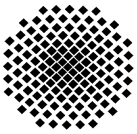
Universität Stuttgart
Fakultät 5, Institut FMI
EfidI 1, Professor Claus



WS 2002/2003
Bertele, Lewandowski
14. Januar 2003

# Aufgabenblatt 9

Musterlösung

**Aufgabe 1 Große Zahlen III (schriftlich, mittel) 5 Punkte**

*In der Aufgabe Große Zahlen haben Sie einen eigenen Datentyp implementiert. Dabei wurde jedoch als Basis auf Ada.Strings.Unbounded.Unbounded_String zurückgegriffen. Das werden wir jetzt ändern. Sie können diese Aufgabe unabhängig von Große Zahlen I und II lösen.*

Nehmen Sie das von Ihnen erstellte Paket *Gzahl* und verändern Sie sowohl das ADS, als auch das ADB, so dass das Paket nicht mehr auf Zeichenketten, sondern aus einer Liste aus Zeichen besteht.

```
 1. -- EfidI1 WS02/03
 2. -- Blatt 9 Aufgabe 1
 3. -- Autor: Jörgen Bertele
 4.
 5. package Gzahl is
 6.    type Grosse_Zahl is private;
 7.
 8.    -- liest eine beliebig grosse Ganzzahl vom Benutzer ein.
 9.    procedure Get(X: out Grosse_Zahl);
10.
11.    -- gibt eine beliebig grosse Ganzzahl auf dem Bildschirm aus.
12.    procedure Put(X: in Grosse_Zahl);
13.
14.    -- die vier Grundrechenarten mit den Ganzzahlen.
15.    function "+"(Left, Right: Grosse_Zahl) return Grosse_Zahl;
16.    function "-"(Left, Right: Grosse_Zahl) return Grosse_Zahl;
17.    function "*"(Left, Right: Grosse_Zahl) return Grosse_Zahl;
18.    function "/"(Left, Right: Grosse_Zahl) return Grosse_Zahl;
19.
20.    -- zusätzliche Erweiterungen
21.    function "abs"(Right: Grosse_Zahl) return Grosse_Zahl;
22.    function "rem"(Left, Right: Grosse_Zahl) return Grosse_Zahl;
23.    function "mod"(Left, Right: Grosse_Zahl) return Grosse_Zahl;
24.    function "="(Left, Right: Grosse_Zahl) return boolean;
25.    function "<"(Left, Right: Grosse_Zahl) return Boolean;
26.    function ">"(Left, Right: Grosse_Zahl) return Boolean;
27.    function "<="(Left, Right: Grosse_Zahl) return Boolean;
28.    function ">="(Left, Right: Grosse_Zahl) return Boolean;
29.
30.    private
31.    type grosse_Zahl_Zeiger is access all grosse_Zahl;
32.    type Grosse_Zahl is record
33.       Wert:Character;
34.       Zeiger:Grosse_Zahl_Zeiger;
35.    end record;
36.
37. end Gzahl;
```

```
 1. -- EfidI1 WS02/03
 2. -- Blatt 9 Aufgabe 1
 3. -- Autor: Jörgen Bertele
 4.
 5. with Ada.Text_Io; use Ada.Text_Io;
 6.
 7. package body Gzahl is
 8.    -- Allgemeine Konstanten
 9.    Gz_1: constant Grosse_Zahl:=('1',null);
10.    Gz_0: constant Grosse_Zahl:=('0',null);
11.
12.    -------------------------
13.    -- Get (verwendet "+") --
```

```
14.    -------------------------
15.    procedure Get (X: out Grosse_Zahl) is
16.       C: Character;
17.       W:Integer;
18.       V,T: Grosse_Zahl_Zeiger:=null;
19.       A:Grosse_Zahl;
20.    begin
21.       A.Zeiger:=null;
22.       Get(C);
23.       W:=Character'Pos(C)-Character'Pos('0');
24.       if (C/='-') and (W > 9 or W < 0) then
25.          raise Data_Error;
26.       else
27.          A.Wert:=C;
28.       end if;
29.       while not End_Of_Line loop
30.          Get(C);
31.          T:=new Grosse_Zahl'(C,null);
32.          if A.Zeiger=null then
33.             A.Zeiger:=T;
34.          else
35.             V.Zeiger:=T;
36.          end if;
37.          V:=T;
38.       end loop;
39.       Skip_Line;
40.       X:=A;
41.    end Get;
42.
43.    ---------
44.    -- Put --
45.    ---------
46.    procedure Put (X: in Grosse_Zahl) is
47.       T:Grosse_Zahl_Zeiger;
48.    begin
49.       Put(X.Wert);
50.       T:=X.Zeiger;
