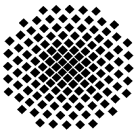
Universität Stuttgart
Fakultät 5, Institut FMI
EfidI 1, Professor Claus



WS 2002/2003
Bertele, Lewandowski
9. Januar 2003

Aufgabenblatt 10

Musterlösung

Aufgabe 1 Mengen I (schriftlich, mittel) 5 Punkte

In dieser Aufgabe üben Sie sowohl die Erzeugung eines ADT, sowie den Umgang mit Listen.

Implementieren Sie einen abstrakten Datentyp IntSet, der eine Menge über dem gesamten Wertebereich des Datentyps Integer realisiert und auf dem folgende Operationen zur Verfügung gestellt werden:

- `Create(S: in out IntSet);` Erzeugt eine leere Menge.
- `Copy(S: IntSet) return IntSet;` Liefert eine Kopie der Menge S.
- `Incl(S: in out IntSet; E: Integer);` Nimmt das Element E in die Menge S auf.
- `Excl(S: in out IntSet; E: Integer);` Entfernt das Element E aus der Menge S.
- `IsIn(S: IntSet; E: Integer) return Boolean;` Liefert True, falls das Element E in S vorkommt, sonst False.

Verwenden Sie zur Repräsentation einer Menge eine verkettete Liste, deren Elemente der Größe nach sortiert sind; diese Eigenschaft soll nach jeder Operation erhalten bleiben.

Aufgabe 2 Mengen II (schriftlich, schwer) 8 Punkte

In dieser Aufgabe geht es um effiziente Programmierung.

Implementieren Sie weitere Operationen auf Ihrem abstrakten Datentyp IntSet aus „Mengen I“:

- `Difference(S1, S2: IntSet) return IntSet;`
  Liefert die Mengendifferenz $S1 - S2 = \{x \mid x \in S1 \wedge x \notin S2\}$.
- `Intersection(S1, S2: IntSet) return IntSet;`
  Liefert die Schnittmenge $S1 \cap S2$.
- `SymmetricDifference(S1, S2: IntSet) return IntSet;`
  Liefert die symmetrische Differenz $\{x \mid (x \in S1 \wedge x \notin S2) \vee (x \in S2 \wedge x \notin S1)\}$.
- `Union(S1, S2: IntSet) return IntSet;`
  Liefert die Vereinigungsmenge $S1 \cup S2$.

