Christopher McDaniel

COSC 3319.01

Submitted: 23 October 2019

Grading Option: “**C**”

Data Set 1

Enter the number of jobs: 9
Assuming at least 1 relation, how many relations are there? 15
Enter the Precedent (J) followed by its Successor (K) 1: **1 < 2**
Enter the Precedent (J) followed by its Successor (K) 2: **1 < 3**
Enter the Precedent (J) followed by its Successor (K) 3: **4 < 1**
Enter the Precedent (J) followed by its Successor (K) 4: **3 < 8**
Enter the Precedent (J) followed by its Successor (K) 5: **8 < 2**
Enter the Precedent (J) followed by its Successor (K) 6: **4 < 2**
Enter the Precedent (J) followed by its Successor (K) 7: **4 < 5**
Enter the Precedent (J) followed by its Successor (K) 8: **6 < 4**
Enter the Precedent (J) followed by its Successor (K) 9: **5 < 7**
Enter the Precedent (J) followed by its Successor (K) 10: **2 < 7**
Enter the Precedent (J) followed by its Successor (K) 11: **9 < 8**
Enter the Precedent (J) followed by its Successor (K) 12: **9 < 6**
Enter the Precedent (J) followed by its Successor (K) 13: **2 < 7**
Enter the Precedent (J) followed by its Successor (K) 14: **4 < 2**
Enter the Precedent (J) followed by its Successor (K) 15: **9 < 8**
Here is the result:

9
6
4
5
1
3
8
2
7

This is a Topological Sort

Data Set 2

Enter the number of jobs: 9
Assuming at least 1 relation, how many relations are there? 18
Enter the Precedent (J) followed by its Successor (K) 1: **1 < 2**
Enter the Precedent (J) followed by its Successor (K) 2: **1 < 3**
Enter the Precedent (J) followed by its Successor (K) 3: **2 < 3**
Enter the Precedent (J) followed by its Successor (K) 4: **4 < 1**
Enter the Precedent (J) followed by its Successor (K) 5: **3 < 8**
Enter the Precedent (J) followed by its Successor (K) 6: **8 < 9**
Enter the Precedent (J) followed by its Successor (K) 7: **8 < 2**
Enter the Precedent (J) followed by its Successor (K) 8: **4 < 2**
Enter the Precedent (J) followed by its Successor (K) 9: **4 < 5**
Enter the Precedent (J) followed by its Successor (K) 10: **6 < 4**
Enter the Precedent (J) followed by its Successor (K) 11: **5 < 7**
Enter the Precedent (J) followed by its Successor (K) 12: **2 < 7**
Enter the Precedent (J) followed by its Successor (K) 13: **7 < 9**
Enter the Precedent (J) followed by its Successor (K) 14: **9 < 8**
Enter the Precedent (J) followed by its Successor (K) 15: **9 < 6**
Enter the Precedent (J) followed by its Successor (K) 16: **2 < 7**
Enter the Precedent (J) followed by its Successor (K) 17: **4 < 2**
Enter the Precedent (J) followed by its Successor (K) 18: **9 < 8**
Here is the result:
There is a loop causing problems:

0 <- **0**

----------GenericTopologicalSort.ads----------

```
generic
   type SortElement is (<>); -- An element J (or K) of the partial ordering J < K processed
   -- by the topological sort.  J and K represent objects in the partial ordering.
   with procedure get(Job:  out SortElement);  -- Reads J or K.
   with procedure put(Job:  in SortElement);  -- Print the value of J or K.
   ItemKnt: Integer;

package GenericTopologicalSort is
   procedure TopologicalSort;
   procedure Result;
end GenericTopologicalSort;
```

----------GenericTopologicalSort.adb----------

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Unchecked_Conversion;

----------GenericTopologicalSort body----------
package body GenericTopologicalSort is -- This should read (get) the relations and print (put) the results.
   type Node;
   type NodePointer is access Node;
   type Node is tagged record
      Suc : SortElement;
      Next: NodePointer;
   end record;

   type JobElement is record
      Knt: Integer := 0; -- This field should be used for counting and as queue links.
      Top: NodePointer;
   end record;

   SortStructure: Array(0..ItemKnt) of JobElement;
   Pointer: NodePointer;
   Remainder, RKnt, Kntr, F, R, K: Integer;
   Precedent, Successor : SortElement;

   function ITP is new Ada.Unchecked_Conversion(Integer, NodePointer);

   ----------Procedure for TopologicalSort----------
   procedure TopologicalSort is
   begin     --Get the number of actions (task to be completed), NA;
      --Step 1--
      for K in 1..ItemKnt   --For K in 1.. NA begin
      loop
         SortStructure(K).Knt := 0;   --Count[K] <- 0;
         SortStructure(K).Top := null;   --Top[K] <- null
      end loop;

      Remainder := ItemKnt;    --Set KN <- NA where KN is the number of actions still to be processed.
      Kntr := 1;
      put_line("Assuming at least 1 relation, how many relations are there? ");
      get(RKnt);
```

```
  --Step 2--
  for K in 1..RKnt
  loop
    Pointer := new Node;
    put_line("Enter the Precedent (J) followed by its Successor (K) ");   --Asking for user input
    put(K);
    put_line(":");   --Relation number
    get(Precedent); put_line(" < "); get(Successor);  --Take user input
    new_line(1);
    SortStructure(SortElement'Pos(Successor)).Knt :=
      SortStructure(SortElement'Pos(Successor)).Knt + 1;   --Increase Count[K] by one;
    Pointer.Suc := Successor;   --Set P <= Avail
    Pointer.Next := SortStructure(SortElement'Pos(Precedent)).Top;   --Set P.Next <- Top[J]
    SortStructure(SortElement'Pos(Precedent)).Top := Pointer;   --Top[J] <- P
    exit when Kntr = RKnt;   --Repeat until out of transactions in the input