51.       while T /=null loop
52.          Put(T.Wert);
53.          T:=T.Zeiger;
54.       end loop;
55.    end Put;
56.
57.    --------------------
58.    -- intern_signum --
59.    --------------------
60.    procedure Intern_Vz(Zahl: in out Grosse_Zahl; Vz: out Boolean) is
61.    begin
62.       if Zahl.Wert='-' then
63.          Vz:=True;
64.          Zahl:=Zahl.Zeiger.all;
65.       else
66.          Vz:=False;
67.       end if;
68.    end Intern_Vz;
69.
70.    ---------------
71.    -- Element --
72.    ---------------
73.    function Element(Zahl:Grosse_Zahl;Position: Positive) return Character is
74.       T: Grosse_Zahl_Zeiger:=Zahl.Zeiger;
75.    begin
76.       if Position=1 then
77.          return Zahl.Wert;
78.       else
79.          for I in 3..Position loop
80.             T:=T.Zeiger;
81.          end loop;
82.          return T.Wert;
83.       end if;
84.    end Element;
85.
86.    ------------
87.    -- length --
88.    ------------
89.    function Length(Zahl:Grosse_Zahl) return Natural is
90.    begin
```

```
91.       if Zahl.Zeiger=null then
92.          return 1;
93.       else
94.          return 1+Length(Zahl.Zeiger.all);
95.       end if;
96.    end Length;
97.
98.    -------------------------------
99.    -- "+" (verwendet intern_vz) --
100.   -------------------------------
101.   function "+" (Left, Right: Grosse_Zahl) return Grosse_Zahl is
102.      -- Vorzeichen
103.      Ln,Rn:Boolean:=False;
104.      -- Unbegrenzte Zahlen
105.      L:Grosse_Zahl:=Left;
106.      R:Grosse_Zahl:=Right;
107.      T,E:Grosse_Zahl:=(' ',null);
108.      Z:Grosse_Zahl_Zeiger;
109.      -- Positionsvariablen
110.      Lz,Rz:Integer:=1;
111.      -- Wertvariablen
112.      Lw,Rw,Ew,Uw:Integer:=0;
113.      -- Hilfsfunktionen
114.      function Wert(C:Character) return Natural is
115.         W: Integer;
116.      begin
117.         W:=Character'Pos(C)-Character'Pos('0');
118.         if W > 9 or W < 0 then
119.            raise Data_Error;
120.         end if;
121.         return W;
122.      end Wert;
123.      function Zeichen(N:Natural) return Character is
124.      begin
125.         return Character'Val(Character'Pos('0') + N);
126.      end Zeichen;
127.   begin
128.      -- Vorzeichen ermitteln
129.      Intern_Vz(L,Ln);
130.      Intern_Vz(R,Rn);
131.      -- Vorbereitung
132.      if (Ln xor Rn) and Ln then
133.         -- vertausche so dass immer rechts negativ ist.
134.         T:=L; L:=R; R:=T;  -- Variablentausch
135.         Ln:=False;
136.         Rn:=True;
137.      end if;
138.      Lz:=Length(L);
139.      Rz:=Length(R);
140.      -- Berechnung
141.      while Lz>=0 or Rz>=0 loop
142.         -- Wertermittlung
143.         if Lz>0 then
144.            Lw:=Wert(Element(L,Lz));
145.         else
146.            Lw:=0;
147.         end if;
148.         if Rz>0 then
149.            Rw:=Wert(Element(R,Rz));
150.         else
151.            Rw:=0;
152.         end if;
153.         -- Wertberechnung
154.         if Ln xor Rn then
155.            -- Subtraktion links-rechts
156.            Ew:=Lw-(Rw+Uw);
157.            if Ew<0 then
158.               Ew:=Ew+10;
159.               Uw:=1;
160.            else
161.               Uw:=0;
162.            end if;
163.         else
164.            -- Addition links+rechts;
165.            Ew:=Lw+Rw+Uw;
166.            if Ew>=10 then
167.               Ew:=Ew-10;
```

```
168.                Uw:=1;
169.             else
