CS3319 Topological Sorting Fall 2019 Burris

Due: October 21 for MWF and October 22 for TTH prior to the start of class.

Lab 3 has two major implementation/grading options. The first implementation ("D" and "C") option (less professional) provides an opportunity to master passing functions to functions (methods) and generics using simple data types. The second implementation option (professional grade "B" and "A") offers the opportunity to master generics, inheritance, function overloading, operator overloading, passing functions and operator overloads, polymorphism and creating linked list of heterogeneous objects. As usual, you indicate your professional worth by your selection of implementation/grading option.

You must include a cover sheet stating your name, class and section, date submitted and grading option. Process all precedence relations in the order specified!

"D" Option (best grade is 65):

Implement the simple version of the topological sort algorithm. Sort the following relations in the order shown: 1<2, 1<3, 4<1, 3<8, 8<2, 4<2, 4<5, 6<4, 5<7, 2<7, 9<8, 9<6, 2<7, 4<2 and 9 < 8. Note that 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. You must implement your code as a package/class.

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

Submit the cover sheet first followed by the output of the first data set. The output of the second data set should next followed by the "D" option code. I recommend using files for obtaining relations and printing results in a file. You may use I/O redirection if desired for the "D" and "C" options.

"C" Option (best grade is 75):

Do not do the "D" Option. Rather implement the topological sort algorithm printing the contents of a loop if no solution is encountered. Process both "D" option data sets. You must implement your code as a package/class.

Submit the cover sheet first followed by the output of the first data set. The output of the second data set should appear next followed by the "C" option code.

“B” Option (best grade is 90):

You need not implement the “D” and “C” Options. Our team has been selected to implement part of a strategy gaming engine for our company’s next major product. The product will utilize a 3-D game board with data displayed after a topological sort has been completed identifying at least one critical path to complete the current level of the game. If no path exist then we must identify the offending relations creating a loop for the user. A combination of game and user precedence restrictions on resource allocation plays a major role in the game.

You must explicitly use text files for input and output for the “B” and “A” Options (open and close files). <u>You may not use I/O redirection for the “B” and “A” Options or enter data by hand</u>.

Players will be encouraged to extend the game with custom tokens/objects. Tokens must adhere to some basic rules including methods to read and print tokens. <u>Tokens used in the game may be heterogeneous as opposed to homogeneous</u>.

<u>Implement the “B” option topological sort using generics, single inheritance, function overloading, passing I/O routines to other functions, and polymorphism</u>. The software to perform the topological sort **must be able to sort heterogeneous objects. Hence the requirement to implement a heterogeneous stack of objects waiting on another object to be processed**. Management insists you use the design format illustrated in the hymnal “Data Structures Programs” pages 60 through 68 allowing homogeneous and heterogeneous linked list for the stacks.

Hint See the “gstack” and “SMailbox” pages 54-56 of DataStructuresPgms.doc for examples of generics and default generic values in Ada. In the program notes pages 76-78 GIOEX demonstrated to passing I/O routines to a generic package in Ada.

<u>To receive credit you must pass the I/O routine to read precedence relations and print the results of the sort as generic parameters!</u>.

Data set 1 “B” Option:

Mary < Tom, Tom < Bob, Tom < Sam, Joe < Sam, Sam < Betty, and Mary < Sam, Bob < Betty, Joe < Betty

Data set 2 “B” Option:

Mary < Tom, Tom < Bob, Joe < Tom, Sam < Joe, Tom < Sam, Joe < Sam, Sam < Betty, Mary < Sam, Bob < Betty, Joe < Betty and Betty < Tom

Data set 3 “B” Option using the vehicle make/fruit/people type fields:

(GMC, 4) → (Ford, 2)

(Burris, wizard) → (GMC,4)
(Ford, 2) → (Dodge, 3)
(Dodge, 3) → (orange, 56.2)
(apple, 30.0) → (Pear, 40.6)
(Ford, 2) → (Apple → 30.0)
(Joe, sorcerer) → (orange, 56.2)
(apple, 30.0) → (banana, 45.0)
(Bennett, warrior) → (GMC, 4)
(Bennett, warrior) → (Joe, sorcerer)
(banana, 45.0)→ (orange, 56.2)
(pear, 40.6) → (orange, 56.2)
(Bennett, warrior) → ((Dodge,3)
(Burris, wizard) → (apple, 30.0)
(apple, 30.0) → (banana, 45.0)

Data set 3 "B" Option:
(GMC, 4) → (Ford, 2)
(Burris, wizard) → (GMC,4)
(Ford, 2) → (Dodge, 3)
(Dodge, 3) → (orange, 56.2)
(apple, 30.0) → (Pear, 40.6)
(Ford, 2) → (Apple → 30.0)
(Joe, sorcerer) → (orange, 56.2)
(apple, 30.0) → (banana, 45.0)
(Bennett, warrior) → (GMC, 4)
(Bennett, warrior) → (Joe, sorcerer)
(banana, 45.0)→ (orange, 56.2)
(pear, 40.6) → (orange, 56.2)
(Bennett, warrior) → (Dodge,3)
(Burris, wizard) → (apple, 30.0)
(apple, 30.0) → (banana, 45.0)
(Burris, wizard) → (Bennett, warrior)
(Bennett, warrior) → (Ford, 2)

First process the "D" data sets for in the order they are defined. Next process the "B Option data sets in the order defined.

Submit the cover sheet first followed by the "D" results in the order specified in the lab followed by the "B" data sets in the order specified in the lab. Next include a copy of the "B" Option code. The main program should instantiate your package to process all data sets in the specified order.

