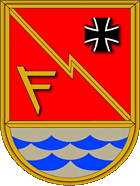
Schule Informationstechnik   
 der Bundeswehr



**Sprachausbildung Ada**



**„Dynamische Datenstrukturen“**

– Lösungen –

Aufgabe 1 - Einführung



Aufgabe 2 - Einführung II

with Ada.Text\_IO;

procedure Uebung2 is

type STRING\_REF is access STRING;

subtype NAME\_TA is STRING(1..10);

subtype PLZ\_TA is STRING(1..5);

type WOHNORTE\_TR is record

Strasse\_Ptr : STRING\_REF;

HausNr : POSITIVE := 1;

Plz : PLZ\_TA := (others => '0');

end record;

type WOHNORTE\_REF is access WOHNORTE\_TR;

type ADRESSE\_TA is array (Positive range <>) of WOHNORTE\_REF;

type PERSON\_TR is record

Vorname\_Ptr : STRING\_REF;

Nachname : NAME\_TA := (others => ' ');

Adressen : ADRESSE\_TA(1..3);

end record;

type AUSWAHL\_T is (J, N);

package AUSWAHL\_IO is new Ada.Text\_IO.Enumeration\_IO(AUSWAHL\_T);

package I\_IO is new Ada.Text\_IO.Integer\_IO(INTEGER);

-- Unterprogramme

-- Spezifikationen

----------------------------------------------------------------

-- GebeWohnort --

----------------------------------------------------------------

-- Beschreibung: --

-- Funktion fragt die Daten (Komponenten) eines Wohnortes --

-- von Tastatur ab und gibt diesen Wohnort als Zeiger --

-- zurueck. --

----------------------------------------------------------------

function GebeWohnort return WOHNORTE\_REF; -- zu Punkt 3

----------------------------------------------------------------

-- Put --

----------------------------------------------------------------

-- Beschreibung: --

-- Prozedure gibt die Daten eines Wohnortes auf dem --

-- Bildschirm aus. --

----------------------------------------------------------------

procedure Put(Wohnort : in WOHNORTE\_TR); -- zu Punkt 5.1

----------------------------------------------------------------

-- Put --

----------------------------------------------------------------

-- Beschreibung: --

-- Prozedure gibt die Daten aller Wohnorte auf dem --

-- Bildschirm aus. --

----------------------------------------------------------------

procedure Put(Adressen : in ADRESSE\_TA); -- zu Punkt 5.1.2

----------------------------------------------------------------

-- Put --

----------------------------------------------------------------

-- Beschreibung: --

-- Prozedure gibt alle Daten einer Person auf dem --

-- Bildschirm aus. --

----------------------------------------------------------------

procedure Put(Person : in PERSON\_TR); -- zu Punkt 5.2

-- Bodies

function GebeWohnort return WOHNORTE\_REF is

Adresse\_Ptr : WOHNORTE\_REF := new WOHNORTE\_TR;

Eingabe : String(1..80) := (others => ' ');

Letzter\_Index : NATURAL := 0;

begin

Ada.Text\_IO.Put("Geben Sie die Strasse ein: ");

Ada.Text\_IO.Get\_Line(Eingabe, Letzter\_Index);

if Eingabe'Last = Letzter\_Index then

Ada.Text\_IO.Skip\_Line;

end if;

Adresse\_Ptr.All.Strasse\_Ptr := new   
 STRING'(Eingabe(Eingabe'First..Letzter\_Index));

Ada.Text\_IO.Put("Geben Sie die Hausnummer ein: ");

I\_IO.Get(Adresse\_Ptr.All.HausNr);

Ada.Text\_IO.Skip\_Line;

Ada.Text\_IO.Put("Geben Sie die PLZ (5-stellig) ein: ");

Ada.Text\_IO.Get(Adresse\_Ptr.All.Plz);

Ada.Text\_IO.Skip\_Line;

return Adresse\_Ptr;

end GebeWohnort;

procedure Put(Wohnort : in WOHNORTE\_TR) is

begin

Ada.Text\_IO.Put\_Line("Wohnort");

Ada.Text\_IO.Put\_Line("=======");

Ada.Text\_IO.Put\_Line("Strasse: " & Wohnort.Strasse\_Ptr.All &   
 POSITIVE'Image(Wohnort.HausNr) );

Ada.Text\_IO.Put\_Line("Plz: " & Wohnort.Plz);

end Put;

procedure Put(Adressen : in ADRESSE\_TA) is

begin

Ada.Text\_IO.Put\_Line("Adressen");

Ada.Text\_IO.Put\_Line("========");

Ada.Text\_IO.New\_Line;

for I in Adressen'Range loop

if Adressen(I) /= null then

Put(Adressen(I).All);

end if;

Ada.Text\_IO.New\_Line;

end loop;

end Put;

procedure Put(Person : in PERSON\_TR) is

begin

Ada.Text\_IO.Put\_Line("Person");

Ada.Text\_IO.Put\_Line("======");

Ada.Text\_IO.Put\_Line("Name: " & Person.Vorname\_Ptr.All & " " &   
 Person.Nachname );

Put(Person.Adressen);

end Put;

-- 1. Aufgabe: eine Person von PERSON\_TR auf dem STACK (statischen Speicher)   
anlegen

Person : PERSON\_TR;

-- 1.1 einen Wohnort mit Initialisierung auf dem HEAP (dynamischen Speicher) anlegen

Wohnort\_Ptr : WOHNORTE\_REF := new WOHNORTE\_TR'(new String'("Bergstr."), 5,   
 "87654");

-- 1.2 Zeichnen Sie den aktuellen Zustand

-- zu 3. Variablen

Weiter : AUSWAHL\_T := J;

AktuelleAnzahl : POSITIVE := Person.Adressen'First;

begin

-- 2. Weisen Sie der Person einen Vornamen und einen Nachnamen zu

Person.Vorname\_Ptr := new STRING'("Peter");

Person.Nachname := "Witzigmann";

-- 3. Nutzer die Moeglichkeit geben, bis zu drei Wohnorte der Person hinzuzufuegen

-- 3.1 Wollen Sie einen weiteren Wohnort eingeben (J/N)?

-- 3.2 Nach Eingabe von drei Wohnorten kein weiteres Nachfragen implementieren

while AktuelleAnzahl <= Person.Adressen'Last and Weiter = J loop

Ada.Text\_IO.Put\_Line("Geben Sie den" & POSITIVE'Image(AktuelleAnzahl) &   
 ". Wohnort ein");

-- Werte einlesen

Person.Adressen(AktuelleAnzahl) := GebeWohnort;

AktuelleAnzahl := AktuelleAnzahl + 1;

if AktuelleAnzahl <= Person.Adressen'Last then

Ada.Text\_IO.Put("Wollen Sie noch weitere Wohnorte hinzufuegen? (J/N) ");

Auswahl\_IO.Get(Weiter);

Ada.Text\_IO.Skip\_Line;

end if;

end loop;

-- 4. Wohnort (von 1.1) der Person als zusaetzlichen Wohnort aufnehmen, wenn

-- dies nicht moeglich, da alle drei Wohnorte schon belegt sind, nachfragen

-- welcher ersetzt werden soll?

if AktuelleAnzahl > Person.Adressen'Last then

Ada.Text\_IO.Put("Welche Adresse soll ersetzt werden? (1-3) ");

I\_IO.Get(AktuelleAnzahl);

Ada.Text\_IO.Skip\_Line;

end if;

Person.Adressen(AktuelleAnzahl) := Wohnort\_Ptr;

-- 5. Ausgaben erzeugen

-- 5.1 Prozedur schreiben, die einen Wohnort (WOHNORT\_TR) ausgibt

-- 5.1.1 Ausgabe des Wohnortes (von 1.1)

Ada.Text\_IO.Put\_Line("Ausgabe des Wohnortes");

Put(Wohnort\_Ptr.All);

-- 5.1.2 Prozedur schreiben, die alle Wohnorte der Person ausgibt

Ada.Text\_IO.New\_Line;

