```
 1  --in file genericTopologicalsort.ads
 2  with Ada.Strings.Fixed;
 3  with Ada.Text_IO;
 4  use Ada.Text_IO;
 5  with Ada.Unchecked_Conversion;
 6  generic  -- You may modify this as required but observe the spirit.
 7     type SortElement is private;  -- An element J (or K) of the partial ordering
       J < K processed
 8     -- by the topological sort.  J and K represent objects in the partial
       ordering.
 9     with procedure getJobs(Precedent, Successor: out SortElement; Current_Line:
       String); --getJobs function to read integer relations from file in the form
       J<K
10     with procedure put(file: File_Type; Job:  in SortElement);  --Print the
       value of J or K.
11
12 package GenericTopologicalSort is
13     procedure TopologicalSort(inputFile, outputFile: String);
14 end GenericTopologicalSort;
15
16 --in file genericTopologicalsort.adb
17 package body GenericTopologicalSort is
18     -- This should read (get) the relations and print (put) the results.
19     type Node;
20     type NodePointer is access Node;
21     type Node is tagged record
22        Suc: SortElement; --Sort element is an "object" placed in the stack.
          changed to integer from SortElement
23        Next: NodePointer;
24     end record;
25
26     type JobElement is record
27        Count: Integer := 0;  -- This field should be used for counting and as
          queue links.
28        Top: NodePointer;
29     end record;
30
31     function Integer_To_Ptr is new Ada.Unchecked_Conversion(Integer, NodePointer);
32     function SEtoint is new Ada.Unchecked_Conversion(SortElement, Integer);
33     package IntegerIO is new Ada.Text_IO.Integer_IO(Integer);
34
35     procedure TopologicalSort(inputFile, outputFile: String) is
36        Precedent, Successor: SortElement;
37        Ptr: NodePointer;
38        F, R, Y, NA, numRelations: Integer;
39        dupe: Boolean := False;
40        f1, f2 : File_Type;
41     begin
42        Ada.Text_IO.Open(File => f1, Mode => In_File, Name => inputFile);
43        Ada.Text_IO.Open(File => f2, Mode => Out_File, Name => outputFile);
44
45        NA := Integer'Value(Get_Line(f1));
46        Put_Line(f2, "Number of tasks to perform: " & NA'Image);
47        numRelations := Integer'Value(Get_Line(f1));
48        Put_Line(f2, "Number of relations to read: " & numRelations'Image);
49
50        declare
51           SortStructure: Array(0..NA) of JobElement;
52           KN: Integer := NA;
53        begin
54
55           --1 Initialization
56           for K in 0..NA loop
```

```
57             SortStructure(K).Count := 0;
58             SortStructure(K).Top := null;
59          end loop;
60
61          --2 Build The Data Structure
62          for J in 1..numRelations loop
63             getJobs(Precedent, Successor, Get_Line(f1));
64
65             --check for duplicates here
66             --iterate through the nodes in each of Precedent's top to find
               duplicate Successor before adding.
67
68             Ptr := SortStructure(SEtoint(Precedent)).Top;
69             dupe := False;
70             while Ptr /= null loop --find duplicates, don't add them.
71                if SEtoint(Ptr.Suc) = SEtoint(Successor) then
72                   dupe := True;
73                   Put_Line(f2, "Ignoring duplicate relation: " &
                     SEtoint(Precedent)'Image & " <" & SEtoint(Successor)'Image);
74                end if;
75                Ptr := Ptr.Next;
76             end loop;
77
78             if dupe = False then  --only add unique relations.
79                Put_Line(f2, "Accepted relation input: " &
                  SEtoint(Precedent)'Image & " <" & SEtoint(Successor)'Image);
80                SortStructure(SEtoint(Successor)).Count :=
                  SortStructure(SEtoint(Successor)).Count + 1;
81                Ptr := new Node'(Suc => Successor, Next =>
                  SortStructure(SEtoint(Precedent)).Top);
82                SortStructure(SetoInt(Precedent)).Top := Ptr;
83             end if;
84          end loop;
85          Ada.Text_IO.Close(f1); --close input file
86
87          --3 Initialize the Output Queue
88          R := 0;
89          SortStructure(0).Count := 0;
90          for K in 1..NA loop
91             if SortStructure(K).Count = 0 then
92                SortStructure(R).Count := K;
93                R := K;
94             end if;
95          end loop;
96          F := SortStructure(0).Count;
97
98          --4
99          Put_Line(f2, "Attempting to sort");
100         new_line;
101         while F /= 0 loop
102            IntegerIO.put(f2, F);--Perform action F Output it
103            KN := KN - 1;
104            Ptr := SortStructure(F).Top;
105            SortStructure(F).Top := Integer_To_Ptr(0);
106            while Ptr /= Integer_To_Ptr(0) loop
107               SortStructure(SEtoint(Ptr.Suc)).Count :=
                  SortStructure(SEtoint(Ptr.Suc)).Count - 1;
108               if SortStructure(SEtoint(Ptr.Suc)).Count = 0 then
109                  SortStructure(R).Count := SEtoint(Ptr.Suc); --Add to output
                     Queue
110                  R := SEtoint(Ptr.Suc);
111               end if;
112               Ptr := Ptr.Next;
```

```
113             end loop;
114             F := SortStructure(F).Count;
115          end loop;
116
117          --5
118          if KN = 0 then
119             Put_Line(f2, "");
120             Put_Line(f2, "Found a solution! See previous line.");
121          else
122             Put_Line(f2, "");
123             Put_Line(f2, "Failed to complete a solution!");
124             Put_Line(f2, "Loop detected.");
125             for K in 1..NA loop
126                SortStructure(K).Count := 0;
127             end loop;
128
129             --6
130             for K in 1..NA loop
131                Ptr := SortStructure(K).Top;
132                SortStructure(K).Top := Integer_To_Ptr(0);
133                while Ptr /= Integer_To_Ptr(0) loop
134                   if SortStructure(SEtoint(Ptr.Suc)).Count = 0 then --trying to
                      test this condition in the above while loop breaks the program.
135                      SortStructure(SEtoint(Ptr.Suc)).Count := K;
136                   end if;
137                   if Ptr /= Integer_To_Ptr(0) then --This is part of the
                      algorithm, but this if statement will never be entered.
138                      Ptr := Ptr.Next;
139                   end if;
140                end loop;
141             end loop;
142
143             --7 Find a K with Qlink(K) /= 0. This will be part of the offending
                loop.
144             Y := 1;
145             while SortStructure(Y).Count = 0 loop
146                Y := Y + 1;
147             end loop;
148
149             --8
150             loop
151                SortStructure(Y).Top := Integer_To_Ptr(1);
152                Y := SortStructure(Y).Count;
153                exit when SortStructure(Y).top /= Integer_To_Ptr(0);
154             end loop;
155
156             --9 Print the loop
157             Put_Line(f2, "Printing offending loop in reverse order");
158             while SortStructure(Y).Top /= Integer_To_Ptr(0) loop
159                IntegerIO.put(f2, Y);
160                SortStructure(Y).Top := Integer_To_Ptr(0);
161                Y := SortStructure(Y).Count;
162             end loop;
163
164          end if;
165       end;
166    end TopologicalSort;
167 end GenericTopologicalSort;
```