Rutger Yager

Dr. Burris COSC 3319.01

11/8/2013 GRADE OPTION “A+”

**INPUT AND RESULTS**

**Lab3in.txt:**

integer

11

16

3

4

3

4

11

9

6

1

8

7

10

9

8

3

6

7

8

10

2

3

5

4

2

7

5

1

6

9

11

1

5

6

integer

11

20

11

1

6

1

8

7

10

9

6

9

3

4

6

7

8

10

2

3

5

4

8

3

2

7

5

1

5

6

4

8

3

5

2

5

11

9

1

6

1

2

names

9

11

Garza

Taylor

Funk

Taylor

Sido

Dedear

Rocha

McLeod

McLeod

Dedear

Taylor

Sido

Funk

Sido

Reioux

Dedear

Davies

Funk

Funk

Dedear

Funk

Taylor

names

9

12

Garza

Taylor

Funk

Taylor

Sido

Dedear

Rocha

McLeod

McLeod

Dedear

Taylor

Sido

Funk

Sido

Reioux

Dedear

Davies

Funk

Funk

Dedear

Funk

Taylor

Dedear

Funk

done

**Lab3.out.txt:**

This file was generated with the command: topoclasstest < Lab3in.txt > Lab3out.txt

integer or names?

Enter Number of Actions:

Partial Order: Action1 < Action2 (Action1 comes before Action2).

Enter number of Partial Orders:

Action1: Action2: Added 3 before 4.

Action1: Action2: Added 3 before 4.

Action1: Action2: Added 11 before 9.

Action1: Action2: Added 6 before 1.

Action1: Action2: Added 8 before 7.

Action1: Action2: Added 10 before 9.

Action1: Action2: Added 8 before 3.

Action1: Action2: Added 6 before 7.

Action1: Action2: Added 8 before 10.

Action1: Action2: Added 2 before 3.

Action1: Action2: Added 5 before 4.

Action1: Action2: Added 2 before 7.

Action1: Action2: Added 5 before 1.

Action1: Action2: Added 6 before 9.

Action1: Action2: Added 11 before 1.

Action1: Action2: Added 5 before 6.

Perform action 2

Perform action 5

Perform action 8

Perform action 11

Perform action 6

Perform action 10

Perform action 3

Perform action 7

Perform action 1

Perform action 9

Perform action 4

Successful Sort!

integer or names?

Enter Number of Actions:

Partial Order: Action1 < Action2 (Action1 comes before Action2).

Enter number of Partial Orders:

Action1: Action2: Added 11 before 1.

Action1: Action2: Added 6 before 1.

Action1: Action2: Added 8 before 7.

Action1: Action2: Added 10 before 9.

Action1: Action2: Added 6 before 9.

Action1: Action2: Added 3 before 4.

Action1: Action2: Added 6 before 7.

Action1: Action2: Added 8 before 10.

Action1: Action2: Added 2 before 3.

Action1: Action2: Added 5 before 4.

Action1: Action2: Added 8 before 3.

Action1: Action2: Added 2 before 7.

Action1: Action2: Added 5 before 1.

Action1: Action2: Added 5 before 6.

Action1: Action2: Added 4 before 8.

Action1: Action2: Added 3 before 5.

Action1: Action2: Added 2 before 5.

Action1: Action2: Added 11 before 9.

Action1: Action2: Added 1 before 6.

Action1: Action2: Added 1 before 2.

Perform action 11

Error!

Loop Found: 1 -> 5 -> 2 -> 1

integer or names?

Enter Number of Actions:

Partial Order: Action1 < Action2 (Action1 comes before Action2).

Enter number of Partial Orders:

Action1: Action2: Added GARZA before TAYLOR.

Action1: Action2: Added FUNK before TAYLOR.

Action1: Action2: Added SIDO before DEDEAR.

Action1: Action2: Added ROCHA before MCLEOD.

Action1: Action2: Added MCLEOD before DEDEAR.

Action1: Action2: Added TAYLOR before SIDO.

Action1: Action2: Added FUNK before SIDO.

Action1: Action2: Added REIOUX before DEDEAR.

Action1: Action2: Added DAVIES before FUNK.

Action1: Action2: Added FUNK before DEDEAR.

Action1: Action2: Added FUNK before TAYLOR.

Perform action GARZA

Perform action ROCHA

Perform action REIOUX

Perform action DAVIES

Perform action MCLEOD

Perform action FUNK

Perform action TAYLOR

Perform action SIDO

Perform action DEDEAR

Successful Sort!

integer or names?

