On Parallel Tree Traverse in Ada

On Parallel Tree Traverse in Ada@

Zensho Nakao*, Kazuo Takeuchi# & Takashi Yogi#

Abstract Two new methods for parallel tree traverse are given, and their implementations are described and tested in Ada which supports parallel programming via Ada tasks. The simulations performed show that (1) when there are no restriction imposed on the number of Ada tasks, the amount of time required for the traverse remains constant (equaling a number related to the height of the tree) for varying number of tree nodes; (2) in case where a pre-assigned number of tasks are used, the traverse time varies inversely as the number of tasks.

0. Preliminary remarks on Ada tasks

Ada tasks are for parallel and real-time programming. An Ada task consists of two parts, i. e., a task specification and a task body, where the former specifies an interface with the rest of the program, and the latter gives an implementation of the objectives of the task. An example of a procedure is given below where two tasks are used; the remarks following "--" symbols should make the list self-explanatory.

```
with text_io;
use text_io;
procedure TASKTEST is
  task T1;                -- task T1 specification
  task T2 is          -- task T2 specification
    entry E1;          -- entry point of task T2
    entry E2;          -- another entry point of task T2
  end task T2;
  task body T1 is      -- task T1 body
  begin
    loop
      T2.E1;          -- entry call to task T2
      T2. E2;       -- another entry call to task T2
    end loop;
```

Received on: May 10,1989.

*Engineering Common Course, #Electrical & Information Engineering

@A preliminary version of this paper was presented at the 40-th Kyushu Regional Joint Meeting of EE-related Societies, Ryukyu Univ., Okinawa, Japan, Oct., 1987.

```
  end T1;
  task body T2 is      -- task T2 body
  begin
    loop
      accept E1;       -- accepts an entry call to E1
      put("Entry E1 is accepted.");
      accept E2 do     -- accepts an entry call to E2
        put ("Entry E2 is accepted.")
      end E2;
    end loop;
  end T2;
begin
  null;
end TASKTEST;
```

1. Traverse methods and evaluations

Given a tree in which no nodes share a common child node. we will consider parallel/concurrent processes which travel all the nodes of the tree. With a sequential method a process can traverse only a subtree at a time, but with parallel methods the processes can traverse the subtrees simultaneously. Here, we will introduce two parallel algorithms.

We will first consider the method in which no restricion on the number of tasks generated is placed. The steps are as follows:

O. To begin with, generate one starting traverse task, (i.e., the task which actually travels along through nodes of a tree) at the root of the tree given. Only traverse tasks should move.

1. Each traverse task reads and display the content of the node where it is at. (It may have to carry out additional work at the node, depending on the nature of the tree to be traced.)

2. Next, each traverse task requests generation of a new child traverse task which behaves exactly the same as the requesting parent task corresponding to each child node except left most child node.

3. Each parent, i. e., old traverse task moves toward the left most node, which is reserved for itself. If no further travel is possible, i.e., the task is at a leaf of the tree, then the task stops its move.

To implement this algorithm into an Ada program, we need one starting traverse task and one generator task which generates traverse tasks at a request of traverse tasks. Upon creation, a new traverse task must be informed on which branching node to go. This information passing can possibly be done via an Ada rendezvous between the old and new tasks. However, a rendezvous between a task and itself is impossible, thus, one generator task is prepared for the purpose, where an old traverse task requests the generator task to create new tasks which continue their traverse further toward the leaves.

In Fig.1 is shown a traverse order of the tree nodes. Note that each task stays at each node for 1.0

second (with a delay statement). Table 1 shows traverse time required. What we find from the results are (1) the traverse time requiued is K*(tree height)+1) (seconds), where K($\geq$1 sec.) denotes the duration of stay at each node (which may be required for processing of information at the node); (2) the number of tasks generated equals the number of leaves; (3) the order of node traverse seems unpredictable.

