

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 1 —

Aufgabe 1

Die folgenden Anweisungen bilden den Kern eines Algorithmus.

```
WHILE n > 1 DO IF ODD(n) THEN n := 3*n + 1 ELSE n := n DIV 2 END END
```

Von diesem Algorithmus ist bis heute unbekannt, ob er stets terminiert. Man kann also weder voraussetzen, daß der Algorithmus terminiert noch, daß er nicht terminiert. Schreiben Sie ein Modul, welches aus zählt, wie groß die Maximalzahl an Durchläufen durch obiges WHILE-Konstrukt im Intervall $1 \leq n \leq N$ ist. (N ist einzulesen.) Es könnte also sein, daß die WHILE-Schleife für bestimmte Werte von n nicht terminiert; treffen Sie im Programmcode entsprechende Vorkehrungen, und sorgen Sie dafür, daß sich dieser Programmcode auch dann vernünftig testen läßt, falls die WHILE-Schleife stets terminiert.

Aufgabe 2

Schreiben Sie eine Funktionsprozedur `ggT(a, b: INTEGER): INTEGER`, die entsprechend dem euklidischen Algorithmus den größten gemeinsamen Teiler zweier ganzer Zahlen a und b berechnet.

Aufgabe 3

- Implementieren Sie zwei Funktionsprozeduren, welche die Fibonacci-Zahlen rekursiv (siehe Vorlesung) und nicht-rekursiv berechnen.
- Vergleichen Sie die Laufzeiten für Aufrufe dieser beiden Prozeduren, indem Sie in einer Testumgebung Zeitmessungen durchführen.

Hinweis: Für Zeitmessungen unter Verwendung der internen Rechneruhr gibt es in Oberon folgende Möglichkeit:

```
MODULE FibonacciZeitMessung;  
  IMPORT Oberon, ...;  
  VAR t: LONGINT;  
  ...  
  
  t := Oberon.Time();          (* Anzahl Millisekunden, die seit  
  ...                          dem Systemstart verstrichen sind *)  
END FibonacciZeitMessung.
```

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 2 —

Aufgabe 4

Von den zahlreichen Sortierv Verfahren, die es gibt, ist **Bubblesort** eines der besonders einfachen wenn auch nicht besonders effizienten. Der Name resultiert daraus, daß man ein Array **A** sortieren kann, indem man benachbarte Elemente, die noch nicht in der richtigen Reihenfolge in **A** stehen, solange vertauscht, bis alle Elemente in der gewünschten Reihenfolge stehen, also entweder aufsteigend oder absteigend sortiert. Die Elemente steigen in dem Array also entsprechend ihrem Gewicht auf wie die Blasen in einer Flüssigkeit.

Schreiben Sie ein Modul **BubbleSort**, dessen Schnittstelle folgendermaßen vorgegeben sei:

DEFINITION BubbleSort;

```
PROCEDURE Do;  
PROCEDURE In;  
PROCEDURE Out;  
PROCEDURE Set;
```

END BubbleSort.

Eine Liste von Kommandos zur Verwendung dieses Moduls mit typischen Parametern könnte folgende Gestalt haben:

```
BubbleSort.Set absteigend  
BubbleSort.Set aufsteigend
```

```
BubbleSort.In 12 40 43 7 56 888 89  
BubbleSort.Do  
BubbleSort.Out
```

Die exportierten Prozeduren aus **BubbleSort** sollen folgende Aufgaben erfüllen:

- **BubbleSort.Do**: Ausführen der Sortierung und zwar aufsteigend oder absteigend, je nach dem vorherigen Aufruf von **Set**
- **BubbleSort.In**: Einlesen der Werte vom Typ **LONGINT** in ein Array (dessen Name nicht exportiert wird). Die Wertezahl muß frei wählbar sein im Rahmen der Größe des Arrays; sie wird nicht explizit eingegeben.
- **BubbleSort.Out**: Ausgabe des Arrays
- **BubbleSort.Set**: Setzen des Zustands *aufsteigend* oder *absteigend*. Diese beiden Wörter sind zugleich die zulässigen Parameter für **Set**.

