Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 01.04.2000

Aufgabe 1 (6 + 6 = 12 Punkte)

Die Prozedur `Vielfache` ermittelt alle Vielfachen der als Prozedurparameter übergebenen Zahl `inZahl` im Bereich zwischen 1 und 1000 und gibt diese mit Hilfe der `wri` teln-Prozedur aus. Sie können davon ausgehen, dass gilt: $1 \leq$ `inZahl` ≤ 1000.
Beim Aufruf von `Vielfache` mit dem Parameter 19 0 erhält man beispielsweise die Ausgabe

```
Die Vielfachen von 190 sind:
190
380
570
760
950
```

Betrachten Sie dazu die folgenden Typdeklarationen und die Prozedur `Vielfache`:

```
const
MINZAHL = 1;
MAXZAHL = 1000;

type
tzahlenbereich = MINZAHL..MAXZAHL;

procedure Vielfache (inZahl : tzahlenbereich);
{ gibt alle Zahlen im Bereich tzahlenbereich aus, die Viel-
  faches von inZahl sind }

  var
  Vielfaches : tzahlenbereich;

begin
  writeln ('Die Vielfachen von ', inZahl, 'sind:');
  Vielfaches := inZahl;
  { inZahl ist Vielfaches von sich selbst }
  writeln (Vielfaches);
  while Vielfaches + inZahl <= MAXZAHL do
  begin
    Vielfaches := Vielfaches + inZahl;
    writeln (Vielfaches)
  end { while }
end; { Vielfache }
```

Bearbeiten Sie nun die Teilaufgaben a) und b). Setzen Sie für beide Teilaufgaben den folgenden Prozedurrahmen voraus.

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 0 1.04.2000

Name: _____ Matrikelnummer: _____

---

a) Wandeln Sie die while-Schleife um in eine repeat-Schleife.

b) Wandeln Sie die while-Schleife um in eine for-Schleife.

```
procedure Vielfache (inZahl : tZahlenbereich);
  { gibt alle Zahlen im Bereich tZahlenbereich aus, die Vielfa-
    ches von inZahl sind }

    var
    Vielfaches : tzahlenbereich;
    ??

  begin
    writeln ('Die Vielfachen von ', inZahl, 'sind:');

    ??

  end; { Vielfache }
```

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 01.04.2000

Aufgabe 2 (13 + 3 + 4 = 20 Punkte)

a) Schreiben Sie eine PASCAL-Prozedur `rekMax`, die in einem Feld von `integer-Zahlen` das Maximum bestimmt. Der Prozedur liegt ein ‘divide and conquer'-Algorithmus zugrunde. d.h. wir teilen das Feld in zwei (möglichst gleichgroße) Hälften und bestimmen in beiden Hälften rekursiv das Maximum. Aus dem Vergleich der zwei Teilmaxima ergibt sich das Maximum des ganzen Feldes.
Beschreiben Sie zunächst, wann das Verfahren abbrechen soll und schreiben Sie erst dann die Prozedur.

b) Begründen Sie, ob Sie das Feld als Wert- oder Referenzparameter übergeben.

c) Erläutern Sie, ob es sich um eine sinnvolle Anwendung der Rekursion handelt.

Vorgaben zur Lösung:

Betrachten Sie die folgenden Konstanten- und Typdefinitionen, sowie den Prozedurkopf von `rekMax` als gegeben. Die Parameter `inVon` und `inBis` geben den zu untersuchenden Ausschnitt des Feldes an. Es muß bei Aufruf der Prozedur stets `inVon` ≤ `inBis` gelten. In `outMax` wird schließlich das Maximum der Feldelemente des angegebenen Ausschnitts zurückgegeben. Der Aufruf der Prozedur `rekMax` im Hauptprogramm lautet folglich `rekMax (UNTEN, OBEN, Feld, maxi )`. wenn `Feld` das zu untersuchende Feld angibt und `maxi` eine Variable für das gesuchte Maximum ist.

Tipp:

Ein Feld vom x-ten bis zum y-ten Element lässt sich in 2 (annähernd gleichgroße) Hälfen teilen, indem man ein Trennelement mit dem Index z bestimmt. Der Index kann wie folgt ermittelt werden: z := (x+y) **div** 2. Damit ergeben sich die zwei Teilfelder von x-ten Element bis zum z-ten Element und vom (z+l)-ten Element bis zum y-ten Element.

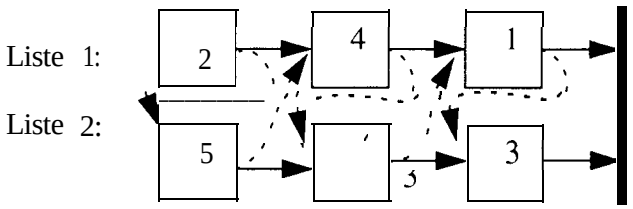
```
const
UNTEN = 0;
OBEN = 10;

type
tIndex = UNTEN..OBEN;
tFeld = array [tIndex] of integer;

procedure rekMax (
                inVon,
                inBis : tIndex;
            ??  ??Feld : tFeld;
            var outMax : integer);
{ bestimmt rekursiv den maximalen Feldeintrag von ??Feld
  ueber ein 'divide and conquer'-Verfahren }
```

Kurs 1613 “Einführung in die imperative Programmierung”

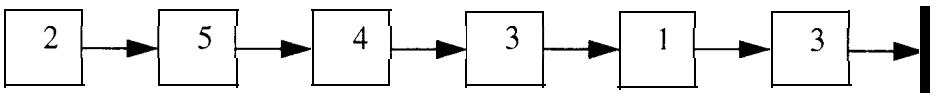
Klausur am 0 1.04.2000

Aufgabe 3 (18 Punkte)