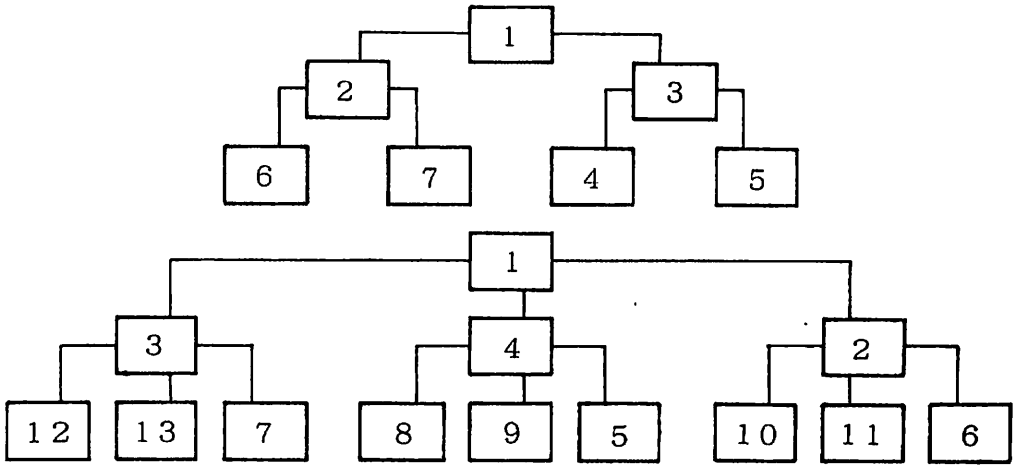

Fig. 1 Traverse order of the tree nodes without restriction on the number of tasks

Table 1 Traverse time* without restriction on the number of tasks

number of branches per node	number of nodes	traverse time (sec)	
		without a task	with tasks
2	7	7.30	3.18
3	13	13.67	3.29
4	21	21.97	3.40
5	31	32.40	3.57

(*Each tree has height 2)

We next consider the second method for tree traverse where a pre–assigned number of Ada tasks are allowed. The steps are as follows:

0. To start with, place into a queue a pointer to the tree root node.
1. Each traverse task picks up a pointer from the queue.
2. Place into the the queue all the pointers (to the next nodes) coming out from the node that was indicated by the pointer just picked.
3. Each traverse task first displays the content of the node where it is at (and may do some additional work depending on properties of the tree traversed.)
4. Each traverse task repeats steps 1–3 until the queue becomes empty.

To express this algorithm in Ada, we introduce a fixed number of traverse tasks, a queue task which keeps record of the pointers sent for from the traverse tasks, and a semaphore task which keeps the traverse tasks from accessing the queue during steps 1 and 2. In Fig. 2 is a block diagram for the method.

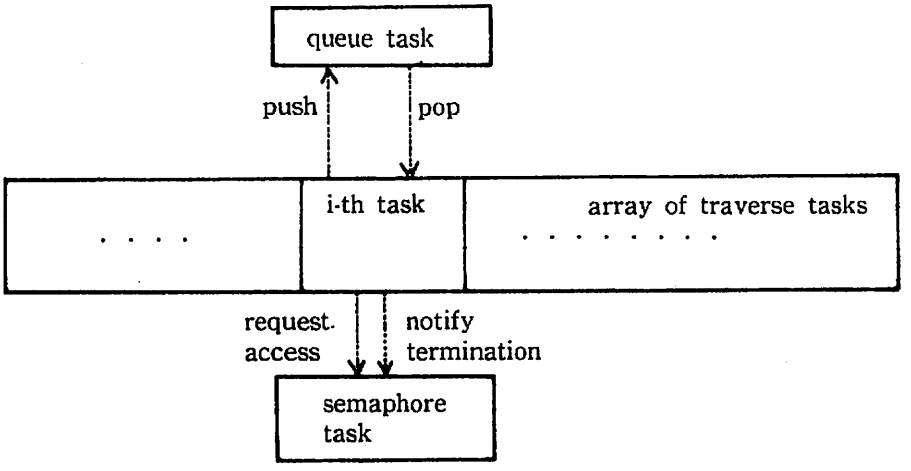

Fig.2 Fixed number of traverse tasks in action

We show in Fig. 3 a traverse order in the method. As in the first method, each traverse task stays at each node for 1.0 second. In Table 2 is shown the total traverse time corresponding to varying combinations of the number of tasks and nodes. We find from the table that the the product of the traverse time and the task number equals the node number. Thus, in case the traverse tasks stay at each node for K($>$ 1) seconds, the traverse time is K * (node number of the tree)/(task number) seconds. Further, we find that the traverse order of the nodes is fixed; and so the present method is better suited for traversing large trees than the first method, but it is necessary to prepare a large array for the queue task.