Die Ein- und Ausgabe soll sich auf die Bibliotheksmodule **In** und **Out** abstützen.

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 3 —

Aufgabe 5

In der Vorlesung wurde eine Modulschablone angegeben, die den Begriff *Sichtbarkeitsbereich* verdeutlicht. Überzeugen Sie sich von der Wirkungsweise des Mechanismus, der die Sichtbarkeit von globalen und lokalen Variablen regelt, indem Sie diese Modulschablone durch Einführen von diversen globalen und lokalen Variablen erweitern und deren Sichtbarkeit im Detail studieren.

Aufgabe 6

- a) Berechnen Sie unter Verwendung von prozeduralen Variablen und nach Einführung einer Typvereinbarung der Gestalt

```
TYPE RealFunction = PROCEDURE (x: REAL): REAL;
```

das folgende Integral mit der Simpson-Regel:

$$I = \frac{2}{\sqrt{\pi}} \int_0^{\infty} \exp(-x^2) dx$$

- b) Berechnen Sie ebenso das Integral $\int_0^{2000\pi} \sin^2 t dt$ und zwar zunächst ohne Hilfsmittel aus der Numerik (bitte fragen im Zweifelsfall!) und dann numerisch mit der Simpson-Regel, indem Sie in der Bezeichnung der Vorlesung für m den Wert $m = 500$ verwenden. Vergleichen Sie die Ergebnisse. Was läuft hier schief? Wie kann man das auftretende Problem beseitigen?

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 4 —

Aufgabe 7

Schreiben Sie eine Prozedur, welche die folgende Funktion tabelliert:

$$\operatorname{erf} x := \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt$$

Aufgabe 8

Unter einer Differentialgleichung (genauer: einer Differentialgleichung erster Ordnung) versteht man eine Beziehung der Art $y' = F(x, y)$, wobei $F : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ eine gegebene Funktion ist. Die *Differentialgleichung zu lösen* bedeutet, eine Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ zu finden, die $f'(x) = F(x, f(x))$ in $\mathbb{R}$ erfüllt. (Im allgemeinen betrachtet man nicht ganz $\mathbb{R}$, sondern gewisse Teilmengen davon, z.B. Intervalle.) Ein Beispiel: $F(x, y) = y$ führt zur Differentialgleichung $y' = y$ mit der allgemeinen Lösung $y = Ce^x$, wobei $C \in \mathbb{R}$ eine beliebige Konstante ist.

Die meisten Differentialgleichungen, die in praktischen Anwendungen auftreten, lassen sich nicht wie obiges Beispiel durch eine geschlossen angebbare Formel lösen sondern nur numerisch. Eines der einfacheren aber trotzdem wirkungsvollen Verfahren ist die *Integration nach Runge-Kutta*:

Gegeben sei $F(x, y)$. Dann läßt sich die Differentialgleichung $y' = F(x, y)$ folgendermaßen näherungsweise lösen:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1^{(n)} + 2k_2^{(n)} + 2k_3^{(n)} + k_4^{(n)}), \quad \text{wobei} \\ k_1^{(n)} &= hF(x_n, y_n) \\ k_2^{(n)} &= hF(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1^{(n)}) \\ k_3^{(n)} &= hF(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2^{(n)}) \\ k_4^{(n)} &= hF(x_n + h, y_n + k_3^{(n)}) \end{aligned}$$

Dabei ist h die verwendete Schrittweite.

Schreiben Sie eine Funktionsprozedur, mit der das Runge-Kutta-Verfahren implementiert wird.

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 5 —

Aufgabe 9

Schreiben Sie eine Prozedur, welche die allgemeine Matrixmultiplikation entsprechend

$$C_{ij} = \sum_{k=0}^{l-1} A_{ik} B_{kj}$$

ausführt. Diese Prozedur und eventuell weitere zum Rechnen mit Matrizen sollten in einem eigenen Modul stehen, getrennt von der Testprozedur. Überlegen Sie, welche Parameter an die Prozeduren sinnvollerweise übergeben werden sollten.