    Kntr := Kntr + 1;
  end loop;

  --Step 3--
  R := 0;   --Set R <- 0
  SortStructure(0).Knt := 0;   --Qlink[0] <- 0

  for K in 1..ItemKnt  --for K in 1.. NA loop
  loop
    if
      (SortStructure(K).Knt = 0)   --If Count[K] = 0 then
    then
       SortStructure(R).Knt := K;   --Qlink[R] <- K;
       R := K;   --R <- K;
    end if;
  end loop;
  F := SortStructure(0).Knt;   --F <- Qlink[0];

  put_line("Here is the result: ");
end TopologicalSort;

----------Procedure for Result----------
procedure Result is
begin
  --Step 4--
  while (F /= 0)   --While F not = 0 loop
```

```
    loop
      put(SortElement'Val(F));   --Perform action F
      new_line(1);
      Remainder := Remainder - 1;   --Set KN <- KN - 1
      Pointer := SortStructure(F).Top;   --P <- Top[F]
      SortStructure(F).Top := ITP(0);

      while (Pointer /= null)   --While P not = null loop
      loop
        SortStructure(SortElement'Pos(Pointer.Suc)).Knt :=
          SortStructure(SortElement'Pos(Pointer.Suc)).Knt - 1;    --Count[Suc(P)] =
Count[Suc(P)] - 1
        if SortStructure(SortElement'Pos(Pointer.Suc)).Knt = 0   --If Count[Suc(P)] = 0 then
        then
          SortStructure(R).Knt := SortElement'Pos(Pointer.Suc);   --Qlink[R] <- Suc[P]; {add to
output queue}
          R := SortElement'Pos(Pointer.Suc);    --R <- Suc[P]
        end if;

        Pointer := Pointer.Next;    --P <- Next[P];
      end loop;

      F := SortStructure(F).Knt;   --F <- Qlink[F]
    end loop;

    --Step 5--
    if
      (Remainder = 0)
    then   --If KN = 0, the topological sort has been completed successfully.
      put_line("This is a Topological Sort");
    else
      put_line("There is a loop causing problems: ");
      new_line(1);

      for K in 1..ItemKnt   --For K in 1 .. NA begin
      loop
        SortStructure(K).Knt := 0;   --QLink[K] <- 0
      end loop;

      --Step 6--
      for K in 1..ItemKnt   --For K in 1 .. NA loop
      loop
        Pointer := SortStructure(K).Top;   --P <- Top[K];
```

```
      SortStructure(K).Top := ITP(0);   --Top[K] <- 0
      while (Pointer /= ITP(0) and then   --While P <> 0 and
             (SortStructure(SortElement'Pos(Pointer.Suc)).Knt = 0))   --Qlink[Succ(P)] = 0 loop
      loop
        SortStructure(SortElement'Pos(Pointer.Suc)).Knt := K;   --Qlink[Succ(P)] <- K
        if (Pointer /= ITP(0))   --If P <> 0 then
        then
          Pointer := Pointer.Next;   --P <- Next(P)
        end if;
      end loop;
    end loop;
    --At this point, QLink[K] will point to one of the predecessors of
    --action K for each action K that has not yet been processed

    --Find a K with QLink[K] not = 0.  This will be part of the loop.
    --Step 7--
    K := 1;   --K <- 1
    while (SortStructure(K).Knt = 0)   --while (QLink[K] = 0) loop
    loop
      K := K + 1;   --K <- K + 1
    end loop;

    --{Look for loop and mark it.}
    --Step 8--
    loop
      SortStructure(K).Top := ITP(1);  --Top[K] <- 1
      K := SortStructure(K).Knt;   --K <- QLink[K]
      exit when (SortStructure(K).Top /= ITP(0));   --Until Top[K] not = 0
    end loop;

    --{Print the loop.}
    --Step 9--
    while (SortStructure(K).Top /= ITP(0))   --While Top[K] not = 0 loop
    loop
      put(SortElement'Val(K)); put(" <-");   --Print {process} action K
      SortStructure(K).Top := ITP(0);   --Top[K] <- 0;
      K := SortStructure(K).Knt;   --K <- QLink[K];
    end loop;

    put(SortElement'Val(K));

    new_line(1);
  end if;
```

```
   end Result;
end GenericTopologicalSort;  --End of program
```

----------**Main.adb**----------

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with GenericTopologicalSort;

----------Procedure for Main----------
procedure Main is

   ItemKnt: Integer;

   package MyInt_IO is new Ada.Text_IO.Integer_IO(Integer);

   procedure IntGet(Action: out Integer) is  --Overload definition for sRocha parameter
   begin
      MyInt_IO.Get(Action);
   end IntGet;

   procedure IntPut(Action: in Integer) is   --Overload definitions for sRocha parameter
   begin
      MyInt_IO.put(Action);
   end IntPut;

   --------Integer Sort----------
begin
   put_line("Enter the number of jobs: ");   --Asking for user input
   get(ItemKnt);


   declare   --Declare package for Number Sort
      package NumberSort is new
        GenericTopologicalSort(Integer, IntGet, IntPut, ItemKnt);
      use NumberSort;

   begin

      NumberSort.TopologicalSort;   --Call Number Sort TopoSort procedure

      NumberSort.Result;   ----Call Number Sort Result procedure

   end;
end Main;
```