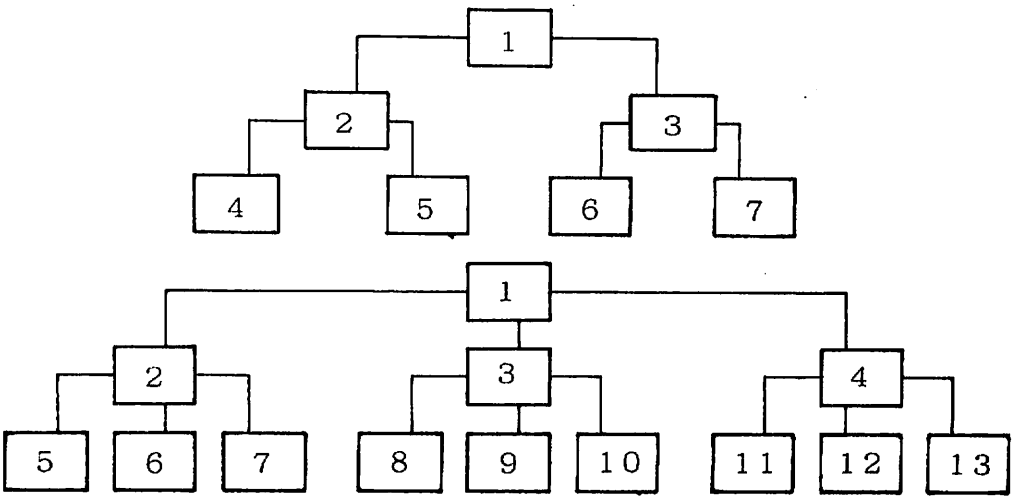

Fig. 3 Traverse order of tree nodes with a fixed number of traverse tasks

Table 2. Traverse time* with a fixed number of traverse tasks

number of branches per node	number of nodes	traverse time (sec)			
		number of tasks			
		1	2	3	4
2	7	7.36	4.22	3.18	2.41
3	13	13.78	7.52	5.27	4.22
4	21	22.13	11.69	7.79	6.31
5	31	32.73	16.91	11.53	8.67

(*Each tree has height 2)

2. Conclusion and remarks

With a usual single processor method (Table 1, without a task or Table 2, with a single task), the traverse time will be at least K*(the number of nodes), where K is the duration of stay at each node; by the multi-task method (i) without restriction on the number of tasks, the traverse time is given by K*(the tree height +1), (ii) with a fixed number of tasks, the time required is K*(the number of nodes)/(the number of tasks), reducing the processing time considerably.

The second multi-task method can be used for tree searching problems because the traverse order of the nodes is known (Fig. 3). Further simulation with taller trees should be done to confirm the validity of those formulas obtained.

References

[G] Gehani, Narain: *Ada: An Advanced Introduction*, Prentice-Hall, Inc., 1983.

[T] Takeuchi, Kazuo: *B. E. Thesis 1983* (in Japansese), Ryukyu Univ., (Unpublished).

Appendix

— Source programs and packages —

[*TT*1] . . . . . *without restriction on the number of tasks*
[*TT*2] . . . . . *without restricrion on the number of tasks (with queue)*
[*TT*3] . . . . . *no task*
[*TT*4] . . . . . *with a fixed number of tasks*
[*TREE–P*] . . . . . *package for constructing trees*
[*QUEUE–MANAGER*] . . *generic package for queues*
[*TIMER*] . . . . . *package for timing*
[*T–OUT*] . . . . . *package for constructing trees (used with TREE–P)*

File name : TT1.ADA Page : 1

```
 1: ------------------------------------------------------------------
 2: --  TEST PROGRAM TREE TRACER !                                   --
 3: --  PROGRAM     :    TT1                                         --
 4: --                 ( PROGRAM )                                  --
 5: --  PROGRAMMER  :  K.Takeuchi                                    --
 6: --  DATE        : JULY 20, 1987.                                 --
 7: --                                                               --
 8: ------------------------------------------------------------------
 9:
10: with text_io, timer, tree_p; use text_io, timer, tree_p;
11:
12: procedure TT1 is
13:
14:   package int_io is new integer_io( integer );
15:   use int_io;
16:
17:   root : link := null;
18:
19: ----- task ---------------------------------------------------------
20:
21:   task worm_maker is
22:     entry create( r : in link; lv : in integer );
23:     entry search( r : out link; id, lv : out integer );
24:   end worm_maker;
25:
26:   task type worm_body;
27:
28:   type worm_type is access worm_body;
29:
30: ----- task body ----------------------------------------------------
31:
32:   task body worm_maker is
33:     worm : worm_type;
34:     sub_t : link := null;
35:     level : integer := 0;
36:     id_number : integer := 1;
37:   begin
38:     loop
39:       select
40:         accept create( r : in link; lv : in integer ) do
41:           sub_t := r;  level := lv;
42:         end create;
43:         worm := new worm_body;
44:         accept search( r : out link; id, lv : out integer ) do
45:           r := sub_t;  id := id_number; lv := level;
46:         end search;
47:         id_number := id_number + 1;
48:       or
49:         terminate;
50:       end select;
```