Ihre Lösung soll die Sortierung der Listenrepräsentation in aufsteigender Reihenfolge erhalten und ausnutzen. Sie können zur Implementierung die Prozedur Copy aus „Mengen I“ verwenden, aber nicht die Prozeduren Incl, Excl und In. (Dadurch sollten Sie lineare Komplexität erreichen.)

```
 1. -- lists.ads
 2. -- Header fuer Blatt 12, generischer Listentyp
 3. -- Autor: Gunnar Hilling
 4. -- Datum: 17.01.2001
 5.
 6. generic
 7.        type T is private;
 8.
 9. package lists is
10.        type liste is private;
11.
12.        function newlist return liste;
13.        function cons(x: T; l: liste) return liste;
14.        function isempty(l: liste) return boolean;
```

```
15.        function head(l: liste) return T;
16.        function tail(l: liste) return liste;
17.
18. private
19.        type knoten;
20.        type liste is access knoten;
21.        type knoten is record
22.                        wert: T;
23.                        nach: liste;
24.                end record;
25.
26. end lists;
```

```
 1. -- lists.adb
 2. -- Implementierung fuer Blatt 12, generischer Listentyp
 3. -- Autor: Gunnar Hilling
 4. -- Datum: 17.01.2001
 5.
 6. package body lists is
 7.
 8.
 9. function newlist return liste is
10. begin
11.        return NULL;
12. end newlist;
13.
14. function cons(x: T; l: liste) return liste is
15. begin
16.        return new knoten'(wert => x, nach => l);
17. end cons;
18.
19. function isempty(l: liste) return boolean is
20. begin
21.        return l = null;
22. end isempty;
23.
24. function head(l: liste) return T is
25. begin
26.        return l.wert;
27. end head;
28.
29. function tail(l: liste) return liste is
30. begin
31.        return l.nach;
32. end tail;
33.
34. end lists;
```

```
 1. -- sets.ads
 2. -- Header fuer Blatt 12, generischer Mengentyp
 3. -- Autor: Gunnar Hilling
 4. -- Datum: 17.01.2001
 5.
 6. with lists;
 7.
 8. generic
 9.        type Element is private;
10.        with function "<"(u,v: Element) return boolean;
11.        with procedure ElPut(u: Element);
12.
13. package sets is
14.        type set is private;
15.
16.        procedure Put(s: In set);
17.
18. -- Aufgabe 1:
19.        function newset return set;
20.        function copy(s: In set) return set;
21.        function incl(s: In set; e: In Element) return set;
22.        function excl(s: In set; e: In Element) return set;
23.        function contains(s: In set; e: In Element) return boolean;
24.
25. -- Aufgabe 2:
26.        function difference(s1, s2: In set) return set;
27.        function intersection(s1, s2: In set) return set;
28.        function union(s1, s2: In set) return set;
29.
30. private
31.        package elem_list is new lists(Element); use elem_list;
```

```
32.        type set is new liste;
33.
34. end sets;
```

```
 1. -- sets.ads
 2. -- Header fuer Blatt 11, generischer Mengentyp
 3. -- Autor: Gunnar Hilling / Joergen Bertele
 4. -- Datum: 22.01.2002
 5.
 6.
 7. with Ada.Text_IO;
 8. use Ada.Text_IO;
 9.
10. package body sets is
11.        procedure Put(s: In set) is
12.        begin
13.                if isempty(s) then
14.                        return;
15.                else
16.                        ElPut(head(s));
17.                        New_Line;
18.                        Put(tail(s));
19.                end if;
20.        end Put;
21.
22.        function newset return set is
23.        begin
24.                return newlist;
25.        end newset;
26.
27.        function copy(s: In set) return set is
28.        begin
29.                if isempty(s) then
30.                        return newset;
31.                else
32.                        return cons(head(s), copy(tail(s)));
33.                end if;
34.        end copy;
35.
36.        function incl(s: In set; e: In element) return set is
37.        begin
38.                if isempty(s) then
39.                        return cons(e, newset);
40.                elsif head(s) = e then -- weiterkopieren
41.                        return copy(s);
42.                elsif head(s) < e then -- vorher einfuegen!
43.                        return cons(e, copy(s));
44.                else
45.                        return cons(head(s), incl(tail(s), e));
46.                end if;
47.        end incl;
48.
49.        function excl(s: In set; e: In element) return set is
50.        begin
51.                if isempty(s) then
52.                        return newset;
53.                elsif head(s) = e then -- dieses ueberspringen!
54.                        return copy(tail(s));
55.                else
56.                        return cons(head(s), excl(tail(s), e));
57.                end if;
58.        end excl;
59.
60.        function contains(s: In set; e: In element) return boolean is
61.        begin
62.                if isempty(s) or else head(s) < e then
63.                        return false;
64.                elsif head(s) = e then
65.                        return true;
66.                else
67.                        return contains(tail(s), e);
68.                end if;
69.        end contains;
70.
71.        function difference(s1, s2: In set) return set is
72.        begin
73.                if isempty(s1) then
74.                        return newset;
```

```
75.                elsif isempty(s2) then
76.                       return copy(s1);
77.                elsif head(s1) = head(s2) then
78.                       return difference(tail(s1), tail(s2));
79.                elsif head(s1) < head(s2) then
80.                       return difference(s1, tail(s2));
81.                else -- head(s1) > head(s2)
82.                       return cons(head(s1), difference(tail(s1),s2));
83.                end if;
84.        end difference;
85.
86.        function intersection(s1, s2: In set) return set is
87.        begin
88.                if isempty(s1) or isempty(s2) then
89.                       return newset;
90.                elsif head(s1) = head(s2) then
91.                       return cons(head(s1), intersection(tail(s1), tail(s2)));
92.                elsif head(s1) < head(s2) then
93.                       return intersection(s1, tail(s2));
94.                else   -- head(s1) > head(s2)
95.                       return intersection(tail(s1), s2);
96.                end if;
97.        end intersection;
98.
99.        function union(s1, s2: In set) return set is
100.       begin
101.               if isempty(s1) then
102.                      return s2;
103.               elsif isempty(s2) then
104.                      return s1;
105.               elsif head(s1) = head(s2) then
106.                      return cons(head(s1), union(tail(s1), tail(s2)));
107.               elsif head(s1) < head(s2) then
108.                      return cons(head(s2), union(s1, tail(s2)));
109.               else   -- head(s1) > head(s2)
110.                      return cons(head(s1), union(s2, tail(s1)));
111.               end if;
112.       end union;
113.
114.       function symdifference(s1, s2: In set) return set is
115.       begin
116.           return union(difference(s1,s2),difference(s2,s1));
117.       end symdifference;
118.
119. end sets;
```

Aufgabe 3 Exceptions (Votieraufgabe, mittel) 7 Punkte

Diese Aufgabe fasst das Arbeiten mit Arrays und Exceptions zusammen. Außerdem sollen Sie sich in dieser Aufgabe auch Gedanken machen, wie Programme getestet werden können.