Ada.Text\_IO.Put\_Line("Ausgabe aller Wohnorte");

Put(Person.Adressen);

-- 5.2 Prozedur schreiben, die die gesamten Personendaten (PERSON\_TR) ausgibt

Ada.Text\_IO.New\_Line;

Ada.Text\_IO.Put\_Line("Ausgabe einer Person");

Put(Person);

end Uebung2;

Aufgabe 3 - dynamische Strings

a)

with Ada.Text\_IO;

procedure Namentabelle is

type STRING\_REF is access String;

type REIHE\_TA is array (POSITIVE range <>) of STRING\_REF;

Reihe : REIHE\_TA (1 .. 12);

Puffer : STRING(1 .. 80);

Letzter\_Index : NATURAL := 0;

begin

for I in Reihe'Range loop

Ada.Text\_IO.Put("Bitte" & Integer'Image (I) & ". Namen eingeben: ");

Ada.Text\_IO.Get\_Line(Puffer, Letzter\_Index);

if Puffer'Last = Letzter\_Index then

Ada.Text\_IO.Skip\_Line;

end if;

Reihe(I) := new STRING'(Puffer (1 .. Letzter\_Index));

end loop;

Ada.Text\_IO.Put\_Line ("Ausgabe");

for I in Reihe'Range loop

Ada.Text\_IO.Put\_Line (Reihe(I).All);

end loop;

end Namentabelle;

b) und c) **n i c h t** auf Lösung 2a) aufbauend

with Ada.Text\_IO;

-- c)

with Ada.Unchecked\_Deallocation;

procedure Hoersaal\_Ptr is

-- Typen

-- -- Zeiger auf dyn. unbeschraenkte String-Objekte

type STRING\_REF is access STRING;

-- -- unbeschraenktes Array auf String-Zeiger Objekte nur Adressen auf Strings

type LEHRGANG\_TA is array (POSITIVE range <>) of STRING\_REF;

-- -- Zeiger auf dyn. Reihung

type LEHRGANG\_REF is access LEHRGANG\_TA;

-- Paketauspraegung

package I\_IO is new Ada.Text\_IO.Integer\_IO(POSITIVE);

-- Variablen

Hoersaal\_Ptr : LEHRGANG\_REF;

Anzahl\_Personen : POSITIVE := 1;

-- Zum Einlesen der Namen

Einlese\_String : STRING(1..80) := (others => ' ');

Letzter : NATURAL := 0;

-- c)

-- Das Pragma Controlled schaltet die automatische Speicherbereinigung

-- fuer den angegebenen Zugriffstyp aus

pragma Controlled(STRING\_REF);

procedure Loesche\_Namen is new Ada.Unchecked\_Deallocation(STRING, STRING\_REF );

pragma Controlled(LEHRGANG\_REF);

procedure Loesche\_Lehrgang is new   
 Ada.Unchecked\_Deallocation(LEHRGANG\_TA, LEHRGANG\_REF );

begin

-- Anzahl der Teilnehmer bestimmen

Ada.Text\_IO.Put("Wie viele Teilnehmer hat der Lehrgang?: ");

I\_IO.Get(Anzahl\_Personen);

Ada.Text\_IO.Skip\_Line;

-- dyn. Array fuer die Namen der Teilnehmer erzeugen

Hoersaal\_Ptr := new LEHRGANG\_TA(1..Anzahl\_Personen);

Ada.Text\_IO.New\_Line;

-- einlesen der Namen

for I in Hoersaal\_Ptr.All'Range loop

-- Namen abfragen

Ada.Text\_IO.Put("Geben Sie den " & POSITIVE'Image(I) & ". Namen ein: ");

Ada.Text\_IO.Get\_Line(Item => Einlese\_String,

Last => Letzter);

if Letzter = Einlese\_String'Last then

Ada.Text\_IO.Skip\_Line;

end if;

-- dyn. String-Objekt anlegen und Initialisieren und dem Array zuweisen

Hoersaal\_Ptr.All(I) := new STRING'(Einlese\_String(1..Letzter));

-- oder auf 2 Schritte

-- -- -- 1. Schritt beschraenktes dyn. String Objekt anlegen

-- Hoersaal\_Ptr.All(I) := new STRING(1..Letzter);

-- -- -- 2. Schritt dyn. String Objekt Inhalt zuweisen

-- Hoersaal\_Ptr.All(I).All := Einlese\_String(1..Letzter);

end loop;

Ada.Text\_IO.New\_Line;

-- Ausgabe der Namen

Ada.Text\_IO.Put\_Line("Ausgabe der Namen: ");

for I in Hoersaal\_Ptr'Range loop -- verkuerzte Schreibweise

Ada.Text\_IO.Put\_Line(Hoersaal\_Ptr.All(I).All);

end loop;

-- c)

-- dyn. Speicher fuer die String Objekte wieder freigeben

for I in Hoersaal\_Ptr'Range loop -- verkuerzte Schreibweise

Loesche\_Namen(Hoersaal\_Ptr.All(I));

end loop;

-- dyn. Array fuer die Namen (Lehrgang) wieder freigeben

Loesche\_Lehrgang(Hoersaal\_Ptr);

end Hoersaal\_Ptr;

Aufgabe 4 - Listengröße

-- Loesung ohne Rekursion

function Zaehle\_Elemente (Start\_Ptr : ELEM\_REF) return Natural is

Hilf\_Ptr : ELEM\_REF := Start\_Ptr;

Anzahl : NATURAL := 0;

begin

while Hilf\_Ptr /= null loop

Anzahl := Anzahl + 1;

Hilf\_Ptr := Hilf\_Ptr.Naechster\_Ptr;

end loop;

return Anzahl;

end Zaehle\_Elemente;

-- Loesung mit Rekursion

function Zaehle\_Elemente (Start\_Ptr : ELEM\_REF) return NATURAL is

Anzahl : NATURAL := 0;

begin

if Start\_Ptr = null then

Anzahl := 0;

else

Anzahl := 1 + Zaehle\_Elemente(Start\_Ptr.Naechster\_Ptr);

end if;

return Anzahl;

end Zaehle\_Elemente;

Aufgabe 5 - einfache Listenverwaltung

with Ada.Text\_IO;

with Listenverwaltung;

procedure Main\_Hoersaal\_Liste is

-- Typen

type WEITER\_T is (Ja, Nein);

-- -- Zeiger auf dyn. unbeschraenkte String-Objekte

type STRING\_REF is access STRING;

-- Paketauspraegung

package Hoersaal is new Listenverwaltung(STRING\_REF);

package I\_IO is new Ada.Text\_IO.Integer\_IO(POSITIVE);

package WEITER\_IO is new Ada.Text\_IO.Enumeration\_IO(WEITER\_T);

-- Variablen

-- Zum Einlesen der Namen

Einlese\_String : STRING(1..80) := (others => ' ');

Letzter : NATURAL := 0;

Laufende\_Nummer : NATURAL := 0;

Weiter : WEITER\_T := Ja;

Anzahl\_Teilnehmer : NATURAL := 0;

procedure Anzahl(Item : in out STRING\_REF);

procedure Anzahl(Item : in out STRING\_REF) is

begin

Anzahl\_Teilnehmer := Anzahl\_Teilnehmer + 1;

end Anzahl;

procedure Ausgabe(Item : in out STRING\_REF);

procedure Ausgabe(Item : in out STRING\_REF) is

begin

Ada.Text\_IO.Put\_Line(Item => Item.All);

end Ausgabe;

-- generische Schleifen auspraegen

procedure Ausgabe\_Name is new Hoersaal.Fuer\_alle\_Elemente(Aktion => Ausgabe);

procedure Anzahl\_Namen is new Hoersaal.Fuer\_alle\_Elemente(Aktion => Anzahl);

begin

Ada.Text\_IO.New\_Line;

-- einlesen der Namen

while Weiter = Ja loop

Laufende\_Nummer := Laufende\_Nummer + 1;