File name : TT1.ADA Page : 2

```
51:     end loop;
52:   end worm_maker;
53:
54: ----- task body ----------------------------------------------------
55:
56:   task body worm_body is
57:     sub_t : link := null;
58:     level, id_number, r_number : integer := 1;
59:   begin
60:     worm_maker.search( sub_t, id_number, level );
61:     put( "WORM #" ); put( id_number, 3 ); put( " STARTS : " ); lap_time;
62:     while sub_t /= null loop
63:       put( "WORM #" );  put( id_number, 3 );  put( " : " );
64:       for i in 1..sub_t.n'last loop
65:         put( sub_t.n(i) );
66:       end loop;
67:       put( "   " ); lap_time;
68:       delay 1.0;
69:       for i in reverse 2..sub_t.b'last loop
70:         if sub_t.b(i) /= null then
71:           worm_maker.create( sub_t.b(i), level+1 );
72:         end if;
73:       end loop;
74:       level := level + 1;
75:       sub_t := sub_t.b(1);
76:     end loop;
77:     put( "WORM #" ); put( id_number, 3 ); put( " STOPS  : " ); lap_time;
78:   end worm_body;
79:
80: ----- main ---------------------------------------------------------
81:
82: begin
83:
84:   grow( root );
85:
86:   put_line( "---------- output of TT1 ----------" );
87:
88:   put( "Main routine starts : " ); timer_start;
89:   worm_maker.create( root, 0 );
90:   put( "Main routine stops  : " ); lap_time;
91:
92: end TT1;
```

File name : TT2.ADA Page : 1

```
 1: ------------------------------------------------------------------
 2: --  TEST PROGRAM TREE_TRACER 2                                   --
 3: --  PROGRAM     :   TT2                                          --
 4: --                  ( PROGRAM )                                  --
 5: --  PROGRAMMER  : K.Takeuchi                                     --
 6: --  DATE        : JULY 20, 1987.                                 --
 7: --                                                               --
 8: ------------------------------------------------------------------
 9:
10: with text_io,timer,tree_p,queue_manager;
11: use  text_io,timer,tree_p;
12:
13: procedure TT2 is
14:
15:   package io_int is new integer_io( integer );
16:   use io_int;
17:
18:   type order_queue_element is record
19:     sub_tree : link ;
20:     level    : integer ;
21:   end record;
22:
23:   limit_queue : constant positive := 512;
24:
25:   package order_queue is new queue_manager( limit_queue, order_queue_
      element );
26:   use order_queue;
27:
28:   root : link := null;
29:   dead_worm : integer := 1;
30:
31: ----- task ------------------------------------------------------
32:
33:   task worm_maker is
34:     entry search( r : out link; id, lv : out integer );
35:   end worm_maker;
36:
37:   task type worm_body;
38:
39:   task order is
40:     entry add_to_queue( r : in link; lv : in integer );
41:     entry get_from_queue( r : out link; lv : out integer );
42:   end order;
43:
44:   type worm_type is access worm_body;
45:
46: ----- task body -------------------------------------------------
47:
48:   task body order is
49:     order_record : order_queue_element;
50:   begin
```