Enter Number of Actions:

Partial Order: Action1 < Action2 (Action1 comes before Action2).

Enter number of Partial Orders:

Action1: Action2: Added GARZA before TAYLOR.

Action1: Action2: Added FUNK before TAYLOR.

Action1: Action2: Added SIDO before DEDEAR.

Action1: Action2: Added ROCHA before MCLEOD.

Action1: Action2: Added MCLEOD before DEDEAR.

Action1: Action2: Added TAYLOR before SIDO.

Action1: Action2: Added FUNK before SIDO.

Action1: Action2: Added REIOUX before DEDEAR.

Action1: Action2: Added DAVIES before FUNK.

Action1: Action2: Added FUNK before DEDEAR.

Action1: Action2: Added FUNK before TAYLOR.

Action1: Action2: Added DEDEAR before FUNK.

Perform action GARZA

Perform action ROCHA

Perform action REIOUX

Perform action DAVIES

Perform action MCLEOD

Error!

Loop Found: FUNK -> DEDEAR -> FUNK

integer or names?

**SOURCE FILES**

**Topoclasstest.adb:**

with TopoClass;

with Ada.Text\_IO;

use Ada.Text\_IO;

with Ada.Integer\_Text\_IO;

use Ada.Integer\_Text\_IO;

with Ada.Strings.Unbounded;

use Ada.Strings.Unbounded;

with Ada.Strings.Unbounded.Text\_IO;

use Ada.Strings.Unbounded.Text\_IO;

procedure TopoClassTest is

type MyType is

(Nothing,

Garza,

Taylor,

Funk,

Sido,

Dedear,

Rocha,

McLeod,

Reioux,

Davies);

package MyTypeIO is new Ada.Text\_IO.Enumeration\_IO(MyType);

procedure IntGet(Thing : out Integer) is

begin

get(Thing);

end IntGet;

procedure MyTypeGet(Thing : out MyType) is

begin

MyTypeIO.Get(Thing);

end MyTypeGet;

procedure IntPut (

Thing : in Integer) is

begin

Put(Thing, 2);

end IntPut;

procedure MyTypePut (

Thing : in MyType) is

begin

MyTypeIO.Put(Thing);

end MyTypePut;

package IntTopo is new TopoClass(Integer, IntGet, IntPut);

package NameTopo is new TopoClass(MyType, MyTypeGet, MyTypePut);

Input : Unbounded\_String;

begin

loop

put\_line("integer or names? ");

Input := Get\_Line;

if Input = "integer" then

IntTopo.SortDemo;

elsif Input = "names" then

NameTopo.SortDemo;

elsif Input = "done" then

exit;

end if;

end loop;

end TopoClassTest;

**TopoClass.ads:**

generic

type ITEM is

(<>);

with procedure Get (

Thing : out ITEM);

with procedure Put (

Thing : in ITEM);

package TopoClass is

type Topological\_Sort

(Num : Integer;

Arrayend : ITEM) is tagged private;

function GenerateTopo (

Number : Integer)

return Topological\_Sort;

procedure AddPartialOrder (

Topo : in out Topological\_Sort;

Before : in ITEM;

After : in ITEM);

procedure PrintTopo (

Topo : in Topological\_Sort);

procedure BuildTopo (

Topo : in out Topological\_Sort);

procedure TopologicalSort (

Topo : in Topological\_Sort);

procedure SortDemo;

private

type Node;

type NodePt is access Node;

type Node is

record

Info : ITEM;

Next : NodePt;

end record;

type Int\_Array is array (ITEM range <>) of Integer;

type Item\_Array is array (ITEM range <>) of ITEM;

type NodePt\_Array is array (ITEM range <>) of NodePt;

type Topological\_Sort

(Num : Integer;

ArrayEnd : ITEM) is tagged

record

Na : Integer := Num;

Count : Int\_Array (ITEM'Val (0) .. ArrayEnd) := (others => 0);

Top : NodePt\_Array (ITEM'Val (0) .. ArrayEnd) := (others => null);

end record;

end TopoClass;

**TopoClass.adb:**

with Ada.Text\_IO;

use Ada.Text\_IO;

with Ada.Integer\_Text\_IO;

use Ada.Integer\_Text\_IO;

with Ada.Strings.Unbounded.Text\_IO;

use Ada.Strings.Unbounded.Text\_IO;

package body TopoClass is

function GenerateTopo (

Number : Integer)

return Topological\_Sort is

Temp : Topological\_Sort (Number, ITEM'Val (Number));

begin

return Temp;

end GenerateTopo;

procedure AddPartialOrder (

Topo : in out Topological\_Sort;