-- Namen abfragen

Ada.Text\_IO.Put("Geben Sie den " & NATURAL'Image(Laufende\_Nummer) &   
 ". Namen ein: ");

Ada.Text\_IO.Get\_Line(Item => Einlese\_String, Last => Letzter);

if Letzter = Einlese\_String'Last then

Ada.Text\_IO.Skip\_Line;

end if;

-- dyn. String-Objekt anlegen und Initialisieren und dem Array zuweisen

Hoersaal.Einfuegen\_Hinten(Item => new STRING'(Einlese\_String(1..Letzter)));

Ada.Text\_IO.Put("Moechten Sie weitere Teilnehmer eingeben? (Ja/Nein) ");

WEITER\_IO.Get(Weiter);

Ada.Text\_IO.Skip\_Line;

Ada.Text\_IO.New\_Line;

end loop;

-- Anzahl der Namen ueber eine generische Schleife ermittelt

Anzahl\_Namen;

Ada.Text\_IO.Put("Es sind ");

I\_IO.put(Anzahl\_Teilnehmer, 2);

Ada.Text\_IO.Put\_Line(" Teilnehmer im Hoersaal.");

Ada.Text\_IO.New\_Line;

-- Ausgabe der Namen ueber eine generische Schleife realisiert

Ausgabe\_Name;

end Main\_Hoersaal\_Liste;

Paketspezifikation:

generic

type ELEMENT is private;

package Listenverwaltung is

procedure In\_LIFO (Item : ELEMENT);

procedure In\_FIFO (Item : ELEMENT);

generic

with procedure Aktion (Item : in out ELEMENT);

procedure Fuer\_Alle\_Elemente;

private

type ELEM\_TR;

type ELEM\_REF is access ELEM\_TR;

type ELEM\_TR is record

List\_Item : ELEMENT;

Next\_Ptr : ELEM\_REF;

end record;

Start\_Ptr : ELEM\_REF;

end Listenverwaltung;

Paketbody:

package body Listenverwaltung is

procedure In\_LIFO (Item : ELEMENT) is

begin

Start\_Ptr := new ELEM\_TR'(List\_Item => Item, Next\_Ptr => Start\_Ptr);

end In\_LIFO;

procedure In\_FIFO (Item : ELEMENT) is

Hilf\_Ptr : ELEM\_REF := Start\_Ptr;

begin

if Hilf\_Ptr = null then

Start\_Ptr := new ELEM\_TR'(List\_Item => Item, Next\_Ptr => Start\_Ptr);

else

while Hilf\_Ptr.Next /= null loop

Hilf\_Ptr := Hilf\_Ptr.Next;

end loop;

Hilf\_Ptr.Next := new ELEM\_TR'(List\_Item => Item, Next\_Ptr => null);

end if;

end In\_FIFO;

procedure Fuer\_Alle\_Elemente is

Hilf\_Ptr : ELEM\_REF := Start\_Ptr;

begin

while Hilf\_Ptr /= null loop

Aktion (Hilf\_Ptr.List\_Item);

Hilf\_Ptr := Hilf\_Ptr.Next;

end loop;

end Fuer\_Alle\_Elemente;

end Listenverwaltung;

Aufgabe 6 - Flavius

with Ada.Text\_IO;

with Ada.Unchecked\_Deallocation;

procedure Flavius is

type STRING\_REF is access String; -- zum Verwalten beliebig langer Namen

type ELEM\_TR; -- Vereinbarungen fuer die Ringliste

type ELEM\_REF is access ELEM\_TR;

type ELEM\_TR is record

Name\_Ptr : STRING\_REF;

Next\_Ptr : ELEM\_REF;

end record;

-- Das Pragma Controlled schaltet die automatische Speicherbereinigung

-- fuer den angegebenen Zugriffstyp aus

pragma Controlled(STRING\_REF);

pragma Controlled(ELEM\_REF);

procedure Freigabe\_String is new Ada.Unchecked\_Deallocation(Object => STRING,   
 Name => STRING\_REF);

procedure Freigabe\_Element is new Ada.Unchecked\_Deallocation(Object => ELEM\_TR,   
 Name => ELEM\_REF);

Start\_Ptr : ELEM\_REF; -- zeigt auf den Anfang der Ringliste.

Letzter\_Ptr : ELEM\_REF; -- zeigt immer auf das letzte Element.

Loesch\_Ptr : ELEM\_REF;

Eingabe : STRING (1 .. 80); -- Eingabepuffer fuer Get\_Line,

Last : NATURAL := 0; -- um die Namen einzulesen.

Weiter\_Einlesen : BOOLEAN := True;

begin -- Flavius

Ada.Text\_IO.Put\_Line ("Geben Sie bitte die Namen der Teilnehmer ein!");

Ada.Text\_IO.Put\_Line ("Die Eingabe '\_' beendet das Einlesen.");

-- Eingabeschleife

while Weiter\_Einlesen loop

Ada.Text\_IO.Get\_Line(Eingabe, Last);

if Eingabe'Last = Last then

Ada.Text\_IO.Skip\_Line;

end if;

if Last /= Eingabe'First - 1 then

if Eingabe(Eingabe'First) /= '\_' or Eingabe(1..Last)'Length /= 1 then

if Start\_Ptr = null then -- Einfuegen in eine leere Liste:

Start\_Ptr := new ELEM\_TR'

(Name\_Ptr => new STRING'(Eingabe (1 .. Last)),

Next\_Ptr => null);

Letzter\_Ptr := Start\_Ptr;

else -- Einfuegen in eine nicht leere Liste: hinten anfuegen

Letzter\_Ptr.Next\_Ptr := new ELEM\_TR'

(new STRING'(Eingabe(1 .. Last)), Start\_Ptr);

Letzter\_Ptr := Letzter\_Ptr.Next\_Ptr;

end if;

else

Weiter\_Einlesen := False;

end if;

end if;

end loop;

-- Ring schliessen

if Start\_Ptr /= null then

Letzter\_Ptr.Next\_Ptr := Start\_Ptr;

end if;

-- jedes dritte Element aus der Ringliste aushaengen:

while Start\_Ptr /= null and then Start\_Ptr /= Start\_Ptr.Next\_Ptr loop

Loesch\_Ptr := Start\_Ptr.Next\_Ptr.Next\_Ptr;

-- Ausgeben des Opfers:

Ada.Text\_IO.Put\_Line("Naechstes Opfer : " & (Loesch\_Ptr.All.Name\_Ptr.All));

-- Ausketten des Opfers:

Start\_Ptr.Next\_Ptr.Next\_Ptr := Loesch\_Ptr.All.Next\_Ptr;

-- Speicher freigeben

Freigabe\_String(Loesch\_Ptr.All.Name\_Ptr);

Freigabe\_Element(Loesch\_Ptr);

-- auf die Person nach dem Opfer weiterschalten:

Start\_Ptr := Start\_Ptr.all.Next\_Ptr.All.Next\_Ptr;

end loop;

if Start\_Ptr = null then

Ada.Text\_IO.Put\_Line("Das Spiel erfordert mindestens einen Mitspieler!");

else

Ada.Text\_IO.Put\_Line ("Ueberlebender ist : " & Start\_Ptr.Name\_Ptr.All);

end if;

end Flavius;

Aufgabe 7 - Rückwärtskette

procedure Rueckwaertsverkettung (Start\_Ptr : in ELEM\_REF;   
 Ende\_Ptr : out ELEM\_REF) is

Vor\_Ptr : ELEM\_REF;

Hilf\_Ptr : ELEM\_REF := Start\_Ptr;

begin

while Hilf\_Ptr /= null loop

Hilf\_Ptr.Vorgaenger\_Ptr := Vor\_Ptr;

Vor\_Ptr := Hilf\_Ptr;

Hilf\_Ptr := Hilf\_Ptr.Nachfolger\_Ptr;

end loop;

Ende\_Ptr := Vor\_Ptr;

end Rueckwaertsverkettung;

Aufgabe 8 - Polynomliste

function Wert (P\_Ptr : POLYNOM\_REF; X : FLOAT) return FLOAT is

Summe : FLOAT := 0.0;

Hilf\_Ptr : POLYNOM\_REF := P\_Ptr;

begin -- Wert

while Hilf\_Ptr /= null loop

Summe := Summe + Hilf\_Ptr.Koeffizient \* X \*\* Hilf\_Ptr.Exponent;

Hilf\_Ptr := Hilf\_Ptr.Verweis\_Ptr;

end loop;

return Summe;

end Wert;

Aufgabe 9 - Mengenverwaltung

-- Dieses Paket dient zur Verwaltung von Mengen beliebigen

-- Elementtyps.