File name : TT2.ADA Page : 2

```
51:     loop
52:       select
53:         when queue_full = false =>
54:           accept add_to_queue( r : in link; lv : in integer ) do
55:             order_record.sub_tree := r;
56:             order_record.level := lv;
57:           end add_to_queue;
58:           queue( order_record );
59:       or
60:         when queue_empty = false =>
61:           accept get_from_queue( r : out link; lv : out integer ) do
62:             order_record := unqueue;
63:             r := order_record.sub_tree;
64:             lv := order_record.level;
65:           end get_from_queue;
66:       or
67:         terminate;
68:       end select;
69:     end loop;
70:   end order;
71:
72: ----- task body -------------------------------------------------
73:
74:   task body worm_maker is
75:     worm : worm_type;
76:     sub_tree : link := null;
77:     level, id_number : integer := 1;
78:   begin
79:     loop
80:       select
81:         order.get_from_queue( sub_tree, level );
82:         worm := new worm_body;
83:         accept search( r : out link; id, lv :  out integer ) do
84:           r := sub_tree; id := id_number; lv := level;
85:         end search;
86:         id_number := id_number + 1;
87:       else
88:         exit when dead_worm /= 1 and id_number = dead_worm;
89:       end select;
90:     end loop;
91:   end worm_maker;
92:
93: ----- task body -------------------------------------------------
94:
95:   task body worm_body is
96:     sub_tree : link := null;
97:     level, id_number, r_number : integer := 1;
98:   begin
99:     worm_maker.search( sub_tree, id_number, level );
100:    put( "WORM #" ); put( id_number, 3 ); put( " STARTS : " ); lap_time;
```

File name : TT2.ADA Page : 3

```
101:     while sub_tree /= null loop
102:       put( "WORK |" );  put( id_number, 3 );  put( " : " );
103:       for i in sub_tree.a'range loop
104:         put( sub_tree.a(i) );
105:       end loop;
106:       put( "   " ); lap_time;
107:       delay 1.0;
108:       for i in reverse 2..sub_tree.b'last loop
109:         if sub_tree.b(i) /= null then
110:           order.add_to_queue( sub_tree.b(i), level+1 );
111:         end if;
112:       end loop;
113:       level := level + 1;
114:       sub_tree := sub_tree.b(1);
115:     end loop;
116:     dead_worn := dead_worn + 1;
117:     put( "WORK #" ); put( id_number, 3 ); put( " STOPS  : " ); lap_time;
118:   end worn_body;
119:
120: ------ main ------------------------------------------------------
121:
122: begin
123:
124:   grow( root );
125:
126:   put_line( "---------- output of TT2 ----------" );
127:   put( "Main routine starts : " ); timer_start;
128:   order.add_to_queue( root, 0 );
129:   put( "Main routine stops  : " ); lap_time;
130:
131: end TT2;
```

File name : TT3.ADA Page : 1

```
 1: ------------------------------------------------------------------
 2: --  TEST PROGRAM TREE_TRACER 3                                   --
 3: --  PROGRAM     :    TT3                                         --
 4: --                 ( PROGRAM )                                  --
 5: --  PROGRAMMER  :  K.Takeuchi                                    --
 6: --  DATE        : JULY 20, 1987.                                 --
 7: --                                                               --
 8: ------------------------------------------------------------------
 9:
10: with text_io,timer,tree_p; use text_io,timer,tree_p;
11:
12: procedure TT3 is
13:
14:   package io_int is new integer_io( integer );
15:   use io_int;
16:
17:   root : link := null;
18:
19: ------ task body ------------------------------------------------
20:
21:   procedure worn( sub_tree : in link; level : in integer ) is
22:   begin
23:     put( "CALL  TIME = " ); lap_time;
24:     if sub_tree /= null then
25:       put( "DATA  : " );
26:       for i in 1..sub_tree.n'last loop
27:         put( sub_tree.n(i) );
28:       end loop;
29:       put( "   " ); lap_time;
30:       delay 1.0;
31:       for i in 1..sub_tree.b'last loop
32:         worn( sub_tree.b(i), level+1 );
33:       end loop;
34:     else
35:       null;
36:     end if;
37:     put( "RETURN TIME = " ); lap_time;
38:   end worn;
39:
40: ------ main ----------------------------------------------------
41:
42: begin
43:
44:   grow( root );
45:
46:   put_line( "---------- output of TT3 ----------" );
47:   put( "Main routine starts : " ); timer_start;
48:   worn( root, 0 );
49:   put( "Main routine stops : " ); lap_time;
50:
```