Before : in ITEM;

After : in ITEM) is

begin

Topo.Count(After) := Topo.Count(After) + 1;

Topo.Top(Before) := new Node'(

Info => After,

Next => Topo.Top (Before));

end AddPartialOrder;

procedure PrintTopo (

Topo : in Topological\_Sort) is

P : NodePt;

Count : Integer;

begin

for I in ITEM'Val(0)..ITEM'Val(Topo.NA) loop

Put(I);

Put(": Count(");

Put(Topo.Count(I), 2);

Put(") ");

P := Topo.Top(I);

Count := 0;

while Count <= 6 and then P /= null loop

Put(P.Info);

Put(" -> ");

P := P.Next;

Count := Count + 1;

end loop;

Put\_Line("null");

end loop;

end PrintTopo;

procedure BuildTopo (

Topo : in out Topological\_Sort) is

J : ITEM;

K : ITEM;

Input : Integer;

begin

Put\_Line("Partial Order: Action1 < Action2 (Action1 comes before Action2).");

Put\_Line("Enter number of Partial Orders: ");

Get(Input);

Skip\_Line;

--loop

while Input > 0 loop

begin

Put("Action1: ");

Get(J);

Skip\_Line;

Put("Action2: ");

Get(K);

Skip\_Line;

AddPartialOrder(Topo, J, K);

put("Added "); put(J); put(" before "); put(K); put\_line(".");

Input := Input - 1;

exception

when Data\_Error =>

Put\_Line("Invalid: Value not acceptable.");

end;

end loop;

end BuildTopo;

procedure TopologicalSort (

Topo : in Topological\_Sort) is

K : Integer;

F : Integer;

R : Integer;

KN : Integer;

P : NodePt;

Temp : Topological\_Sort := Topo;

begin

KN := Temp.NA;

R := 0;

Temp.Count(ITEM'Val(0)) := 0;

for I in 1..Temp.NA loop

if Temp.Count(ITEM'Val(I)) = 0 then

Temp.Count(ITEM'Val(R)) := I;

R := I;

end if;

end loop;

F := Temp.Count(ITEM'Val(0));

while F /= 0 loop

Put("Perform action ");

Put(ITEM'Val(F));

New\_Line;

KN := KN - 1;

P := Temp.Top(ITEM'Val(F));

Temp.Top(ITEM'Val(F)) := null;

while P /= null loop

Temp.Count(P.Info) := Temp.Count(P.Info) - 1;

if Temp.Count(P.Info) = 0 then

Temp.Count(ITEM'Val(R)) := ITEM'Pos(P.Info);

R := ITEM'Pos(P.Info);

end if;

P := P.Next;

end loop;

F := Temp.Count(ITEM'Val(F));

end loop;

if KN = 0 then

Put\_Line("Successful Sort!");

else

Put\_Line("Error!");

for I in 1..Temp.NA loop

Temp.Count(ITEM'Val(I)) := 0;

end loop;

for I in 1..Temp.NA loop

P := Temp.Top(ITEM'Val(I));

Temp.Top(ITEM'Val(I)) := null;

while P /= null loop

if Temp.Count(P.Info) = 0 then

Temp.Count(P.Info) := I;

end if;

P := P.Next;

end loop;

end loop;

K := 1;

while Temp.Count(ITEM'Val(K)) = 0 loop

K := K + 1;

end loop;

loop

Temp.Top(ITEM'Val(K)) := new Node'(

Info => ITEM'Val (0),

Next => null);

K := Temp.Count(ITEM'Val(K));

exit when Temp.Top(ITEM'Val(K)) /= null;

end loop;

Put("Loop Found: ");

while Temp.Top(ITEM'Val(K)) /= null loop

Put(ITEM'Val(K));

Put(" -> ");

Temp.Top(ITEM'Val(K)) := null;

K := Temp.Count(ITEM'Val(K));

end loop;

Put(ITEM'Val(K));

New\_Line;

end if;

end;

procedure SortDemo is

NA : Integer;

begin

Put\_Line("Enter Number of Actions: ");

Get(NA);

Skip\_Line;

declare

TopoTest : Topological\_Sort := GenerateTopo(NA);

begin

Buildtopo(TopoTest);

TopologicalSort(TopoTest);

end;

end SortDemo;

end TopoClass;