170.                Uw:=0;
171.             end if;
172.          end if;
173.          if E.Wert=' ' then
174.             E.Wert:=Zeichen(Ew);
175.          else
176.             Z:=new Grosse_Zahl'(E.Wert,E.Zeiger);
177.             E:=(Zeichen(Ew),Z);
178.          end if;
179.          Lz:=Lz-1;
180.          Rz:=Rz-1;
181.       end loop;
182.       -- Führende Nullen streichen
183.       while Element(E,1)='0' and Length(E)>1 loop
184.          E:=E.Zeiger.all;
185.       end loop;
186.       if Ln and Rn then
187.          -- Ergebnis ist negativ, da -links-rechts<0
188.          Z:=new Grosse_Zahl'(E.Wert,E.Zeiger);
189.          E:=('-',Z);
190.       elsif Ln xor Rn then
191.          -- e_us kann negativ sein, da links-rechts?<0
192.          -- Probe machen
193.          T:=E+R;
194.          if T/=L then
195.             -- e negativ:
196.             E:=R-L;
197.             Z:=new Grosse_Zahl'(E.Wert,E.Zeiger);
198.             E:=('-',Z);
199.          end if;
200.       end if;
201.       return E;
202.    end "+";
203.
204.    -------------------------
205.    -- "-" (verwendet "+") --
206.    -------------------------
207.    function "-" (Left, Right: Grosse_Zahl) return Grosse_Zahl is
208.       Nz:Grosse_Zahl_Zeiger;
209.       N:Grosse_Zahl;
210.    begin
211.       if Element(Right,1)='-' then
212.          return Left+Right.Zeiger.all;
213.       else
214.          Nz:=new Grosse_Zahl'(Right.Wert,Right.Zeiger);
215.          N:=('-',Nz);
216.          return Left+N;
217.       end if;
218.    end "-";
219.
220.    ---------------------------------------
221.    -- "*" (verwendet "+","-", intern_vz) --
222.    ---------------------------------------
223.    function "*" (Left, Right: Grosse_Zahl) return Grosse_Zahl is
224.       -- Vorzeichen
225.       Ln,Rn:Boolean:=False;
226.       -- Unbegrenzte Zahlen
227.       T,E: Grosse_Zahl:=Gz_0;
228.       L:Grosse_Zahl:=Left;
229.       R:Grosse_Zahl:=Right;
230.       Z:Grosse_Zahl_Zeiger;
231.    begin
232.       -- Optimierung (beschleunigt Berechnung manchmal erheblich)
233.       if Length(L)>Length(R) then
234.          T:=L; L:=R; R:=T;  -- Variablentausch
235.       end if;
236.       -- Vorzeichen ermitteln
237.       Intern_Vz(L,Ln);
238.       Intern_Vz(R,Rn);
239.       -- Multiplikation
240.       while L/=Gz_0 loop
241.          E:=R+E;
242.          L:=L-Gz_1;
243.       end loop;
244.       -- Ergebnis ist negativ
```

```
245.       if Ln xor Rn then
246.          Z:=new Grosse_Zahl'(E.Wert,E.Zeiger);
247.          E:=('-',Z);
248.       end if;
249.       return E;
250.    end "*";
251.
252.    ------------------------------------------------
253.    -- intern_divison (verwendet "-","+", intern_vz) --
254.    ------------------------------------------------
255.    procedure Intern_Division(Left, Right: in Grosse_Zahl; Ergebnis, Rest: out
Grosse_Zahl) is
256.       -- Vorzeichen
257.       Ln,Rn:Boolean:=False;
258.       -- Unbegrenzte Zahlen
259.       E: Grosse_Zahl:=Gz_0;
260.       L:Grosse_Zahl:=Left;
261.       R:Grosse_Zahl:=Right;
262.       Z:Grosse_Zahl_Zeiger;
263.    begin
264.       -- Vorzeichen ermitteln
265.       Intern_Vz(L,Ln);
266.       Intern_Vz(R,Rn);
267.       -- Division
268.       while Element((L-R),1)/='-' loop
269.          L:=L-R;
270.          E:=Gz_1+E;
271.       end loop;
272.       -- Ergebnis ist negativ
273.       if Ln xor Rn then
274.          Z:=new Grosse_Zahl'(E.Wert,E.Zeiger);