--

-- Es werden folgende Operationen zur Verfuegung gestellt:

-- \* Zuweisung

-- \* Aufnahme eines Elementes (vorne oder hinten)

-- \* Entfernen eines angegebenen Elementes

-- \* Entnahme des ersten Elementes einer Menge,

-- das zur Verarbeitung zur Verfuegung gestellt wird

-- \* Test auf Mengengleichheit

-- \* Test ob ein angegebenes Element in einer Menge

-- enthalten ist

-- \* Kardinalitaet

-- (Anzahl Elemente) einer Menge

--

-- Als Konstante ist die leere Menge definiert.

--------------------------------------------------------------------

generic

type ELEMENT is limited private;

with procedure Zuweisung (L : out ELEMENT; R : in ELEMENT);

with function "=" (L, R : in ELEMENT) return BOOLEAN is <>;

with function "<" (L, R : in ELEMENT) return BOOLEAN is <>; -- fuer Sortierung   
 -- notwendig

package Mengen\_Paket is

type MENGE is limited private;

Leere\_Menge : constant MENGE;

procedure Zuweisung (Nach : in out MENGE; Von : in MENGE);

procedure Einf\_LIFO (M : in out MENGE; E : in ELEMENT);

procedure Einf\_FIFO (M : in out MENGE; E : in ELEMENT);

procedure Entferne (M : in out MENGE; E : in ELEMENT);

-- Anmerkung: Funktion hier nicht moeglich, da ELEMENT ein limited private Typ   
 -- ist, return ELEMENT nicht erlaubt!

procedure Entnahme (M : in out MENGE; E : out ELEMENT);

procedure Sortieren (M : in out MENGE);

generic

with procedure Aktion( E : in out ELEMENT );

procedure Fuer\_Alle\_Elemente(M : in MENGE);

function "=" (L, R : in MENGE) return BOOLEAN;

function Ist\_Enthalten (M : in MENGE; E : in ELEMENT) return BOOLEAN;

function Kardinalitaet (M : in MENGE) return NATURAL;

Zugriff\_Auf\_Leere\_Menge : exception;

private

type ELEM\_TR;

type ELEM\_REF is access ELEM\_TR;

type ELEM\_TR is record

Info : ELEMENT;

Next\_Ptr : ELEM\_REF;

end record;

pragma Controlled(ELEM\_REF);

type MENGE is record

Anzahl\_Elemente : NATURAL := 0;

Start\_Element\_Ptr : ELEM\_REF;

end record;

Leere\_Menge : constant MENGE := (0, null);

end Mengen\_Paket;

with Ada.Unchecked\_Deallocation;

package body Mengen\_Paket is

procedure Freigabe is new Ada.Unchecked\_Deallocation(Object => ELEM\_TR,

Name => ELEM\_REF);

procedure AuskettenElement(Start\_Ptr : in out ELEM\_REF; Min\_Ptr : out ELEM\_REF);

procedure AuskettenElement(Start\_Ptr : in out ELEM\_REF; E : in ELEMENT);

-----------------------------------------------------------------

procedure Zuweisung (Nach : in out MENGE; Von : in MENGE) is

Hilfs\_Ptr : ELEM\_REF := Von.Start\_Element\_Ptr;

begin -- Zuweisung

Nach := Leere\_Menge;

while Hilfs\_Ptr /= null loop

Einf\_FIFO (Nach, Hilfs\_Ptr.Info);

Hilfs\_Ptr := Hilfs\_Ptr.Next\_Ptr;

end loop;

end Zuweisung;

-----------------------------------------------------------------

procedure Fuer\_Alle\_Elemente(M : in MENGE) is

Hilfs\_Ptr : ELEM\_REF := M.Start\_Element\_Ptr;

begin

while Hilfs\_Ptr /= null loop

-- Aktion ausfuehren

Aktion(Hilfs\_Ptr.All.Info);

-- weiterschalten zum naechsten Element

Hilfs\_Ptr := Hilfs\_Ptr.All.Next\_Ptr;

end loop;

end Fuer\_Alle\_Elemente;

-----------------------------------------------------------------

function Kardinalitaet (M : in MENGE) return NATURAL is

begin -- Kardinalitaet

return M.Anzahl\_Elemente;

end Kardinalitaet;

-----------------------------------------------------------------

function Ist\_Enthalten (M : in MENGE; E : in ELEMENT) return BOOLEAN is

Hilfs\_Ptr : ELEM\_REF := M.Start\_Element\_Ptr;

Enthalten : BOOLEAN := False;

begin -- Ist\_enthalten

while not Enthalten and Hilfs\_Ptr /= null loop

if Hilfs\_Ptr.Info = E then

Enthalten := True;

else

Hilfs\_Ptr := Hilfs\_Ptr.Next\_Ptr;

end if;

end loop;

return Enthalten;

end Ist\_Enthalten;

-----------------------------------------------------------------

function "=" (L, R : in MENGE) return BOOLEAN is

Hilfs\_Ptr : ELEM\_REF := R.Start\_Element\_Ptr;

Gleich : BOOLEAN := True;

begin -- "="

if Kardinalitaet(L) = Kardinalitaet(R) then

while Gleich and Hilfs\_Ptr /= null loop

if not Ist\_Enthalten(L, Hilfs\_Ptr.Info) then

Gleich := False;

else

Hilfs\_Ptr := Hilfs\_Ptr.Next\_Ptr;

end if;

end loop;

else

Gleich := False;

end if;

return Gleich;

end "=";

-----------------------------------------------------------------

procedure Einf\_LIFO (M : in out MENGE; E : in ELEMENT) is

Hilfs\_Ptr : ELEM\_REF;

begin -- Einf\_LIFO

if not Ist\_Enthalten(M, E) then

Hilfs\_Ptr := new ELEM\_TR; -- Initialisierung nicht moeglich!

Zuweisung (Hilfs\_Ptr.Info, E);

Hilfs\_Ptr.Next\_Ptr := M.Start\_Element\_Ptr;

M.Start\_Element\_Ptr := Hilfs\_Ptr;

M.Anzahl\_Elemente := M.Anzahl\_Elemente + 1;

end if;

end Einf\_LIFO;

---------------------------------------------------------------------

procedure Einf\_FIFO (M : in out MENGE; E : in ELEMENT) is

Hilfs\_Ptr : ELEM\_REF := M.Start\_Element\_Ptr;

begin

if Hilfs\_Ptr = null then --Liste leer

Einf\_LIFO(M, E);

elsif not Ist\_Enthalten(M, E) then

while Hilfs\_Ptr.All.Next\_Ptr /= null loop --"vorspulen" auf letztes element

Hilfs\_Ptr := Hilfs\_Ptr.All.Next\_Ptr;

end loop;

Hilfs\_Ptr.All.Next\_Ptr := new ELEM\_TR; -- Initialisierung nicht moeglich!

