CS3319 Lab 3 Spring 2016 Burris

Due: Friday March 11. This lab will not be accepted for credit after Friday March 18.

Lab 3 has two major implementation/grading options. The first implementation 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) 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 implementation option for the lab and grading option within the implementation option.

Implementation Option 1:

"D" Option (best grade is 65):

Implement the simple version of the topological sort algorithm. Sort the following relations in the order shown: 6 < 5, 5 < 1, 2 < 6, 3 < 2, 10 < 4, 3 < 4, 8 < 7, 1 < 8, 6 < 1, 4 < 6, 5 < 1, 4 < 6 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 package/class.

Now process the following relations: 6 < 5, 5 < 1, 2 < 6, 1 < 5, 3 < 2, 10 < 4, 3 < 4, 8 < 7, 7 < 9, 1 < 8, 6 < 1, 4 < 6, 5 < 1, 4 < 6, 8 < 6 and 9 < 8.

"C" Option (best grade is 70):

Do not do the "C" Option. Implement the topological sort algorithm printing the contents of a loop (no solution) if no solution is encountered, print the actions that make up the loop. Process the data for the "C" option. You must get at least one loop. Did you get all possible loops? If you can find multiple (preferably all) loops, brag on yourself. It is worth a celebration. You must implement your code as package/class.

"B" Option (best grade is 80 to 85):

Implement the "C: option allowing the user to specify actions using **any programmer defined <u>enumeration</u> data type** they desire using generic instantiation. Process the following data in addition to the "C" option data. A partial specification in Ada follows. To receive a grade of 85 you must print the contents of a loop if encountered. In addition to **utilizing generics** (templates**) you must pass I/O routines from the main program in Ada, C++, or Java. If you use "C/C++" or Java you will need to master function pointers.**

Sort the following in the order shown:
Additional Data: Tom < Bob, Tom < Sam, Joe < Sam, Sam < Betty, and Mary < Sam

Additional Data: Tom < Bob, Tom < Sam, Joe < Sam, Sam < Betty, and Mary < Sam, Betty < Joe, Betty < Mary

Hint 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)**.

"B+" Option (best grade is 89):
To make a grade higher than 80 you must either find all solutions, all loops, or all solutions and loops.

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 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;
```

Implementation Option 2:

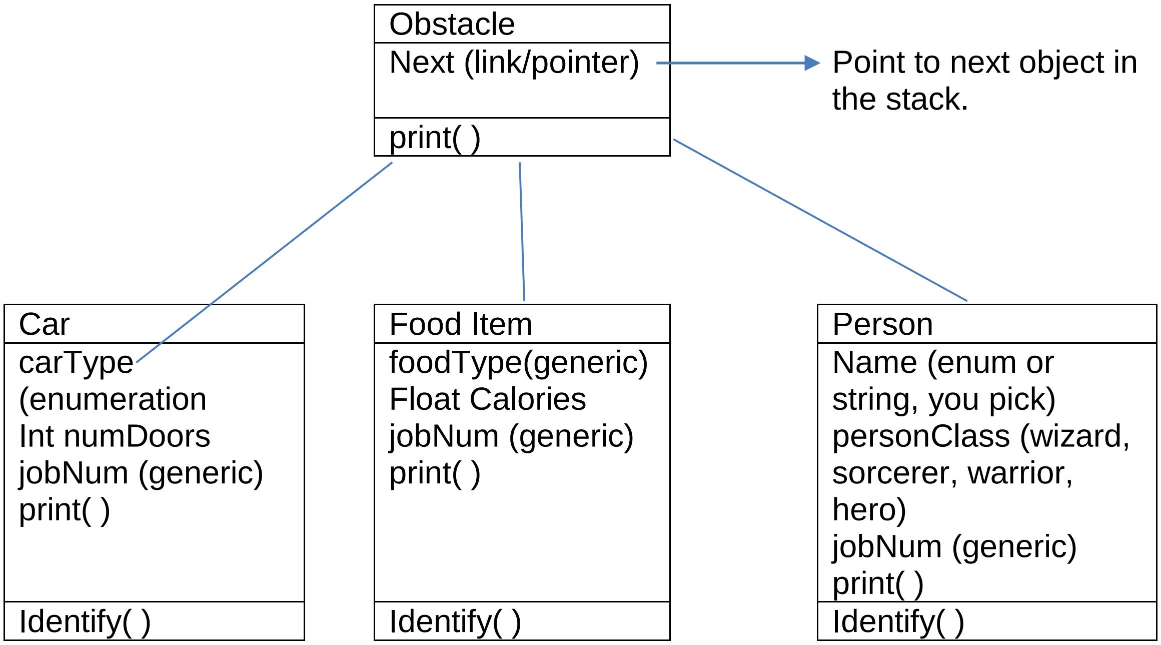
“A” Option (best grade is 90):
Implement the “B” option of Implementation Option 1 using generics, single inheritance, function overloading, passing I/O routines to other functions, and polymorphism. Rather than limiting ourselves to enumeration types however, **management would like to be able to sort heterogeneous objects inserting complete objects into the stack of objects waiting on another object to be completed**. Management insists you use the design format illustrated in the hymnal “Data Structures Programs” pages 63 through 68 allowing homogeneous and heterogeneous linked list.

Hint: Ada “unchecked_conversion,” or COBOL “renames” may be useful for all options with respect to treating the count field as pointers (links).

This material will be used to build a gaming engine by our company for distributing obstacles on a 3-D game board. Players will need to identify at least one of the critical paths to complete each level of the game. The requirements and resulting definition requires the following structure and methods at a minimum. You may add additional fields and methods as desired.

Hint: You may find it convenient to utilize Ada “unchecked_conversion,” or COBOL “renames” for all options.

Best Wishes,
Management



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

In addition to the above classes, you will require child classes (inheriting from Obstacle) to hold integers and names as described in Implementation Option 1.

First sort the “D” data sets for Implementation 1 using the integers as the values of the “jobNum” filed. Next use the names in the “B” option data sets as the values of the “jobNum” field.

After sorting the above data sets, sort the following data sets in the order shown:

Set 1:
(Burris, male, wizard, 2) < (apple, 50, 3), (Ford, 4, 5) < (apple, 50, 3),
(apple, 50, 3) < (Cooper, male, warrior, 6), (Ford, 4, 5) < (Cooper, male, warrior),
(Ford, 4, 5) < (Jo, female, sorcerer, 4),
(Jo, female, sorcerer, 4) < (Bennett, male, hero, 7),
(Cooper, male, warrior, 6) < (Bennett, male, hero, 7),
(Cooper, male, warrior,6) < (GMC, 2, 1)

Set 2:
(Burris, male, wizard, 2) < (apple, 50, 3), (Ford, 4, 5) < (apple, 50, 3),
(apple, 50, 3) < (Cooper, male, warrior, 6), (Ford, 4, 5) < (Cooper, male, warrior),
(Ford, 4, 5) < (Jo, female, sorcerer, 4),
(Jo, female, sorcerer, 4) < (Bennett, male, hero, 7),
(Cooper, male, warrior, 6) < (Bennett, male, hero, 7),
(Cooper, male, warrior,6) < (GMC, 2, 1), (Bennett, male, hero, 7) < (Ford, 4, 5)
(GMC, 2, 1) < (Burris, male, wizard, 2)

“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.

Hint: You may find it convenient to utilize Ada “unchecked_conversion,” or COBOL “renames” for all options to utilize a field for more than 1 data type..