Schreiben Sie eine PASCAL-Funktion `Lis tenMerge`, die zwei lineare Listen mit `integer-Zahlen` nach dem aus dem Straßenverkehr bekannten „Reißverschlußverfahren“ nur durch Änderung der Verkettung zu einer einzigen Liste zusammenfaßt. Als Rückgabewert wird der Zeiger auf das erste Element der Ergebnisliste geliefert. Dabei soll das erste Element der ersten Liste an den Listenanfang kommen, wie im folgenden Beispiel gezeigt ist (die gestrichelten Pfeile deuten die Änderung der Verkettung an):



Resultierende Liste:



Sie können davon ausgehen, daß die beiden Listen **nicht-leer** und **gleich lang** sind. Benutzen Sie folgende Typdefinitionen sowie den angegebenen Funktionskopf von `ListenMerge`:

```
type
tRefListe = ^tListe;
tListe =  record
            info  : integer;
            next : tRefListe
          end;

function ListenMerge (
            inListe1,
            inListe2 : tRefListe) : tRefListe;
{ fasst die Elemente der nicht-leeren und gleich langen Listen
  inListe1 und inListe2 mit dem Reissverschlussverfahren durch
  Aenderung der Verkettung zu einer einzigen Liste zusammen
  und gibt einen Zeiger auf den Anfang der Ergebnisliste
  zurueck }
```

Kurs 1613 “Einführung in die imperative Programmierung”

Zusammenfassung der Muß-Regeln

1. Selbstdefinierte Konstantenbezeichner bestehen nur aus Großbuchstaben. Bezeichner von Standardkonstanten wie z.B. maxint sind also ausgenommen
2. Typbezeichnern wird ein t vorangestellt. Bezeichner von Zeigertypen beginnen mit `tRef`. Bezeichner formaler Parameter beginnen mit in, io oder out.
3. Jede Anweisung beginnt in einer neuen Zeile; **`begin`** und **`end`** stehen jeweils in einer eigenen Zeile
4. Anweisungsfolgen werden zwischen **`begin`** und end um eine konstante Anzahl von 2 - 4 Stellen eingerückt. **`begin`** und **`end`** stehen linksbündig unter der zugehörigen Kontrollanweisung, sie werden nicht weiter eingerückt.
5. Anweisungsteile von Kontrollanweisungen werden genauso eingeruckt.
6. Im Programmkopf wird die Aufgabe beschrieben, die das Programm löst.
7. Jeder Funktions- und Prozedurkopf enthält eine knappe Aufgabenbeschreibung als Kommentar. Ggf. werden zusätzlich die Parameter kommentiert.
8. Die Parameter werden sortiert nach der Übergabeart: Eingangs-, Änderungs- und Ausgangsparameter.
9. Die Übergabeart jedes Parameters wird durch Voranstellen von in, io oder out vor den Parameternamen gekennzeichnet.
10. Das Layout von Funktionen und Prozeduren entspricht dem von Programmen.
11. Jede von einer Funktion oder Prozedur benutzte bzw. manipulierte Variable wird als Parameter übergeben. Es werden keine globalen Variablen manipuliert. Einzige Ausnahme sind Modul-lokale Variablen, die in den Parameterlisten der exportierten Prozeduren und Funktionen des Moduls nicht auftauchen, selbst wenn sie von diesen geändert werden.
12. Jeder nicht von der Prozedur veränderte Parameter wird als Wertparameter übergeben. Lediglich Felder können auch anstatt als Wertparameter als Referenzparameter übergeben werden, um den Speicherplatz für die Kopie und den Kopiervorgang zu sparen. Der Feldbezeichner beginnt aber stets mit dem Präfix in, wenn das Feld nicht verändert wird.
13. Funktionsprozeduren werden wie Funktionen im mathematischen Sinne benutzt, d.h. sie besitzen nur Wertparameter. Wie bei Prozeduren ist eine Ausnahme nur bei Feldern erlaubt, um zusätzlichen Speicherplatz und Kopieraufwand zu vermeiden.
14. Wertparameter werden nicht als lokale Variable mißbraucht.
15. Die Schlüsselworte **`unit`**, **`interface`** und **`implementation`** werden ebensowie **`beginund`** end des Initialisierungsteils linksbündig positioniert. Nach dem Schlüsselwort **`unit`** folgt ein Kommentar, der die Aufgabe beschreibt, welche die Unit löst.
16. Für die Schnittstelle gelten dieselben Programmierstilregeln wie für ein Programm. Dies betrifft Layout und Kommentare. Nach dem Schlüsselwort **`interf ace`** folgt im Normalfall kein Kommentar.
17. Für den Implementationsteil gelten dieselben Programmierstilregeln wie für ein Programm. Nach dem Schlüsselwort **`implementation`** folgt nur dann ein Kommentar, wenn die Realisierung einer Erläuterung bedarf (z.B. wegen komplizierter Datenstrukturen und/oder Algorithmen).
18. In Programmen oder Moduln, die andere Moduln importieren („benutzen“), wird das Schlüsselwort uses auf dieselbe Position eingerückt wie die Schlüsselworte **`const`**, **`type`**, **`var usw.`**
19. Die Laufvariable wird innerhalb einer **`f`** or-Anweisung nicht manipuliert.
20. Die Grundsätze der strukturierten Programmierung sind strikt zu befolgen.