File name : TT3.ADA Page :

```
51: end TT3;
```

File name . TT4.ADA Page : 1

```
 1: ------------------------------------------------------------
 2: --  TEST PROGRAM FOR TASK TRACER I                        --
 3: --  PROGRAM    :    TT4                                   --
 4: --               ( PROGRAM )                             --
 5: --  PROGRAMMER : K.Takeuchi.                             --
 6: --  DATE       : JULY 20, 1987.                          --
 7: --                                                        --
 8: ------------------------------------------------------------
 9:
10: with text_io,timer,tree_p,queue_manager;
11: use text_io,timer,tree_p;
12:
12: procedure tt4 is
14:
15:    package int_io is new integer_io( integer );
16:    use int_io;
17:
18:    queue_size : constant positive := 5120;
19:
20:    package que_io is new queue_manager( queue_size, link );
21:    use que_io;
22:
23:    subtype readers is integer range 1..2;
24:
25:    root : link;
26:
27:    task type semaphore is
28:       entry p;
29:       entry v;
30:    end semaphore;
31:
32:    task type reading_task is
33:       entry run( task_id : in readers );
34:    end reading_task;
35:
36:    task link_queue is
37:       entry push( sub_tree : in link );
38:       entry pop( sub_tree : out link );
39:       entry task_stop( answer : out boolean );
40:    end link_queue;
41:
42:    s      : semaphore;
43:    reader : array( readers ) of reading_task;
44:
45:    task body semaphore is
46:    begin
47:       loop
48:          select
49:             accept p;
50:             accept v;
```

File name : TT4.ADA Page : 2

```
51:          or
52:             terminate;
53:          end select;
54:       end loop;
55:    end semaphore;
56:
57:    task body link_queue is
58:       stopped_task : integer := 0;
59:    begin
60:       loop
61:          select
62:             when not queue_full =>
63:                accept push( sub_tree : in link ) do
64:                   queue( sub_tree );
65:                end push;
66:          or
67:             when not queue_empty =>
68:                accept pop( sub_tree : out link ) do
69:                   sub_tree := unqueue;
70:                end pop;
71:          or
72:             accept task_stop( answer : out boolean ) do
73:                answer := queue_empty;
74:             end task_stop;
75:          or
76:             terminate;
77:          end select;
78:       end loop;
79:    end link_queue;
80:
81:    task body reading_task is
82:       sub_tree : link;
83:       id_number : readers := 1;
84:       stop_ok : boolean := false;
85:    begin
86:       accept run( task_id : in readers ) do
87:          id_number := task_id;
88:       end run;
89:       put( "READER" ); put( id_number, 3 ); put( " STARTS : " ); lap_time;
90:       loop
91:          s.p;
92:          link_queue.task_stop( stop_ok );
93:          exit when stop_ok;
94:          link_queue.pop( sub_tree );
95:          for i in sub_tree.b'range loop
96:             if sub_tree.b(i) /= null then
97:                link_queue.push( sub_tree.b(i) );
98:             end if;
99:          end loop;
100:         s.v;
```

File name : TT4.ADA Page : 3

```
101:          put( "READER" ); put( id_number, 3 ); put( " : " );
102:          for i in sub_tree.n'range loop
103:             put( sub_tree.n(i) );
104:          end loop;
105:          put( "  TIME = " );  lap_time;
106:          delay 1.0;
107:       end loop;
108:       s.v;
109:       put( "READER" ); put( id_number, 3 ); put( " STOPS  : " ); lap_time;
110:    end reading_task;
111:
112: begin
113:
114:    grow( root );
115:    put_line( "---------- output of TT4 ----------" );
116:    put( "MAIN ROUTINE STARTS : " ); timer_start;
117:    link_queue.push( root );
118:    for i in readers loop
119:       reader( i ).run( i );
120:    end loop;
121:    put( "MAIN ROUTINE STOPS  : " ); lap_time;
122:
123: end tt4;
```