275.          E:=('-',Z);
276.       end if;
277.       Ergebnis:=E;
278.       Rest:=L;
279.    end Intern_Division;
280.
281.    ---------------------------------------
282.    -- "/" (verwendet "intern_division") --
283.    ---------------------------------------
284.    function "/" (Left, Right: Grosse_Zahl) return Grosse_Zahl is
285.       E,R: Grosse_Zahl;
286.    begin
287.       Intern_Division(Left,Right,E,R);
288.       return E;
289.    end "/";
290.
291.    ---------------------------------------
292.    -- "mod" (verwendet intern_division) --
293.    ---------------------------------------
294.    function "mod" (Left, Right: Grosse_Zahl) return Grosse_Zahl is
295.       E,R: Grosse_Zahl;
296.       Z:Grosse_Zahl_Zeiger;
297.    begin
298.       Intern_Division(Left,Right,E,R);
299.       if Right < Gz_0 then
300.          Z:=new Grosse_Zahl'(R.Wert,R.Zeiger);
301.          R:=('-',Z);
302.       end if;
303.       return R;
304.    end "mod";
305.
306.    ---------------------------------------
307.    -- "rem" (verwendet intern_division) --
308.    ---------------------------------------
309.    function "rem" (Left, Right: Grosse_Zahl) return Grosse_Zahl is
310.       E,R: Grosse_Zahl;
311.       Z:Grosse_Zahl_Zeiger;
312.    begin
313.       Intern_Division(Left,Right,E,R);
314.       if Left < Gz_0 then
315.          Z:=new Grosse_Zahl'(R.Wert,R.Zeiger);
316.          R:=('-',Z);
317.       end if;
318.       return R;
319.    end "rem";
320.
```

```
321.    ---------------------------------
322.    -- "abs" (verwendet intern_vz) --
323.    ---------------------------------
324.    function "abs" (Right: Grosse_Zahl) return Grosse_Zahl is
325.       R:Grosse_Zahl:=Right;
326.       Rn: Boolean;
327.    begin
328.       -- Vorzeichen ermitteln
329.       Intern_Vz(R,Rn);
330.       return R;
331.    end "abs";
332.
333.    ---------
334.    -- "=" --
335.    ---------
336.    function "=" (Left, Right: Grosse_Zahl) return Boolean is
337.       L: Grosse_Zahl:=Left;
338.       R: Grosse_Zahl:=Right;
339.    begin
340.       -- Minimale Fehlerbehandlung
341.       if L.Zeiger=null and R.Zeiger=null then
342.          return L.Wert=R.Wert;
343.       elsif L.Zeiger=null xor R.Zeiger=null then
344.          return False;
345.       else
346.          return Left.Zeiger.all=Right.Zeiger.all;
347.       end if;
348.    end "=";
349.
350.    -------------------------
351.    -- ">" (verwendet "-") --
352.    -------------------------
353.    function ">" (Left, Right: Grosse_Zahl) return Boolean is
354.       E:Grosse_Zahl;
355.    begin
356.       E:=Left-Right;
357.       return Element(E,1)/='-';
358.    end ">";
359.
360.    --------------------------
361.    -- ">=" (verwendet "-") --
362.    --------------------------
363.    function ">=" (Left, Right: Grosse_Zahl) return Boolean is
364.       E:Grosse_Zahl;
365.    begin
366.       E:=Left-Right;
367.       return Element(E,1)/='-'or E=Gz_0;
368.    end ">=";
369.
370.    -------------------------
371.    -- "<" (verwendet ">") --
372.    -------------------------
373.    function "<" (Left, Right: Grosse_Zahl) return Boolean is
374.    begin
375.       return Right>Left;
376.    end "<";
377.
378.    --------------------------
379.    -- ">" (verwendet ">=") --
380.    --------------------------
381.    function "<=" (Left, Right: Grosse_Zahl) return Boolean is
382.    begin
383.       return Right>=Left;
384.    end "<=";
385.
386. end Gzahl;
```