Aufgabe 10

Schreiben Sie zwei Module, die jeweils eine Array-artige Datenstrukturen mit Komponenten des Typs **REAL** realisieren mit der Bedingung, daß die Modul-Schnittstelle beider Module gleich ist, und schreiben Sie dazu *ein* kleines Testmodul, um die richtige und gleiche Funktionsweise *beider* Module zu überprüfen. Die konkrete Realisierung des einen Moduls soll ein Array mit 5 Elementen sein, die des anderen Moduls ein Record mit 5 Feldern.

Die vom Modulnamen abgesehen gemeinsame Schnittstelle beider Module soll die folgende Gestalt haben:

```
DEFINITION ...;

    PROCEDURE Clear;
    PROCEDURE Get (i: INTEGER): REAL;
    PROCEDURE Put (i: INTEGER; x: REAL);
    PROCEDURE ShowAll;

END ... .
```

Überzeugen Sie sich, daß ein Anwender die interne Realisierung der beiden Module aufgrund ihrer Funktionsweise nicht unterscheiden kann, ohne den Quellcode zur Verfügung zu haben.

Eine Fehlerbehandlung braucht nicht unbedingt implementiert zu werden.

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 6 —

Aufgabe 11

Arbeiten Sie das Beispiel *Personaldatenbank* (Modul PersonenDB) im Detail weiter aus. Es soll insbesondere eine sinnvolle Benutzungsschnittstelle durch Bereitstellung von Oberon-Kommandos in einem gesonderten Modul implementiert werden.

Beispielkommandos wären etwa: Einfügen, Löschen, Suchen von Personendaten. Auf Effizienzüberlegungen soll hier (noch) keine Rücksicht genommen werden: als Basisstruktur kann ein statisches Array gewählt werden.

Aufgabe 12

Implementieren Sie die Datenstruktur *Heap* durch Ausarbeitung und Verwendung der Prozeduren UpHeap(i: INTEGER) und DownHeap(i: INTEGER) in Anlehnung an die in der Vorlesung angegebenen Spezifikationen.

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 7 —

Aufgabe 13

Schreiben Sie ein Modul `LinearAlgebra`, welches einen „Taschenrechner“ für Matrizen realisiert sowie ein Modul `Test` zum Testen des Moduls `LinearAlgebra`.

Die Schnittstellen der beiden Module seien wie folgt vorgegeben:

DEFINITION `LinearAlgebra`;

TYPE

Matrix = POINTER TO MatrixDesc;

PROCEDURE Add (A, B: Matrix): Matrix;

PROCEDURE Multiply (A, B: Matrix): Matrix;

PROCEDURE Read (VAR A: Matrix);

PROCEDURE SetLength;

PROCEDURE SetUnitMatrix (VAR A: Matrix);

PROCEDURE SetZeroMatrix (VAR A: Matrix);

PROCEDURE Write (A: Matrix);

END `LinearAlgebra`.

DEFINITION `Test`;

PROCEDURE Add;

PROCEDURE Multiply;

PROCEDURE Read;

PROCEDURE Reset;

PROCEDURE Write;

END `Test`.

In `Test` sollen mehrere Matrizen eingelesen werden können (Maximalzahl `max` sei statisch gegeben) und dann addiert sowie multipliziert werden können.

Die Aufgabe soll auch als Übung zum Umgang mit dynamischen Arrays und somit auch mit Zeigern dienen; daher sei hier auch noch die (nicht exportierte) Definition von `MatrixDesc` vorgegeben:

MatrixDesc = ARRAY OF ARRAY OF INTEGER;

Eine solche Realisierung ist natürlich nur eine unter mehreren Möglichkeiten.