Hint: Ada "unchecked_conversion" is useful for all options with respect to treating the count field as pointers (links).

Best Wishes,
Management

“A” Option (best grade is 100):

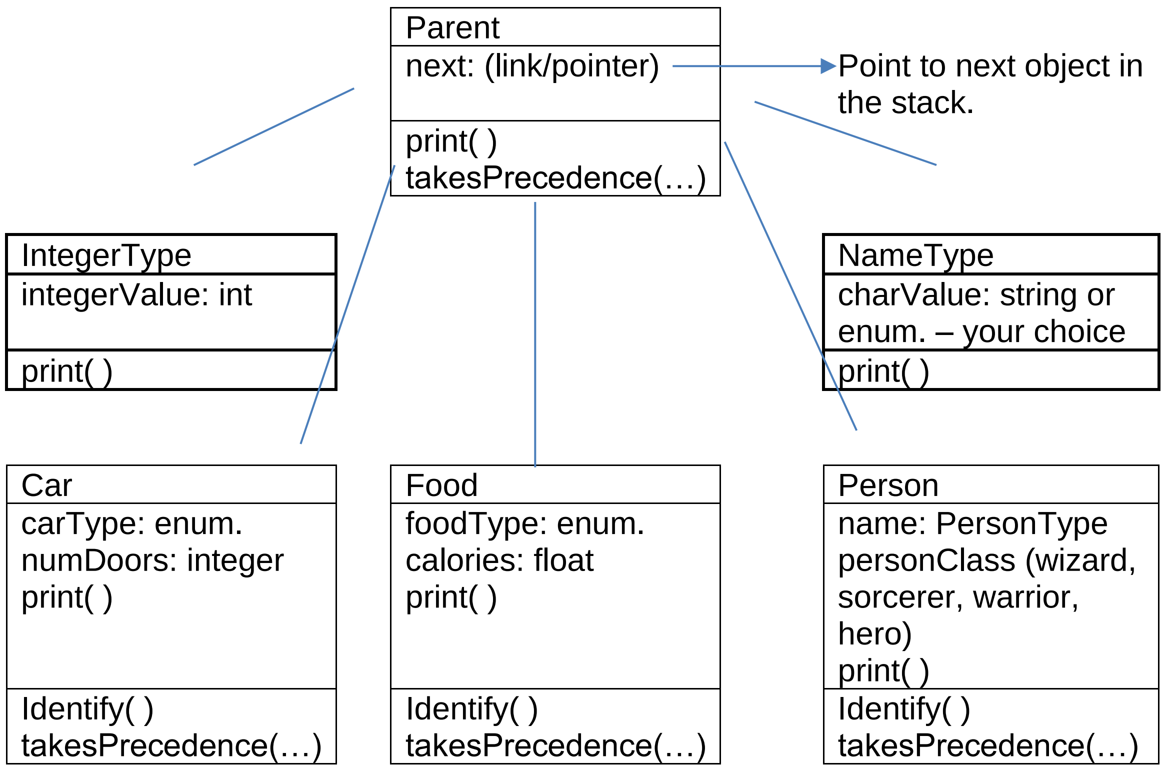
To make a grade higher than 90 you must either find all solutions; all loops; or all solutions and loops subject to the “B” option requirements. Process the data for the “D” and “B” option data sets to demonstrate your code works properly. Hint: The key to finding all loops/ or all solutions may be found in “Fundamental Algorithms” by Donald Knuth. It is in the university library. The secret was mentioned in class.

First process the “D” data sets in the order they are defined. Next process the “B Option data sets in the order defined.

Submit the cover sheet first followed by the “D” results in the order specified in the lab followed by the “B” data sets in the order specified in the lab. Next include a copy of the “B” Option code. The main program should instantiate your package to process all data sets in the specified order.

Design Restrictions for “B” and “A” Options:

The class “Parent” is to be used in creating objects to be placed in the heterogeneous stacks. Note all objects inherit from the common parent “Parent.” The class “IntegerType” should allow the user to process the “D” Option data. The “NameType” should allow for processing part of the “B” Option data.



The “print” function should be polymorphic and print all available information for the object. All sorts will utilize the “enumeration type” contained in the body of the child.

Currently the “enumeration type” is defined as follows for the existing Tokens.

integerValue: A positive integer.
charType: A character or enumeration type, your choice
foodType: apple, banana, orange, pear.
nameType: Bennett, Burris, Cooper, Joe, (enum or string, you pick)
carType: GMC, Chevy, Ford, Buick, Jeep, Dodge (enum or string, you pick)
personType:
personClass: warrior, sorcerer, wizard (enum or string, you pick)

<u>Ideally the array created by the topological sort algorithm with count fields and list heads should be allocated in the system stack. The objects and linked list should appear in the heap. Impress management by returning nodes (objects) to an</u> available storage list when no longer needed. Use a highlighter to bring this section of your code to my attention if accomplished.

General Structure of Topological Sort Routing and Main Program using one data type.

```
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 objects in the partial ordering.

**       with function get(Job:  out SortElement) return SortElement;  // Reads J or K.
**       with procedure put(Job:  in SortElement);  // Print the value of J or K.
**       with function takesPrecedence(…);

package GenericTopologicalSort is
         procedure 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;  -- Sort element is an "object" placed in the stack.
                  Next:    NodePointer;
         end record;

         type JobElement is record
                  Count:  Integer := 0;  -- This field should be used for counting and as queue links.
                  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 sRocha 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;
```