**Aufgabe 2 Ringliste I (schriftlich, mittel) 7 Punkte**

*Eine Ringliste ist eine sehr interessante Speicherstruktur. In dieser Aufgabe lernen Sie damit umzugehen.*

Implementieren Sie einen abstrakten Datentyp Ringlist, der eine einfach verkettete Liste realisiert. Die Elemente der Liste sollen natürliche Zahlen (ohne Null) sowie das ausgezeichnete Nullelement (repräsentiert durch Null) sein. Es sollen folgende Operationen zur Verfügung gestellt werden:

- `Create return RingList`; Erzeugt eine leere Ringliste.
- `Insert(RL: in out RingList; E: in Natural);` Fügt das Element E ein und zwar an der Position hinter dem Element, auf das der Zeiger RL zeigt. RL soll nun auf das neu eingefügte Element zeigen. Handelt es sich bei dem einzufügenden Element um das Nullelement, so bleibt die Ringliste unverändert. Das Nullelement wird also nicht eingefügt.
- `Value(RL: in RingList) return Natural;` Liefert das Element der Ringliste, auf das RL zeigt. Ist die Ringliste leer, so wird das Nullelement zurückgeliefert.
- `RotateForward(RL: in out RingList);` Rotiert die Ringliste um eine Position in Richtung der Verkettung.
- `RotateBackward(RL: in out RingList);` Rotiert die Ringliste um eine Position entgegen der Verkettungsrichtung.
- `Delete(RL: in out RingList);` Löscht das Element, auf das RL zeigt, aus der Ringliste. RL zeigt dann auf das dem gelöschten Element nachfolgende Element. Ist die Liste leer oder enthält sie nur ein Element, so ist das Ergebnis eine leere Liste.

**Aufgabe 3 Ringliste II (schriftlich, schwer) 4 Punkte**

*Diese Aufgabe ist jetzt etwas trickreich. Aber man lernt sehr gut den Umgang mit Zeigern. Versuchen Sie mit möglichst wenigen auszukommen.*

Implementieren Sie eine weitere Operation auf dem abstrakten Datentyp Ringliste aus der vorigen Aufgabe:

- `Reverse(RL: in out RingList);` Kehrt die Richtung der Verkettung der Ringliste um.

```
 1. -- EfidI1 WS02/03
 2. -- Blatt 9 Aufgabe 2/3
 3. -- Autor: Jörgen Bertele
 4.
 5. package Ringliste is
 6.
 7.    type Element is new Natural;
 8.    type Ringlist is private;
 9.
10.    function Create return Ringlist;
11.    Procedure Insert(Rl: in out Ringlist; E: in Element);
12.    function  Value (Rl: in Ringlist) return Element;
13.    procedure Rotateforward (Rl: in out Ringlist);
14.    procedure Rotatebackward(Rl: in out Ringlist);
15.    procedure Delete(Rl: in out Ringlist);
16.    procedure rev(Rl: in out Ringlist);
17.
18. private
19.    type ringelement;
20.    type ringlist is access ringelement;
21.    type ringelement is record
22.          value: Element;
23.          pointer: Ringlist;
24.    end record;
25. end Ringliste;
```

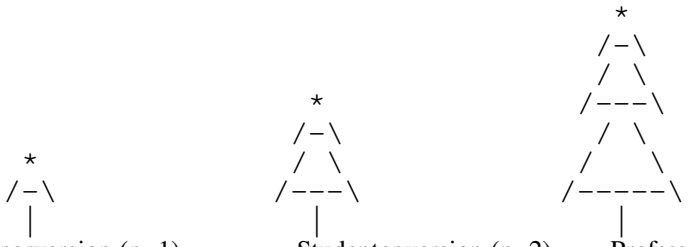
```
1. -- EfidI1 WS02/03
2. -- Blatt 9 Aufgabe 2/3
3. -- Autor: Jörgen Bertele
4.
5.
6. package body Ringliste is
7.
8.    ------------
9.    -- Create --
10.   ------------
11.
12.   function Create return Ringlist is
13.   begin
14.      return null;
15.   end Create;
16.
17.   ------------
18.   -- Delete --
19.   ------------
20.
21.   procedure Delete (Rl: in out Ringlist) is
22.      Todelete: Ringlist;
23.   begin
24.      if Rl/=null then
25.         if Rl.Pointer = Rl then
26.            Rl:=null;
27.         else
28.            Todelete:=Rl;
29.            Rotatebackward(Rl);
30.            Rl.Pointer:=Todelete.Pointer;
31.         end if;
32.      end if;
33.   end Delete;
34.
35.   ------------
36.   -- Insert --
37.   ------------
38.
39.   procedure Insert (Rl: in out Ringlist; E: in Element) is
40.      Newlistelem: Ringlist;
41.   begin
42.      if E/=0 then
43.         Newlistelem:= new Ringelement;
44.         Newlistelem.Value:=E;
45.         if Rl /= null then
46.            Newlistelem.Pointer:=Rl.Pointer;
47.            Rl.Pointer:=Newlistelem;
48.         else
49.            Newlistelem.Pointer:=Newlistelem;
50.            Rl:=Newlistelem;
51.         end if;
52.      end if;
53.   end Insert;
54.
55.   ---------
56.   -- rev --
57.   ---------
58.
59.   procedure Rev (Rl: in out Ringlist) is
60.      Prev: Ringlist;
61.      Curr: Ringlist;
62.      Next: Ringlist;
63.   begin
64.      if Rl/=null then
65.         Curr:=Rl.Pointer;
66.         Next:=Curr.Pointer;
67.         Prev:=Rl;
68.         while Next/=Prev loop
69.            Curr.Pointer:=Prev;
70.            Prev:=Curr;
71.            Curr:=Next;
72.            Next:=Next.Pointer;
73.         end loop;
74.         Rl.Pointer:=Prev;
75.      end if;
76.   end Rev;
```

```
77.
78.    --------------------
79.    -- Rotatebackward --
80.    --------------------
81.
82.    procedure Rotatebackward (Rl: in out Ringlist) is
83.       Old: Ringlist;
84.    begin
85.       if Rl/=null then
86.          Old:=Rl;
87.          while Rl.Pointer /= Old loop
88.             Rl:=Rl.Pointer;
89.          end loop;
90.       end if;
91.
92.    end Rotatebackward;
93.
94.    -------------------
95.    -- Rotateforward --
96.    -------------------
97.
98.    procedure Rotateforward (Rl: in out Ringlist) is
99.    begin
100.      if Rl/=null then
101.         Rl:=Rl.Pointer;
102.      end if;
103.   end Rotateforward;
104.
105.   -----------
106.   -- Value --
107.   -----------
108.
109.   function Value(Rl: in Ringlist) return Element is
110.
111.   begin
112.      if Rl=null then return 0;
113.      end if;
114.      return Rl.Value;
115.   end Value;
116.
117. end Ringliste;
```