File name : TREE_P.ADA Page : 1

```
 1: ----------------------------------------
 2: --  Program    :  TREE_P               --
 3: --             ( package )             --
 4: --  Programmer :  K.Takeuchi           --
 5: --  Date       :  July 20, 1987.       --
 6: --                                      --
 7: ----------------------------------------
 8:
 9: package TREE_P is
10:
11:   type NODE( MAX_ELEMS : INTEGER := 1;
12:              MAX_LINKS : INTEGER := 2 );
13:
14:   type LINK is access NODE;
15:   type LINKS is array( INTEGER range <> ) of LINK;
16:
17:   type ELEMS is array( INTEGER range <> ) of INTEGER;
18:
19:   type NODE( MAX_ELEMS : INTEGER := 1;
20:              MAX_LINKS : INTEGER := 2 ) is record
21:     N : ELEMS( 1..MAX_ELEMS );
22:     B : LINKS( 1..MAX_LINKS ) := ( 1..MAX_LINKS => NULL );
23:   end record;
24:
25:   procedure grow( r : in out link );
26:
27: end TREE_P;
28:
29: ------------------------------------------------------------------
30: --  Program    :   TREE_P                                        --
31: --               ( package body )                                --
32: --  Programmer :  K.Takeuchi                                     --
33: --  Date       :  JULY 20, 1987.                                 --
34: --                                                               --
35: ------------------------------------------------------------------
36:
37: with text_io; use text_io;
38:
39: package body TREE_P is
40:
41:   package io_int is new integer_io( integer );
42:   use io_int;
43:
44:   procedure grow( r : in out link ) is
45:     links_max, elems_max, e_number : integer := 0;
46:   begin
47:     get( links_max );
48:     get( elems_max );
49:     if links_max = 0 then
50:       r := new node( elems_max, 1 );
```

File name : TREE_P.ADA Page : 2

```
51:     else
52:       r := new node( elems_max, links_max );
53:     end if;
54:     for i in 1..r.n'last loop
55:       get( e_number );  r.n(i) := e_number;
56:     end loop;
57:     if links_max = 0 then
58:       r.b(1) := null;
59:     else
60:       for i in 1..r.b'last loop
61:         grow( r.b(i) );
62:       end loop;
63:     end if;
64:   end grow;
65:
66: end TREE_P;
```

File name : QUEUE.ADA Page : 1

```
 1: ------------------------------------------
 2: --  PROGRAM  QUEUE_MANAGER                --
 3: ------------------------------------------
 4: --  Program designer : M. Nakaoa          --
 5: --  Language         : AdaVantage         --
 6: --  File name        : QUEUE.PKG          --
 7: --  Date    Ver. 1   : June 1th 1987.     --
 8: --          Ver. 2   : July 30th 1987.    --
 9: ------------------------------------------
10: generic
11:     length : positive ;
12:     type elem is private ;
13: package QUEUE_MANAGER IS
14:
15:   type message_array is array( 1.. length ) of elem ;
16:
17:   procedure queue( mess : in elem ) ;
18:
19:   function  unqueue return elem ;
20:
21:   function queue_empty return boolean ;
22:
23:   function queue_full return boolean ;
24:
25: end  QUEUE_MANAGER ;
26:
27: ------------------------------------------
28: --  PROGRAM  QUEUE_MANAGER   BODY        --
29: ------------------------------------------
30: --  Program designer : M. Nakaoa          --
31: --  File name        :   QUEUE_B.ADA      --
32: --  Language         : Alsys/Ada          --
33: --  Date  Ver. 1     : June 1th 1987.     --
34: --        Ver. 2     : July 30th 1987.    --
35: ------------------------------------------
36: package body QUEUE_MANAGER is
37:
38:     type situation is ( empty, full, normal ) ;
39:     STI : situation := empty ;
40:   message_queue : message_array ;
41:     input_point : positive := 1 ;
42:     output_point :  positive := 1 ;
43:
44:   procedure queue( mess : in elem ) is
45:
46:    begin
47:
```

File name : QUEUE.ADA Page : 2

```
48:     STI := normal ;
49:     message_queue( input_point ) := mess ;
50:     input_point := input_point mod length + 1 ;
51:      if input_point = output_point then
52:         STI := full ;
53:      end if ;
54:
55:  end queue ;
56:
57:
58:   function unqueue return elem is
59:
60:    put_data : elem ;
61:
62:     begin
63:       STI := normal ;
64:        put_data := message_queue( output_point ) ;
65:        output_point := output_point mod length + 1 ;
66:        if output_point = input_point then
67:        STI := empty ;
68:        end if ;
69:      return put_data ;
70:   end unqueue ;
71:
72:
73:    function queue_empty return boolean is
74:     begin
75:       return ( STI = empty ) ;
76:     end queue_empty ;
77:
78:
79:    function queue_full return boolean is
80:       begin
81:         return ( STI = full ) ;
82:       end queue_full ;
83:
84: end QUEUE_MANAGER ;
```

*Grad College of Engineering, Osaka Univ.