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 8 —

Aufgabe 14

Den Hauptspeicher eines Rechners kann man sich abstrakt vorstellen als eine Folge von Bits, die einzeln gesetzt und wieder gelöscht werden können. Eine derartige Struktur kann z.B. modelliert werden als ein Array, dessen Komponenten jeweils vom Typ SET sind. Natürlich wird der Anwender einer solchen Struktur auf die Bits direkt per Adresse zugreifen wollen, ohne jedesmal entsprechende Umrechnungen selbst vornehmen zu müssen.

Realisieren Sie eine derartige Struktur als Abstrakten Datentyp (ADT) in einem Modul, dessen Schnittstelle wie folgt vorgegeben sei; die Adressen sollen bei 0 beginnen.

DEFINITION LargeBitsets;

TYPE

LargeBitset = RECORD END;

PROCEDURE HowMany (VAR s: LargeBitset): LONGINT; (* Wie viele Bits sind gesetzt? *)

PROCEDURE Reset (VAR s: LargeBitset; i: LONGINT); (* Bit i löschen *)

PROCEDURE ResetAll (VAR s: LargeBitset); (* alles löschen *)

PROCEDURE Set (VAR s: LargeBitset; i: LONGINT); (* Bit i setzen *)

PROCEDURE ShowAll (VAR s: LargeBitset); (* Inhalt ausgeben als Folge von 0en und 1en*)

END LargeBitsets.

Schreiben Sie außerdem ein Modul zum interaktiven Testen mit folgender Schnittstelle:

DEFINITION TestLargeBitset;

PROCEDURE HowMany;

PROCEDURE Reset;

PROCEDURE ResetAll;

PROCEDURE Set;

PROCEDURE ShowAll;

END TestLargeBitset.

Die Kommandos Set und Reset sollen als Parameter auch mehrere Adressen aufnehmen können.

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 9 —

Aufgabe 15

Abgedruckt ist die abstrakte Spezifikation einer linearen Liste in einer zu Programmiersprachen ähnlichen mathematischen Form. (Erläuterungen wurden in der Vorlesung gegeben.)

Schreiben Sie nach dem in der Vorlesung angedeuteten Muster eine Implementierung dieses ADTs in Oberon, die sich auch äußerlich eng an diese Spezifikation anlehnt.

```
...
TYPE Lists IS
  SORTS
    int, bool, intlist
  OPNS
    true, false: -> bool
    empty: -> intlist
    add: intlist, int -> intlist
    hd: intlist -> int          (* head of list *)
    tl: intlist -> intlist      (* tail of list *)
    hasany: intlist -> bool     (* has the list any elements? *)
  EQNS
    FORALL n: int, ll: intlist
      OFSORT int
        hd (add (ll, n)) = n ;

    OFSORT intlist
      tl (add (ll, n)) = ll ;

    OFSORT bool
      hasany (empty) = false ;
      hasany (add (ll, n)) = true ;
ENDTYPE
...
```

Aufgabe 16

Schreiben Sie ein Oberon-Modul, welches – in Analogie zu dem in der Vorlesung angegebenen entsprechenden Modul für `LinLists` – das interaktive Testen des Moduls `Trees` aus dem Vorlesungsskript mittels Oberon-Kommandos gestattet.

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 10 —

Aufgabe 17

Schreiben Sie drei Module mit den Namen `PriorityQueues`, `StringPriorityQueues` und `Test` entsprechend den unten angegebenen Spezifikationen.

Modul `PriorityQueues` realisiert eine generische Prioritätswarteschlange, die – in direkter Anlehnung an ein entsprechendes Beispiel aus der Vorlesung – als Heap implementiert werden soll.

Der ADT `Item` aus `PriorityQueues` wird in `StringPriorityQueues` spezialisiert zu einem `StringItem`. Als die in einer Heapstruktur erforderliche Vergleichsoperation soll der normale Vergleich von Strings in Oberon gewählt werden.

In `Test` wird eine konkrete Prioritätswarteschlange aus Elementen vom Typ `StringItem` erzeugt; diese soll mittels der angegebenen Kommandos interaktiv getestet werden. Um dem Benutzer die Anwendung der Schnittstelle zu erleichtern, soll das Kommando `Insert` auch eine Liste von Strings direkt einlesen können. Eventuell doppelt auftretende Werte seien dabei zulässig; sie sollen nicht gesondert behandelt werden.

