  TYPE
     CountField = ARRAY numberOfLetters OF INTEGER;
  VAR
     ch: CHAR; i: INTEGER;
     countField: CountField;
```

```
BEGIN
  (* Initialisierung: *)
  i := 0;
  WHILE i < numberOfLetters DO
    countField[i] := 0;
    INC(i);
  END;

  Read.Char(ch);
  WHILE ~ Streams.stdin.eof DO
     IF (ORD(ch) >= ORD("a") ) & (ORD(ch) <= ORD("z")) THEN
        i := ORD(ch)-ORD("a");
        INC(countField[i]);
     END;
     Read.Char(ch);
  END;

  i := 0;
  WHILE i < Number DO
    Write.Char(CHR(ORD("a")+i)); Write.String(" : ");
    Write.Int(countField[i],0); Write.Ln;
    INC(i);
  END;

END Count.
```

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. Seien a, b zwei solche Arrays, so könnte die Berechnung von $a := a + b$ durch eine Prozedur `Sum` realisiert werden:

```
PROCEDURE Sum(VAR a: ARRAY OF CHAR; b: ARRAY OF CHAR);
```

Realisieren Sie diese Prozedur! Sie können davon ausgehen,

- dass die aktuellen Parameter den üblichen Konventionen (Null-Byte) folgen,
- dass das Ergebnis auf jeden Fall in a dargestellt werden kann und
- dass $a \geq b$ ist!

Lösung:

```
PROCEDURE Sum(VAR a: ARRAY OF CHAR; b: ARRAY OF CHAR);
  (* a >= b *)
  VAR i, sum, carry: INTEGER;
BEGIN
  i := 0; carry := 0;
  WHILE (i < LEN(b)) & (b[i] # 0X) DO
     sum := ORD(a[i]) + ORD(b[i]) - 2* ORD("0") + carry;
     carry := sum DIV 10; sum := sum MOD 10;
     a[i] := CHR(ORD("0")+sum);
     INC(i);
  END;
  (* b[i] = 0X *);
  IF (a[i] = 0X) & (carry # 0) THEN
       a[i] := CHR(ORD("0")+carry); a[i+1] := 0X;
       RETURN;
  ELSIF carry = 0 THEN
       RETURN
  ELSE
    WHILE (i < LEN(a)) & (a[i] # 0X) DO
       sum := ORD(a[i])-ORD("0")+carry;
       carry := sum DIV 10; sum := sum MOD 10;
       a[i] := CHR(ORD("0")+sum);
       IF carry = 0 THEN RETURN ELSE INC(i); END;
    END
  END
 END Sum;
```

Aufgabe 8 **(14 Punkte)**

Argumentverarbeitung:

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

`progname [-n number] [-xz] file ...`

Die Voreinstellung (*default*) für *number* ist **1**. Schreiben Sie ein komplettes Oberon-Programm mit der behandelten Argumentverarbeitung, welches bei korrektem Aufruf die Zahl der auf der Kommandozeile angegebenen Optionen, deren Werte und die Zahl der angegebenen Dateien ausgibt.

Beispiel: Für `$ cmd -n 33 -z d1 d2` wird folgende Ausgabe produziert:

```
Number of Options: <2>
Options set: <n> with 33, <z>
Number of Files: <2>
```

```
MODULE Arg;

  IMPORT Print, Read, Streams, Args := UnixArguments;

  CONST
    ArgLength = 256;
  TYPE
    ArgName = ARRAY ArgLength OF CHAR;
  VAR
    flag: CHAR;
    flagcount, flagnumber, argcount, counter: INTEGER;
    arg, flagarray: ArgName;
    args: Streams.Stream;

  PROCEDURE PrintFlags(ch : CHAR);
    BEGIN
      Print.F1("<%c>", ch);
      IF (ch = "n") THEN
        Print.F1(" with %d", flagnumber);
      END;
  END PrintFlags;


  BEGIN
    Args.Init("[-n number] [-xz] file ...");
```

```
    flagcount := 0;
    flagnumber := 1;
    WHILE Args.GetFlag(flag) DO
      CASE flag OF
        | "n" :
          flagarray[flagcount] := flag;
          Args.Fetch(args);
          Read.IntS(args, flagnumber);
          INC(flagcount);
        | "x" :
          flagarray[flagcount] := flag;
          INC(flagcount);
        | "z" :
          flagarray[flagcount] := flag;
          INC(flagcount);
        ELSE
          Args.Usage;
      END;
    END;

    WHILE Args.GetArg(arg) DO
      INC(argcount);
    END;

    counter := 0;
    Print.F1("Number of Options: <%d>\n", flagcount);
    IF (flagcount > 0) THEN
      Print.F("Options set: ");
      WHILE (counter < flagcount-1) DO
        PrintFlags(flagarray[counter]);
        Print.F(", ");
        INC(counter);
      END;
      PrintFlags(flagarray[counter]);
      Print.F("\n");
    END;
    Print.F1("Number of Files: <%d>\n", argcount);

END Arg.
```