**CS3319 Lab 3 Spring 2012 Burris**

Due: November 2, 2012. Be sure to specify the grading option you complete or your work will be evaluated according to the “C” option grading criteria.

**“C” Option (best grade is 70):**

Implement the simple version of the topological sort algorithm. Sort the following relations: 5<4, 6<9, 3<2, 5<7, 5<3, 4<6, 6<3, 6<1, 8<9, 7<3, 2<9, 5<4. Note the data contains duplicate relations. Your program should not be affected by duplicate data except for additional run time and space. Your output should clearly indicate if no solution exists.

Now process the following relations: 5<4, 6<9, 3<2, 5<7, 3<5, 4<6, 6<3, 6<1, 8<9, 7<3, 2<9, 5<4.

**“B” Option (best grade is 80):**

Implement the topological sort algorithm. If a loop (no solution) is encountered, print the actions that make up the loop. Process the data for the “C” option. You need not implement the “C” Option program. You must get at least one loop. Did you get all possible loops? If you can find multiple (preferably all) loops, brag on yourself.

**“A” Option (best grade is 90):**

Allow the user to specify actions using any programmer defined enumeration data type they desire using generic instantiation. Process the following data in addition to the “C” option data. A partial specification follows. The “A” option must print the contents of a loop if encountered. **You must use OOP to receive credit utilizing generics** (templates). Treat the following names appearing in the partial ordering as an enumeration type.

Additional Data: Henderson < Acevedo, Leky < Acevedo, Smith < Utsey, Sanchez < Mills, Mills < Utsey, Acevedo < Smith, Leky < Smith, Pruett < Utsey, Cleek < Leky, Leky < Utsey, and Leky < Acevedo.

Additional Data: Henderson < Acevedo, Leky < Acevedo, Smith < Utsey, Sanchez < Mills, Mills < Utsey, Acevedo < Smith, Leky < Smith, Pruett < Utsey, Cleek < Leky, Leky < Utsey, Leky < Acevedo, and Utsey < Leky.

Hint for A option: See the generic circular queue pages 54-55 (approximately) of DataStructuresPgms.doc. In the program notes a method, GIOEX, is demonstrated to pass generic I/O routines to a generic package. Assume a partial order of the form JobA < JobB read as JobA precedes JobB. **To receive full credit you must pass the I/O routine to the sort program (which does the printing).**

-- **General Structure:**

generic **-- You may modify this as required but observe the spirit.**

type SortElement is private; -- An element J (or K) of the partial ordering

-- J < K processed by the topological sort. J and K represent jobs in the partial ordering.

with procedure get(Job: out SortElement); // Reads J or K.

with put(Job: in SortElement); // Print the value of J or K.

package GenericTopologicalSort is

TopologicalSort;

-- additional procedures/functions to export if required

end GenericTopologicalSort;

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

Count: Integer := 0;

Top: NodePointer;

end record;

SortStructure: Array(SortElement) of JobElement;

-- other declarations

procedure TopologicalSort is

begin -- Program to obtain the relations in the partial ordering,

-- sort the jobs, and print results;

end TopologicalSort;

end GenericTopologicalSort;

with GenericTopologicalSort;

procedure Main is

type NameType is (Mary, Joe, Tom, Bob, Sara, Julie, Larry, Sam);

package NameTypeIO is new Ada.Text\_IO.Enumeration\_IO(NameType);

use NameTypeIO;

-- Overload definitions for sSanchez parameter “get(Action : out SortElement)”

-- and “put(Action: in SortElement)” for NameTypeIO.

package NameTopologicalSort is new

GenericTopologicalSort(NameType, get, put);

use NameTopologicalSort;

begin

-- rest of program

end Main;

**“A+” Option(best grade is 100):**

Implement the “A” option. You must use the count field to build the linked queue, not a separate array.