Zuweisung(Hilfs\_Ptr.All.Next\_Ptr.All.Info, E);

M.Anzahl\_Elemente := M.Anzahl\_Elemente + 1;

end if;

end Einf\_FIFO;

----------------------------------------------------------------------------

procedure AuskettenElement (Start\_Ptr : in out ELEM\_REF; E : in ELEMENT) is

Ausf\_Ptr, Vorg\_Ptr : ELEM\_REF := Start\_Ptr;

begin

-- Ausfuegestelle suchen

while Ausf\_Ptr /= null and then Ausf\_Ptr.Info /= E loop

Vorg\_Ptr := Ausf\_Ptr;

Ausf\_Ptr := Ausf\_Ptr.Next\_Ptr;

end loop;

-- das Element mit der Info E ausketten

if Ausf\_Ptr /= null and then Ausf\_Ptr.Info = E then

if Ausf\_Ptr = Vorg\_Ptr then -- am Listenanfang entfernen

Start\_Ptr := Start\_Ptr.Next\_Ptr;

else -- in der Liste und am Listenende

Vorg\_Ptr.Next\_Ptr := Ausf\_Ptr.Next\_Ptr;

end if;

end if;

end AuskettenElement;

----------------------------------------------------------------------------

-- Prozedure gibt Menge (Start\_Ptr) ohne kleinstes Listenelement und

-- den Zeiger auf kleinstes Listenelement zurueck.

-----------------------------------------------------------------------------

procedure AuskettenElement(Start\_Ptr : in out ELEM\_REF;   
 Min\_Ptr : out ELEM\_REF) is

Hilfs\_Ptr : ELEM\_REF := Start\_Ptr;

begin

-- suche kleinstes Element

Min\_Ptr := Start\_Ptr;

while Hilfs\_Ptr /= null loop

if Hilfs\_Ptr.All.Info < Min\_Ptr.All.Info then

Min\_Ptr := Hilfs\_Ptr;

end if;

Hilfs\_Ptr := Hilfs\_Ptr.All.Next\_Ptr;

end loop;

-- kettet kleinstes Listenelement aus

AuskettenElement(Start\_Ptr => Start\_Ptr,

E => Min\_Ptr.all.Info);

end AuskettenElement;

procedure Sortieren (M : in out MENGE) is

Vorgaenger\_Ptr : ELEM\_REF := M.Start\_Element\_Ptr;

Aktueller\_Ptr : ELEM\_REF := M.Start\_Element\_Ptr;

Rest\_Menge\_Ptr : ELEM\_REF;

Min\_Ptr : ELEM\_REF;

begin

while Aktueller\_Ptr /= null loop

Rest\_Menge\_Ptr := Aktueller\_Ptr;

AuskettenElement(Rest\_Menge\_Ptr, Min\_Ptr);

-- gefundenes Listenelement ist immer Vorgaenger der restlichen Menge

Min\_Ptr.All.Next\_Ptr := Rest\_Menge\_Ptr;

if Aktueller\_Ptr = Vorgaenger\_Ptr then -- Listenanfang

M.Start\_Element\_Ptr := Min\_Ptr;

else

Vorgaenger\_Ptr.All.Next\_Ptr := Min\_Ptr;

end if;

Aktueller\_Ptr := Min\_Ptr;

-- weiterschalten zum naechsten Listenelement

Vorgaenger\_Ptr := Aktueller\_Ptr;

Aktueller\_Ptr := Aktueller\_Ptr.All.Next\_Ptr;

end loop;

end Sortieren;

-----------------------------------------------------------------

procedure Entferne (M : in out MENGE; E : in ELEMENT) is

Ausf\_Ptr, Vorg\_Ptr : ELEM\_REF := M.Start\_Element\_Ptr;

begin -- Entferne

-- Ausfuegestelle suchen

while Ausf\_Ptr /= null and then Ausf\_Ptr.Info /= E loop

Vorg\_Ptr := Ausf\_Ptr;

Ausf\_Ptr := Ausf\_Ptr.Next\_Ptr;

end loop;

-- das Element mit der Info E loeschen

if Ausf\_Ptr /= null and then Ausf\_Ptr.Info = E then

if Ausf\_Ptr = Vorg\_Ptr then -- am Listenanfang entfernen

M.Start\_Element\_Ptr := M.Start\_Element\_Ptr.Next\_Ptr;

else -- mittendrin oder am Listenende entfernen

Vorg\_Ptr.Next\_Ptr := Ausf\_Ptr.Next\_Ptr;

end if;

M.Anzahl\_Elemente := M.Anzahl\_Elemente - 1;

Freigabe(Ausf\_Ptr);

end if;

end Entferne;

-----------------------------------------------------------------

procedure Entnahme (M : in out MENGE; E : out ELEMENT) is

Hilfs\_Ptr : ELEM\_REF := M.Start\_Element\_Ptr;

begin -- Entnahme

if M = Leere\_Menge then

raise Zugriff\_Auf\_Leere\_Menge;

else

Zuweisung (E, M.Start\_Element\_Ptr.All.Info);

M.Start\_Element\_Ptr := M.Start\_Element\_Ptr.Next\_Ptr;

M.Anzahl\_Elemente := M.Anzahl\_Elemente - 1;

Freigabe(Hilfs\_Ptr);

end if;

end Entnahme;

end Mengen\_Paket;

Aufgabe 10 - Blätter zählen

function Anzahl\_Blaetter (Wurzel\_Ptr : KNOTEN\_REF) return NATURAL is

Erg : NATURAL := 0;

begin

if Wurzel\_Ptr = null then

Erg := 0;

elsif Wurzel\_Ptr.Links\_Ptr = null and Wurzel\_Ptr.Rechts\_Ptr = null then

Erg := 1;

else

Erg := Anzahl\_Blaetter (Wurzel\_Ptr.Links\_Ptr) +   
 Anzahl\_Blaetter (Wurzel\_Ptr.Rechts\_Ptr);

end if;

return Erg;

end Anzahl\_Blaetter;

Aufgabe 11 - Baumhöhe

function Hoehe (Wurzel\_Ptr : KNOTEN\_REF) return NATURAL is

Hoehe\_Links,

Hoehe\_Rechts : NATURAL := 0;

Erg : NATURAL := 0;

begin

if Wurzel\_Ptr = null then

Erg := 0;

else

Hoehe\_Links := Hoehe(Wurzel\_Ptr.Links\_Ptr);

Hoehe\_Rechts := Hoehe(Wurzel\_Ptr.Rechts\_Ptr);

if Hoehe\_Links > Hoehe\_Rechts then

Erg := 1 + Hoehe\_Links;

else

Erg := 1 + Hoehe\_Rechts;

end if;

end if;

return Erg;

end Hoehe;

-- alternativ

function Hoehe (Wurzel\_Ptr : KNOTEN\_REF) return NATURAL is

Erg : NATURAL := 0;

begin

if Wurzel\_Ptr = null then

Erg := 0;

else

Erg := 1 + NATURAL'Max (Hoehe(Wurzel\_Ptr.Links\_Ptr),   
 Hoehe(Wurzel\_Ptr.Rechts\_Ptr));

end if;

return Erg;

end Hoehe;

Aufgabe 12 - Character-Baum

with Ada.Text\_IO;

procedure Buchstaben is

-- fuer die Eingabe des Texts durch den Benutzer:

Eingabebereich : STRING(1 .. 80) := (others => ' ');

EingabeLast : NATURAL := 0;

type KNOTEN\_TR;

type KNOTEN\_REF is access KNOTEN\_TR;

type KNOTEN\_TR is record

Zeichen : CHARACTER;

Zaehler : NATURAL;

Links\_Ptr : KNOTEN\_REF;

Rechts\_Ptr : KNOTEN\_REF;

end record;

package Nat\_IO is new Ada.Text\_IO.Integer\_IO (NATURAL);

Wurzel\_Ptr : KNOTEN\_REF;

--------------------------------------------------------------------------

procedure Ausgabe (Wurzel\_Ptr : in KNOTEN\_REF) is

begin -- Ausgabe

if Wurzel\_Ptr /= null then

Ausgabe (Wurzel\_Ptr.Links\_Ptr);

Ada.Text\_IO.Put (Wurzel\_Ptr.Zeichen);

Nat\_IO.Put (Wurzel\_Ptr.Zaehler, 5);

Ada.Text\_IO.New\_Line;