DEFINITION `PriorityQueues`;

TYPE

Item = POINTER TO ItemDesc;

ItemDesc = RECORD

PROCEDURE (x: Item) IsLessThan (y: Item): BOOLEAN;

PROCEDURE (x: Item) Show;

END;

Queue = POINTER TO QueueDesc;

QueueDesc = RECORD

n-: INTEGER;

PROCEDURE (q: Queue) Clear; (* Grundzustand einstellen *)

PROCEDURE (q: Queue) Insert (x: Item); (* Element einfüegen *)

PROCEDURE (q: Queue) Remove (VAR x: Item); (* Element loeschen *)

PROCEDURE (q: Queue) SetSentinel (s: Item); (* Sentinel (abstrakt) setzen *)

PROCEDURE (q: Queue) ShowAll; (* Array-Inhalt zeigen *)

END;

END `PriorityQueues`.

DEFINITION StringPriorityQueues;

IMPORT PriorityQueues;

TYPE

Queue = POINTER TO QueueDesc;

QueueDesc = RECORD (PriorityQueues.QueueDesc)

PROCEDURE (q: Queue) Clear;

END;

StringItem = POINTER TO StringItemDesc;

StringItemDesc = RECORD (PriorityQueues.ItemDesc)

key: ARRAY 64 OF CHAR;

PROCEDURE (x: StringItem) IsLessThan (y: PriorityQueues.Item): BOOLEAN;

PROCEDURE (x: StringItem) Show;

END;

END StringPriorityQueues.

DEFINITION Test;

PROCEDURE Clear;

PROCEDURE Insert;

PROCEDURE Remove;

PROCEDURE ShowAll;

END Test.

Aufgaben zur Informatik II

im WS 00/01

— Übungsblatt 11 —

Aufgabe 18

Erweitern Sie das Programmbeispiel Nr. 5.5.3 (S. 14) des (englischen) Tutorials um ein Kommando **Sort**, mit dem die aktuell vorhandenen Figuren entsprechend der Größe ihrer Fläche nach sortiert werden können. Bedingung dabei ist, dass diese zusätzlich eingefügte Funktion die Erweiterbarkeit der gesamten Programmstruktur nicht zerstört.

Im Einzelnen ist folgendes zu tun:

- a) Schreiben Sie das Modul **Graphics** so um, dass die Figuren in ein Array anstelle der linearen Liste eingefügt werden. Der Einfachheit halber kann das Array statisch gewählt werden (Größe **N**). Bei Überschreiten dieser Grenze soll das Programm mit einem entsprechenden Hinweis an den Benutzer abgebrochen werden.
- b) Fügen Sie im Modul **Draw** das Kommando

```
PROCEDURE Sort*;  
BEGIN  
  G.Sort(g);  
END Sort;
```

hinzu. Dieses ist die einzige Änderung in Modul **Draw**.

Dieser Teil der Aufgabe besteht im wesentlichen darin, die Prozedur **Sort** in Modul **Graphics** auszuarbeiten. Das Sortieren soll mittels **BubbleSort** realisiert werden. In *struktureller* Hinsicht ist das Problem aber unabhängig von der Art des Algorithmus, die Effizienz des Sortierens natürlich nicht.

Beachten Sie, dass die Module **Rectangles** und **Ellipses** von der Änderung nicht betroffen sind und daher auch weitere analoge Module nicht, die möglicherweise später hinzugefügt werden,

Beschreiben Sie Ihre Lösung kurz und möglichst prägnant auf ca. einer Din A4-Seite und fügen Sie den ausgedruckten Programmtext des neuen Moduls **Graphics** bei (ausdrücklich: nur diesen!).

Hinweis: Übernehmen Sie die vorgegebenen Namen der Vorlage soweit wie möglich; Sie vereinfachen damit die Bearbeitung der Aufgabe beträchtlich.