Schreiben Sie ein Paket „Felder“ (ADS und ADB) zur Verarbeitung von unbegrenzten Zeichenketten. Definieren Sie dazu einen Typ, der ein Array aus Ustrings darstellt. Schreiben Sie dann folgende Routinen:

- `function erzeuge(Start, Ende: Integer) return Feldtyp;`
  Liefert ein Array mit den angegebenen Grenzen zurück. Die Elemente sollten leer sein.
- `function erzeuge(Anzahl: Integer) return Feldtyp;`
  Wie die vorige Funktion, aber hier wird ein Array mit der angegebenen Anzahl von Elementen zurückgeliefert.
- `function hole(Feld: Feldtyp; Position: Integer) return Ustring;`
  Liefert den an der Stelle „Position“ gespeicherten String zurück. Ist der String leer, wird die Exception „Leer“ geworfen.
- `procedure setze(Feld: in out Feldtyp; Position: in Integer; Wert: Ustring);`
  Trägt an der Stelle „Position“ den String „Wert“ ein. Ist an dieser Stelle bereits an String eingetragen, wird die Exception „Voll“ geworfen.
- `function frei(Feld: feldtyp; Position: Integer) return Boolean;`
  Überprüft, ob die Stelle „Position“ noch frei ist.
- `procedure loesche(Feld: in out Feldtyp; Position: in Integer);`
  Löscht die entsprechende Stelle im übergebenen Feld. Ist die Stelle bereits leer, wird die Exception „leer“ geworfen.
- `function zaehle(Feld: Feldtyp) return Natural;`
  Zählt die Anzahl der im Feld enthaltenen, belegten Strings.

Schreiben Sie zusätzlich noch ein Hauptprogramm, das ihr Programmpaket testet. Das heißt es sollen sowohl funktionierende Fälle, als auch Fehlerfälle überprüft werden. Das heißt, sie sollten die Exceptions im Hauptprogramm abfangen. Bedenken Sie, dass außer den von Ihnen erzeugten Exceptions auch noch die Ada95-Exceptions geworfen werden können.

```
 1. -- EfidI1 WS01/02
 2. -- Blatt 8 Aufgabe 4
 3. -- Autor: Jörgen Bertele
 4.
 5. with Ustrings;
 6. use Ustrings;
 7.
 8. package Felder is
 9.    voll, leer: exception;
10.    type Feldtyp is array (Integer range <>) of Ustring;
11.    function Erzeuge (Start, Ende: Integer) return Feldtyp;
12.    function Erzeuge (Anzahl: Integer)      return Feldtyp;
13.    function Hole (Feld: Feldtyp; Position: Integer) return Ustring;
14.    procedure Setze (Feld: in out Feldtyp; Position: in Integer; Wert: Ustring);
15.    procedure Loesche (Feld: in out Feldtyp; Position: in Integer);
16.    function Frei (Feld: Feldtyp; Position: Integer) return Boolean;
17.    function Zaehle (Feld: Feldtyp ) return Natural;
18. end;
 1. -- EfidI1 WS01/02
 2. -- Blatt 8 Aufgabe 4
 3. -- Autor: Jörgen Bertele
 4.
 5. package body Felder is
 6.
 7.    -------------
 8.    -- Erzeuge --
 9.    -------------
10.
11.    function Erzeuge (
12.          Start,
13.          Ende  : Integer )
14.      return Feldtyp is
15.       Feld : Feldtyp (Start .. Ende) := (others => U (""));
```

```
16.    begin
17.       return Feld;
18.    end Erzeuge;
19.
20.    -------------
21.    -- Erzeuge --
22.    -------------
23.
24.    function Erzeuge (
25.          Anzahl : Integer )
26.      return Feldtyp is
27.       Feld : Feldtyp (0 .. Anzahl - 1) := (others => U (""));
28.    begin
29.       return Feld;
30.    end Erzeuge;
31.
32.    ----------
33.    -- Frei --
34.    ----------
35.
36.    function Frei (
37.          Feld     : Feldtyp;
38.          Position : Integer  )
39.      return Boolean is
40.    begin
41.       return S(Feld(Position))="";
42.    end Frei;
43.
44.    ----------
45.    -- Hole --
46.    ----------
47.
48.    function Hole (
49.          Feld     : Feldtyp;
50.          Position : Integer  )
51.      return Ustring is
52.    begin
53.       if Frei(Feld,Position) then
54.          raise Leer;
55.       else
56.          return Feld(Position);
57.       end if;
58.    end Hole;
59.
60.    -------------
61.    -- Loesche --
62.    -------------
63.
64.    procedure Loesche (
65.          Feld     : in out Feldtyp;
66.          Position : in     Integer  ) is
67.    begin
68.       if Frei(Feld,Position) then
69.          raise Leer;
70.       else
71.          Feld(Position):=U("");
72.       end if;
73.    end Loesche;
74.
75.    -----------
76.    -- Setze --
77.    -----------
78.
79.    procedure Setze (
80.          Feld     : in out Feldtyp;
81.          Position : in     Integer;
82.          Wert     :        Ustring  ) is
83.    begin
84.       if Frei(Feld,Position) then
85.          Feld(Position):=Wert;
86.       else
87.          raise Voll;
88.       end if;
89.    end Setze;
90.
91.    ------------
92.    -- Zaehle --
```

```
   ------------

   function Zaehle (
         Feld : Feldtyp )
     return Natural is
      Zahl : Natural := 0;
   begin
      for I in Feld'range loop
         if not Frei(Feld,I) then
            Zahl:=Zahl+1;
         end if;
      end loop;
      return Zahl;
   end Zaehle;

end Felder;
```