Ausgabe (Wurzel\_Ptr.Rechts\_Ptr);

end if;

end Ausgabe;

--------------------------------------------------------------------------

procedure Einfuegen (Wurzel\_Ptr : in out KNOTEN\_REF; Zeichen : in CHARACTER) is

Hilf\_Ptr : KNOTEN\_REF := Wurzel\_Ptr;

begin -- Einfuegen

if Wurzel\_Ptr = null then -- leerer Baum

Wurzel\_Ptr :=

new KNOTEN\_TR'

(Zeichen => Zeichen,

Zaehler => 1,

Links\_Ptr => null,

Rechts\_Ptr => null);

else

if Wurzel\_Ptr.Zeichen = Zeichen then -- Zeichen gefunden

Wurzel\_Ptr.Zaehler := Wurzel\_Ptr.Zaehler + 1;

elsif Zeichen < Wurzel\_Ptr.Zeichen then -- links weitersuchen

Einfuegen (Wurzel\_Ptr.Links\_Ptr, Zeichen);

else -- rechts weitersuchen

Einfuegen (Wurzel\_Ptr.Rechts\_Ptr, Zeichen);

end if;

end if;

end Einfuegen;

begin -- Buchstaben

Ada.Text\_IO.Put ("Geben Sie den Text ein : ");

Ada.Text\_IO.Get\_Line (Eingabebereich, EingabeLast);

if Eingabebereich'Last = EingabeLast then

Ada.Text\_IO.Skip\_Line;

end if;

-- Aufbau des Baums:

for I in Eingabebereich'First .. EingabeLast loop

Einfuegen (Wurzel\_Ptr, Eingabebereich (I));

end loop;

-- Ausgabe des Baums:

Ada.Text\_IO.Put\_Line ("Nach Buchstaben sortierte Haeufigkeit:");

Ausgabe(Wurzel\_Ptr);

end Buchstaben;

Aufgabe 13 - Kanten zählen

function Anzahl\_Kanten (G : GRAPH\_T) return NATURAL is

Anzahl : NATURAL := 0;

Aktueller\_Knoten\_Ptr : KNOTEN\_REF := G;

Aktuelle\_Kante\_Ptr : KANTEN\_REF;

begin -- Anzahl\_Kanten

while Aktueller\_Knoten\_Ptr /= null loop

Aktuelle\_Kante\_Ptr := Aktueller\_Knoten\_Ptr.Kantenliste\_Ptr;

while Aktuelle\_Kante\_Ptr /= null loop

Anzahl := Anzahl + 1;

Aktuelle\_Kante\_Ptr := Aktuelle\_Kante\_Ptr.Naechster\_Ptr;

end loop;

Aktueller\_Knoten\_Ptr := Aktueller\_Knoten\_Ptr.Naechster\_Ptr;

end loop;

return Anzahl;

end Anzahl\_Kanten;

Aufgabe 14 - ungerichteter Graph

procedure Bilde\_Ungerichteten\_Graph (G : in GRAPH\_T) is

Akt\_Knoten\_Ptr : KNOTEN\_REF := G;

Akt\_Kanten\_Ptr : KANTEN\_REF;

-- Hilfsfunktion zur Ueberpruefung, ob eine Kante existiert:

function Vorhanden (Start\_Ptr, Ziel\_Ptr : in KNOTEN\_REF) return BOOLEAN is

Kanten\_Ptr : KANTEN\_REF := Start\_Ptr.Kantenliste\_Ptr;

Ist\_Vorhanden : BOOLEAN := False;

begin -- Vorhanden

while Kanten\_Ptr /= null and not Ist\_Vorhanden loop

if Kanten\_Ptr.Endknoten\_Ptr = Ziel\_Ptr then

Ist\_Vorhanden := True;

end if;

Kanten\_Ptr := Kanten\_Ptr.Naechster\_Ptr;

end loop;

return Ist\_Vorhanden;

end Vorhanden;

begin -- Bilde\_ungerichteten\_Graph

while Akt\_Knoten\_Ptr /= null loop

Akt\_Kanten\_Ptr := Akt\_Knoten\_Ptr.Kantenliste\_Ptr;

while Akt\_Kanten\_Ptr /= null loop

-- wenn zu einer vorhandenen Kante keine Gegenkante existiert, dann wird

-- diese Gegenkante vorne in die Kantenliste des Zielknotens neu eingefügt:

if not Vorhanden (Start => Akt\_Kanten\_Ptr.Endknoten\_Ptr,   
 Ziel\_Ptr => Akt\_Knoten\_Ptr) then

Akt\_Kanten\_Ptr.Endknoten\_Ptr.Kantenliste\_Ptr := new KANTEN\_TR'(

Endknoten\_Ptr => Akt\_Knoten\_Ptr,

Info\_Ptr => Akt\_Kanten\_Ptr.Info\_Ptr,

Markierung => Akt\_Kanten\_Ptr.Markierung,

Naechster\_Ptr => Akt\_Kanten\_Ptr.Endknoten\_Ptr.Kantenliste\_Ptr);

end if;

Akt\_Kanten\_Ptr := Akt\_Kanten\_Ptr.Naechster\_Ptr;

end loop;

Akt\_Knoten\_Ptr := Akt\_Knoten\_Ptr.Naechster\_Ptr;

end loop;

end Bilde\_Ungerichteten\_Graph;

Aufgabe 15 - Graphen-Markierungen löschen

procedure Markierungen\_Loeschen (G : in GRAPH\_T) is

Aktueller\_Knoten\_Ptr : KNOTEN\_REF := G;

Aktueller\_Kanten\_Ptr : KANTEN\_REF;

begin -- Loeschen

while Aktueller\_Knoten\_Ptr /= null loop

Aktueller\_Knoten\_Ptr.Markierung := 0;

Aktueller\_Kanten\_Ptr := Aktueller\_Knoten.Kantenliste\_Ptr;

while Aktueller\_Kanten\_Ptr /= null loop

Aktueller\_Kanten\_Ptr.Markierung := 0;

Aktueller\_Kanten\_Ptr := Aktueller\_Kanten\_Ptr.Naechster\_Ptr;

end loop;

Aktueller\_Knoten\_Ptr := Aktueller\_Knoten\_Ptr.Naechster\_Ptr;

end loop;

end Markierungen\_Loeschen;

Aufgabe 16 - Isolierte Knoten

function Isolierte\_Knoten (G : GRAPH\_T) return BOOLEAN is

Aktueller\_Knoten\_Ptr : KNOTEN\_REF := G;

Aktueller\_Kanten\_Ptr : KANTEN\_REF;

Isoliert\_Vorhanden : BOOLEAN := False;

begin -- Isolierte\_Knoten

-- alle Knotenmarkierungen auf 0 setzen:

Markierungen\_Loeschen(G); -- siehe Aufgabe 14

while Aktueller\_Knoten\_Ptr /= null loop

Aktueller\_Kanten\_Ptr := Aktueller\_Knoten\_Ptr.Kantenliste\_Ptr;

while Aktueller\_Kanten\_Ptr /= null loop

-- Schlingen nicht weiter betrachten

if Aktueller\_Kanten\_Ptr.Endknoten\_Ptr /= Aktueller\_Knoten\_Ptr then

-- der aktuelle Knoten wird markiert,

-- weil mindestens eine Kante von ihm abgeht

Aktueller\_Knoten\_Ptr.Markierung := 1;

-- der Endknoten der aktuellen Kante wird markiert,

-- weil mindestens eine Kante auf ihn zeigt

Aktueller\_Kanten\_Ptr.Endknoten\_Ptr.Markierung := 1;

end if;

Aktueller\_Kanten\_Ptr := Aktueller\_Kanten\_Ptr.Naechster\_Ptr;

end loop;

Aktueller\_Knoten\_Ptr := Aktueller\_Knoten\_Ptr.Naechster\_Ptr;

end loop;

-- isolierte Knoten suchen:

Aktueller\_Knoten\_Ptr := G;

while not Isoliert\_Vorhanden and Aktueller\_Knoten\_Ptr /= null loop

if Aktueller\_Knoten\_Ptr.Markierung = 0 then

Isoliert\_Vorhanden := True;

end if;

Aktueller\_Knoten\_Ptr := Aktueller\_Knoten\_Ptr.Naechster\_Ptr;

end loop;

return Isoliert\_Vorhanden;

end Isolierte\_Knoten;

Praktikum 1 - Gefädelter Binärbaum

Paketspezifikation

package Nadelbaum is

type SCHLUESSEL\_T is range 1 .. 1000;

subtype INFORMATION\_TA is STRING(1 .. 5);

function Suchen (S : SCHLUESSEL\_T) return INFORMATION\_TA;

procedure Einfuegen (S : in SCHLUESSEL\_T; I : in INFORMATION\_TA);

procedure Ausgabe;

procedure AusgabeR;

Schluessel\_Nicht\_Gefunden, Schluessel\_Schon\_Vorhanden : exception;

Baum\_Leer : exception;

private

type KNOTEN\_TR;

type BAUM\_REF is access KNOTEN\_TR;

type KNOTEN\_TR is record

Schl : SCHLUESSEL\_T;

Info : INFORMATION\_TA;

Links\_Ptr, Rechts\_Ptr : BAUM\_REF;

Links\_Echt, Rechts\_Echt : Boolean;

end record;

Baum\_Ptr : BAUM\_REF;

end Nadelbaum;

Paketbody

with Ada.Text\_IO;

package body Nadelbaum is

procedure Einfuegen (S : in SCHLUESSEL\_T; I : in INFORMATION\_TA) is

H\_Ptr : BAUM\_REF := Baum\_Ptr;

Hilf\_Ptr : BAUM\_REF;

Stoppen : Boolean := False;

begin

if Baum\_Ptr = null then

Baum\_Ptr := new KNOTEN\_TR'(S, I, null, null, False, False);

else

while not Stoppen loop

if S = H\_Ptr.Schl then

raise Schluessel\_Schon\_Vorhanden;

elsif S < H\_Ptr.Schl then

if H\_Ptr.Links\_Ptr = null then --neues kleinstes

H\_Ptr.Links\_Ptr :=

new KNOTEN\_TR'(S, I, null, H\_Ptr, False, False);

H.Links\_Echt := True;

Stoppen := True;

else

if not H\_Ptr.Links\_Echt then

Hilf\_Ptr :=

new KNOTEN\_TR'(S, I, H\_Ptr.Links\_Ptr, H\_Ptr, False, False);--55

H\_Ptr.Links\_Ptr := Hilf\_Ptr;

H\_Ptr.Links\_Echt := True;

Stoppen := True;

else

H\_Ptr := H\_Ptr.Links\_Ptr;

end if;

end if;

else

if H\_Ptr.Rechts\_Ptr = null then --neues groesstes

H\_Ptr.Rechts\_Ptr :=

new KNOTEN\_TR'(S, I, H, null, False, False);

H\_Ptr.Rechts\_Echt := True;

Stoppen := True;

else

if not H\_Ptr.Rechts\_Echt then

Hilf\_Ptr :=

new KNOTEN\_TR'(S, I, H\_Ptr, H\_Ptr.Rechts\_Ptr, False, False);--48

H\_Ptr.Rechts\_Ptr := Hilf\_Ptr;

H\_Ptr.Rechts\_Echt := True;

Stoppen := True;

else

H\_Ptr := H\_Ptr.Rechts\_Ptr;

end if;

end if;

end if;

end loop;

end if;

end Einfuegen;

function Suchen (S : SCHLUESSEL\_T) return INFORMATION\_TA is

H : BAUM\_REF := B;

Gefunden : Boolean := False;

Info : INFORMATION\_TA;

begin

if H = null then

raise Baum\_Leer;

else

while not gefunden loop

while H.Links\_Echt loop

H := H.Links\_Ptr;

end loop;

if H.Schl = S then

Gefunden := True;

end if;

while not H.Rechts\_Echt and H.Rechts\_Ptr /= null and not gefunden loop

H := H.Rechts\_Ptr;

if H.Schl = S then

Gefunden := True;

end if;

end loop;

H := H.Rechts\_Ptr;

end loop Aussen;

if not Gefunden then

raise Schluessel\_Nicht\_Gefunden;

else

Info := H.Info;

end if;

end if;

return Info;

end Suchen;

procedure Ausgabe is

H\_Ptr : BAUM\_REF := Baum\_Ptr; -- Wurzel in Hilfsvariable uebernehmen

Weiter : BOOLEAN := True; -- Schleifenbedingung

package Schluessel\_IO is new Ada.Text\_IO.Integer\_IO (SCHLUESSEL\_T);

begin

if H\_Ptr /= null then -- leerer Baum?

while Weiter loop

-- nach links gehen, bis ein unechter Verweis kommt

-- (gefaedelt oder null)

while H\_Ptr.Links\_Echt loop

H\_Ptr := H\_Ptr.Links\_Ptr;

end loop;

Schluessel\_IO.Put(H\_Ptr.Schl);

Ada.Text\_IO.Put\_Line(" : " & H\_Ptr.Info);

-- nach rechts gehen, bis ein echter Verweis kommt. Dabei auf

-- jeden besuchten KNOTEN\_TR die Aktion auffuehren

while not H\_Ptr.Rechts\_Echt and Weiter loop

if H\_Ptr.Rechts\_Ptr = null then -- Verarbeitungsende?

Weiter := False;

else

H\_Ptr := H\_Ptr.Rechts\_Ptr;

Schluessel\_IO.Put(H\_Ptr.Schl);

Ada.Text\_IO.Put\_Line(" : " & H\_Ptr.Info);

end if;

end loop;

if Weiter then

-- Hilfsvariable auf den echten Verweis weiterschalten

H\_Ptr := H\_Ptr.Rechts\_Ptr;

end if;

end loop;

else

null;

end if;

end Ausgabe;

procedure Durchlaufen (Wurzel\_Ptr : in BAUM\_REF) is

package Schluessel\_IO is new Ada.Text\_IO.Integer\_IO (SCHLUESSEL\_T);

begin

if Wurzel\_Ptr /= null then

if Wurzel\_Ptr.Links\_Echt then

Durchlaufen (Wurzel\_Ptr.Links\_Ptr);

end if;

Schluessel\_IO.Put (Wurzel\_Ptr.Schl);

Ada.Text\_IO.Put\_Line (" : " & Wurzel\_Ptr.Info);

if Wurzel\_Ptr.Rechts\_Echt then

Durchlaufen (Wurzel\_Ptr.Rechts\_Ptr);

end if;

end if;

end Durchlaufen;

procedure AusgabeR is

begin

Durchlaufen (B);

end AusgabeR;

end Nadelbaum;

Praktikum 2 – Dreifachverkettung

Paketbody

package body Dvl is

type ELEM\_TR;

type ELEM\_REF is access ELEM\_TR;

type ELEM\_TR is record

Schl : SCHLUESSEL;

Info : INFORMATION;

Lifo\_Ptr : ELEM\_REF;

Fifo\_Ptr : ELEM\_REF;

Alpha\_Ptr : ELEM\_REF;

end record;

S\_FIFO\_Ptr : ELEM\_REF := null;

S\_LIFO\_Ptr : ELEM\_REF := null;

S\_ALPHA\_Ptr : ELEM\_REF := null;

procedure Aufnahme (S : in SCHLUESSEL; I : in INFORMATION) is

H\_Ptr : ELEM\_REF := null;

Last\_Ptr : ELEM\_REF := null;

Z\_Ptr : ELEM\_REF := null;

begin

Z\_Ptr := new ELEM\_TR'(S, I, Lifo\_Ptr => null,   
 Fifo\_Ptr => null, Alpha\_Ptr => null);

if S\_LIFO\_Ptr = null then

S\_LIFO\_Ptr := Z;

else

Z\_Ptr.Lifo\_Ptr := S\_LIFO\_Ptr;