**Aufgabe 4 Christbäume (schriftlich, leicht) 4 Punkte**

*Tja Weihnachten naht, das Jahr 2002 ist fast zu Ende und überall sieht man Weihnachtsbäume, warum dann nicht auch bei uns auf dem Bildschirm?!*

Schreiben Sie ein Ada95-Programm, das in Abhängigkeit von einem Parameter n (als Übergabeparameter auf der Kommandozeile) Christbäume auf dem Bildschirm ausgibt. Die Christbäume sollen nach folgendem Muster gebaut sein:



schwäbische Sparversion (n=1) Studentenversion (n=2) Professorenversion (n=3)
usw.

Erzeugen Sie die Bäume mit Hilfe von Schleifen. Die Bildschirme, die Ihnen zur Verfügung stehen, können 25 Zeilen mit je 80 Zeichen darstellen. Es gibt also eine maximale Grenze für *n*. Wird ein größeres *n* übergeben, so soll eine Erklärung auf dem Bildschirm ausgegeben werden. Übrigens finden Sie alles um Argumente von der Kommandozeile zu lesen im Paket Command_Line.

```
 1. -- EfidI1 WS02/03
 2. -- Blatt 9 Aufgabe 4
 3. -- Autor: Hilling / Bertele
 4.
 5. with Text_Io,Ada.Command_Line;
 6. use Text_Io,Ada.Command_Line;
 7. procedure Baum is
 8.    Fehler : exception;
 9.    G      : Integer;
10.
11.    procedure Putn (
12.          C : Character;
13.          N : Natural    ) is
14.    begin
15.       for I in 1..N loop
16.          Put(C);
17.       end loop;
18.    end Putn;
19.
20.    procedure Put_Tree_Line (
21.          G,
22.          N : Natural;
23.          C : Character ) is
24.    begin
25.       Putn(' ',G-N);
26.       Put('/');
27.       Putn(C, 2*N-1);
28.       Put_Line("\");
29.    end Put_Tree_Line;
30.
31.    procedure Put_Tree_Part (
32.          G,
33.          N : Natural ) is
34.    begin
35.       for I in 1.. N-1 loop
36.          Put_Tree_Line(G,I,' ');
37.       end loop;
38.       Put_Tree_Line(G,N,'-');
39.    end Put_Tree_Part;
```

```
40. begin
41.    if Argument_Count/=1 then
42.       raise Fehler;
43.    end if;
44.    if Argument(1)'Length /=1 then
45.       raise Fehler;
46.    end if;
47.    G:=Integer(Character'Pos(Argument(1)(1))-Character'Pos('0'));
48.    if G not in 1..6 then
49.       raise Fehler;
50.    end if;
51.    Putn(' ',G);
52.    Put_Line("*");
53.    for I in 1..G loop
54.       Put_Tree_Part(G,I);
55.    end loop;
56.    Putn(' ',G);
57.    Put_Line("|");
58.
59. exception
60.    when others =>
61.       Put_Line("Syntax: baum [1..6]");
62. end Baum;
```