File name : TIMER.ADA Page : 1

```
 1: ------------------------------------------------------------------
 2: --  Package TIMER                                               --
 3: --  program    : TIMER                                          --
 4: --  programmer : K.TAKEUCHI.                                    --
 5: --  date       : AUG. 9, 1987.                                  --
 6: ------------------------------------------------------------------
 7:
 8: package TIMER is
 9:
10:   procedure TIMER_START ;
11:
12:   procedure LAP_TIME ;
13:
14: end TIMER;
15:
16: ------------------------------------------------------------------
17: --  Package TIMER                                               --
18: --  program    : TIMER                                          --
19: --  programmer : K.TAKEUCHI.                                    --
20: --  date       : AUG. 9, 1987.                                  --
21: ------------------------------------------------------------------
22:
23: with text_io,time,date_types; use text_io,date_types;
24:
25: package body TIMER is
26:
27:   package io_int is new integer_io( integer );
28:   use io_int;
29:
30:   procedure TIMER_START is
31:   begin
32:     put( "00:00:00'00" ); new_line;
33:     time.set( 0, 0, 0, 0 );
34:   end TIMER_START;
35:
36:   procedure LAP_TIME is
37:     zero    : constant integer := character'pos('0');
38:     s       : string( 1..11 )  := ( "00:00:00'00" );
39:     hour    : hours_range;
40:     minute  : minutes_range;
41:     second  : seconds_range;
42:     hundred : hundredths_range;
43:   begin
44:     time.get( hour, minute, second, hundred );
45:     s( 1) := character'val( zero + hour   /  10 );
46:     s( 2) := character'val( zero + hour mod 10 );
47:     s( 4) := character'val( zero + minute /  10 );
48:     s( 5) := character'val( zero + minute mod 10 );
49:     s( 7) := character'val( zero + second /  10 );
50:     s( 8) := character'val( zero + second mod 10 );
```

File name : TIMER.ADA Page : 2

```
51:     s(10) := character'val( zero + hundred  /  10 );
52:     s(11) := character'val( zero + hundred mod 10 );
53:     put_line( s );
54:   end LAP_TIME;
55:
56: end TIMER;
```

File name : T_OUT.ADA Page : 1

```
 1: ------------------------------------------------
 2: --  program   :  T_OUT                       --
 3: --  programmer : K.Takeuchi.                  --
 4: --  date      : Feb. 22, 1988.                --
 5: ------------------------------------------------
 6:
 7: with text_io; use text_io;
 8:
 9: procedure t_out is
10:
11:   package natural_io is new integer_io( natural );
12:   use natural_io;
13:
14:   branch : natural := 0;
15:   height : integer := -1;
16:
17:   procedure out_data( b, h, n : in natural ) is
18:
19:     procedure put_string( n : in natural ) is
20:       a : integer := n;
21:       l : natural := 0;
22:       c : character := ' ';
23:       s : string( 1..6 ) := ( 1..6 => ' ' );
24:     begin
25:       if n = 0 then
26:         l := 1;
27:         s(1) := '0';
28:       else
29:         while a /= 0 loop
30:           l := l + 1;
31:           s( l ) := character'val( a mod 10 + character'pos('0') );
32:           a := a / 10;
33:         end loop;
34:         for i in 1..l/2 loop
35:           c := s(i);
36:           s(i) := s( l-i+1 );
37:           s( l-i+1 ) := c;
38:         end loop;
39:       end if;
40:       put( s(1..l) );
41:     end put_string;
42:
43:   begin
44:
45:     if h = 0 then
46:       put( "0 1 " );
47:       put_string(n);
48:       new_line;
49:     else
50:       put_string(b);
```

File name : T_OUT.ADA Page : 2

```
51:       put( " 1 " );
52:       put_string(n);
53:       new_line;
54:       for i in (2-b)..1 loop
55:         out_data( b, h-1, bint*i );
56:       end loop;
57:     end if;
58:   end out_data;
59:
60: begin
61:
62:   while branch = 0 loop
63:     get( branch );
64:   end loop;
65:
66:   while height < 0 loop
67:     get( height );
68:   end loop;
69:
70:   out_data( branch, height, 1 );
71:
72: end t_out;
```