S\_LIFO\_Ptr := Z\_Ptr; -- eingehaengt in LIFO

end if;

if S\_FIFO\_Ptr = null then

S\_FIFO\_Ptr := Z\_Ptr;

else

H\_Ptr := S\_FIFO\_Ptr;

while H\_Ptr.Fifo\_Ptr /= null loop

H\_Ptr := H\_Ptr.Fifo\_Ptr;

end loop;

H\_Ptr.Fifo\_Ptr := Z\_Ptr; -- eingehaengt in FIFO

end if;

if S\_ALPHA\_Ptr = null then

S\_ALPHA\_Ptr := Z\_Ptr;

else

H\_Ptr := S\_ALPHA\_Ptr;

Last\_Ptr := S\_ALPHA\_Ptr;

while not (H\_Ptr = null) and then H\_Ptr.Schl <= Z\_Ptr.Schl loop

Last\_Ptr := H\_Ptr;

H \_Ptr := H.Alpha\_Ptr;

end loop;

if H\_Ptr = null then

Last\_Ptr.Alpha\_Ptr := Z\_Ptr; -- eingehaengt in ALPHA am Ende

else

if H\_Ptr = S\_ALPHA\_Ptr then

Z\_Ptr.Alpha\_Ptr := S\_ALPHA\_Ptr;

S\_ALPHA\_Ptr := Z\_Ptr; -- eingehaengt in ALPHA am Anfang

else

Last\_Ptr.Alpha\_Ptr := Z\_Ptr;

Z\_Ptr.Alpha\_Ptr := H\_Ptr; -- eingehaengt in ALPHA in der Mitte

end if;

end if;

end if;

end Aufnahme;

procedure Entnahme

(V : in VERFAHREN\_T;

S : out SCHLUESSEL;

I : out INFORMATION)

is

begin

case V is

when FIFO =>

if S\_FIFO\_Ptr = null then

raise Zugriffsfehler;

else

S := S\_FIFO\_Ptr.Schl;

I := S\_FIFO\_Ptr.Info;

S\_FIFO\_Ptr := S\_FIFO\_Ptr.Fifo\_Ptr;

end if;

when LIFO =>

if S\_LIFO\_Ptr = null then

raise Zugriffsfehler;

else

S := S\_LIFO\_Ptr.Schl;

I := S\_LIFO\_Ptr.Info;

S\_LIFO\_Ptr := S\_LIFO\_Ptr.Lifo\_Ptr;

end if;

when ALPHA =>

if S\_ALPHA\_Ptr = null then

raise Zugriffsfehler;

else

S := S\_ALPHA\_Ptr.Schl;

I := S\_ALPHA\_Ptr.Info;

S\_ALPHA\_Ptr := S\_ALPHA\_Ptr.Alpha\_Ptr;

end if;

end case;

end Entnahme;

procedure Loeschen (S : in SCHLUESSEL) is

Vor\_Ptr, Z\_Ptr : ELEM\_REF := S\_ALPHA\_Ptr;

Gefunden : BOOLEAN := False;

Begin

while Z\_Ptr /= null loop --aus ALPHA

if Z\_Ptr.Schl = S then

Gefunden := True;

if Z\_Ptr = S\_ALPHA\_Ptr then

S\_ALPHA\_Ptr := S\_ALPHA\_Ptr.Alpha\_Ptr;

Vor\_Ptr := S\_ALPHA\_Ptr;

Z\_Ptr := S\_ALPHA\_Ptr;

else

Vor\_Ptr.Alpha\_Ptr := Z.Alpha\_Ptr;

Z\_Ptr := Z\_Ptr.Alpha\_Ptr;

end if;

else

Vor\_Ptr := Z\_Ptr;

Z\_Ptr := Z\_Ptr.Alpha\_Ptr;

end if;

end loop;

Vor\_Ptr := S\_FIFO\_Ptr; --aus FIFO

Z\_Ptr := S\_FIFO\_Ptr;

while Z\_Ptr /= null loop

if Z\_Ptr.Schl = S then

Gefunden := True;

if Z\_Ptr = S\_FIFO\_Ptr then

S\_FIFO\_Ptr := S\_FIFO\_Ptr.Fifo\_Ptr;

Vor\_Ptr := S\_FIFO\_Ptr;

Z\_Ptr := S\_FIFO\_Ptr;

else

Vor\_Ptr.Fifo\_Ptr := Z\_Ptr.Fifo\_Ptr;

Z\_Ptr := Z\_Ptr.Fifo\_Ptr;

end if;

else

Vor\_Ptr := Z\_Ptr;

Z\_Ptr := Z\_Ptr.Fifo\_Ptr;

end if;

end loop;

Vor\_Ptr := S\_LIFO\_Ptr; --aus LIFO

Z\_Ptr := S\_LIFO\_Ptr;

while Z\_Ptr /= null loop

if Z\_Ptr.Schl = S then

Gefunden := True;

if Z\_Ptr = S\_LIFO\_Ptr then

S\_LIFO\_Ptr := S\_LIFO\_Ptr.Lifo\_Ptr;

Vor\_Ptr := S\_LIFO\_Ptr;

Z\_Ptr := S\_LIFO\_Ptr;

else

Vor\_Ptr.Lifo\_Ptr := Z\_Ptr.Lifo\_Ptr;

Z\_Ptr := Z\_Ptr.Lifo\_Ptr;

end if;

else

Vor\_Ptr := Z\_Ptr;

Z\_Ptr := Z\_Ptr.Lifo\_Ptr;

end if;

end loop;

if not Gefunden then

raise Zugriffsfehler;

end if;

end Loeschen;

function Leer return BOOLEAN is

begin

return S\_FIFO\_Ptr = null and S\_LIFO\_Ptr = null and S\_ALPHA\_Ptr = null;

end Leer;

procedure Fuer\_Einen\_Schluessel (V : in VERFAHREN\_T; S : in SCHLUESSEL) is

Z\_Ptr : ELEM\_REF;

Gefunden : BOOLEAN := False;

begin

case V is

when FIFO =>

Z\_Ptr := S\_FIFO\_Ptr;

when LIFO =>

Z\_Ptr := S\_LIFO\_Ptr;

when ALPHA =>

Z\_Ptr := S\_ALPHA\_Ptr;

end case;

while Z\_Ptr /= null loop

if Z\_Ptr.Schl = S then

Gefunden := True;

Aktion (Z\_Ptr.Info);

end if;

case V is

when FIFO =>

Z\_Ptr := Z\_Ptr.Fifo\_Ptr;

when LIFO =>

Z\_Ptr := Z\_Ptr.Lifo\_Ptr;

when ALPHA =>

Z\_Ptr := Z\_Ptr.Alpha\_Ptr;

end case;

end loop;

if not Gefunden then

raise Zugriffsfehler;

end if;

end Fuer\_Einen\_Schluessel;

procedure Fuer\_Alle\_Elemente (V : in VERFAHREN\_T) is

Z\_Ptr : ELEM\_REF;

begin

case V is

when FIFO =>

Z\_Ptr := S\_FIFO\_Ptr;

while Z\_Ptr /= null loop

Aktion (Z\_Ptr.Schl, Z\_Ptr.Info);

Z\_Ptr := Z\_Ptr.Fifo\_Ptr;

end loop;

when LIFO =>

Z\_Ptr := S\_LIFO\_Ptr;

while Z\_Ptr /= null loop

Aktion (Z\_Ptr.Schl, Z\_Ptr.Info);

Z\_Ptr := Z\_Ptr.Lifo\_Ptr;

end loop;

when ALPHA =>

Z\_Ptr := S\_ALPHA\_Ptr;

while Z\_Ptr /= null loop

Aktion (Z\_Ptr.Schl, Z\_Ptr.Info);

Z\_Ptr := Z\_Ptr.Alpha\_Ptr;

end loop;

end case;

end Fuer\_Alle\_Elemente;

end Dvl;