UNIVERSITY OF SASKATCHEWAN
DEPARTMENT OF COMPUTER SCIENCE

CMPT 250
Final Examination

3 hours **Closed Book** **April 28, 2003**
Marks

This exam is out of 150. Thus, you have 1 minute per mark with 30 minutes to spare. Use this in judging how much time to spend on a question.

---

1. Suppose that it is decided to implement a queue by storing the items in a linked list. This can be done by means of a client-supplier relationship or by means of inheritance.

8 (a) Describe the differences between these two approaches in terms of the code written to do the implementation. Note that you do NOT need to write the code for either implementation – just generally describe the code differences.

3 (b) From a strict object-oriented viewpoint, which approach is generally preferred? Briefly justify your answer.

6 2. For the polymorphism design pattern, state the problem to be solved by the pattern, and how the pattern is used to solve the problem.

11 3. Consider an ADT for a sequence, called SEQ[T], where T denotes a generic parameter. Design a reasonable set of operations for such an ADT.
(a) In your specification of the operations, include the sets involved, the signatures for the operations, and their preconditions. It is not necessary to give the axioms for all the operations, but give the build operations and the axioms for 2 of the operations.
(b) Illustrate each operation with CHARACTER used for T.

8 4. Suppose that a dictionary is to be used to store a collection of items. Also, suppose that the decision has been made to implement the dictionary by either a hash table or a height-balanced binary tree. Describe what characteristics of the application should be used to determine which implementation to use. Also, describe how the characteristics are used.

8 5. Two possible types of files are direct and B-tree files. Briefly explain how each file type is organized.

8 6. Prove by mathematical induction that

$$\sum_{i=1}^{n}(3i-2)=\frac{n(3n-1)}{2}$$

10 7. Using the techniques described in class, list the situations that should be tested for the following routine (from the class HT_BAL_BASIC_DICTIONARY_UOS of the UOS data structure library):

```
single_rotate_right (parent : like root_node) is
        --Rotate right in order to balance the tree
        --Analysis: Time = O(1)

    require
        valid_height: (root_left_subtree.height = 2 + root_right_subtree.height)
                and (root_left_subtree.root_left_subtree.height >=
                        root_left_subtree.root_right_subtree.height)

    local new_root_node, new_right_node, new_right_left_node : like root_node

    do
        new_root_node := root_node.left_node
        new_right_node := root_node
        new_right_left_node := root_node.left_node.right_node
        if parent /= Void then
            if parent.left_node = root_node then
                parent.set_left_node(new_root_node)
            else
                parent.set_right_node(new_root_node)
            end
        end
        set_root_node(new_root_node)
        new_right_node.set_left_node(new_right_left_node)
        root_node.set_right_node(new_right_node)
        root_right_subtree.set_height(1 + root_right_subtree.max_subtree_height)
        set_height(1 + max_subtree_height)

    ensure
        height_correct:  (is_empty and height = 0) or else (height = 1 + max_subtree_height)
        height_decrease_at_most_1:  (height = old height) or else (height = old height - 1)
        tree_balanced: is_empty
                or else (root_left_subtree.height - root_right_subtree.height).abs <= 1

    end
```

17 8. Recall that in a weight-balanced tree, each item has an associated positive integer called its count. A weight-balanced tree has two properties. First, it is an ordered binary tree ordered by the item values. Second, it is structured so that the root of each subtree is the item of the subtree with the largest count value, i.e., the count of the item in any node is greater than or equal to the counts of the items in its two children. Give the **Eiffel** code for a BOOLEAN function to be included in a binary tree class that tests whether the current tree is weight balanced. Note that you do not need to test whether the tree is properly ordered – just test whether it is properly weight balanced. Assume that the tree class is a descendant of LINKED_SIMPLE_TREE[G] and has the following features:

make	initialize (lt : like Current; x : G; rt : like Current)
root_item : G	out : STRING
is_empty : BOOLEAN	is_full : BOOLEAN
root_left_subtree : like Current	root_right_subtree : like Current
root_count : INTEGER	

The last feature (root_count) returns the count associated with the item in the root of the tree.

8 9. Consider the directed graph given below. Give the tree that is obtained by a depth-first search of the graph **starting at vertex 4**. Also, for each non-tree edge, indicate whether it is a forward edge, a back edge, or a cross edge as defined in assignment #9.

1: 3
2: 5 9 7
3: 6 8 5
4: 3 5 2
5: 8
6: 1
7: 3
8: 1 6 5
9: 7 4 1

13 10. An undirected graph G=(V, E) is called a bipartite graph if the set of vertices can be partitioned into 2 sets, call them A and B, such that all edges have one end in A and the other end in B. An example of a bipartite graph is given below. Give a detailed algorithm to test whether a graph is bipartite or not. You can assume that the graph is connected. This algorithm should be based on either a breadth-first search or a depth-first search. You should assume that the vertices have a type with the following features:

set_partition (p : STRING) -- mark the vertex as being in partition *p*
partition : STRING -- the partition in which the vertex has been placed

In your detailed algorithm, you can use the following notation:

for each vertex v
. . .

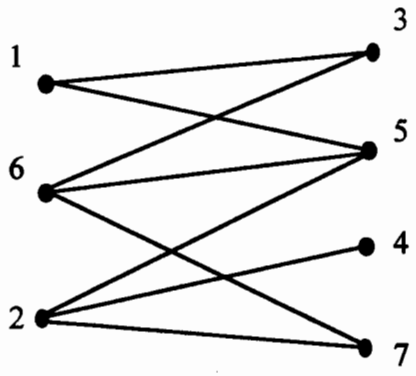
for each vertex u adjacent from vertex w
. . .

or

for each neighbor w of vertex v
. . .

If you wish, you can give Eiffel code instead of a detailed algorithm.

Here is an example of a bipartite graph.



50 11. Royal Service Station Requirements

- The Royal Service Station provides two types of services to its customers: refueling and vehicle maintenance: that is, a customer can add fuel to the tank of his or her vehicle (car, motorcycle, or truck) or can have the vehicle repaired. A customer has the option to be billed automatically at the time of purchase (of fuel or maintenance) or to be sent a monthly paper bill. In either case, customers can pay using cash, credit card, or personal check. Royal Service Station fuel is sold according to price per litre, depending on whether the fuel is diesel, regular, or premium. Repair service is priced according to the cost of parts and labour. The prices for fuel, maintenance services, and parts may vary; only Manny, the station manager, can enter or change prices. At his discretion, Manny may designate a discount on purchases for a particular customer; his discount may vary from one customer to another. A 6% local sales tax applies to all purchases.

- The system must track bills on a month-to-month basis, and the products and services provided by the gas station on a day-to-day basis. The results of this tracking can be reported to the station manager upon request.

- The station manager uses the system to control inventory. The system will warn of low inventory and automatically order new parts and fuel.

- The system will track credit history and send warning letters to customers whose payments are overdue. Bills are sent to customers on the first day of the month after the purchases are made. Payment is due on the first day of the succeeding month. Any bills not paid within 90 days of the billing date will result in the cancellation of the customer's credit.

- The system must handle the data requirements for interfacing with other systems. A credit card system is used to process credit card transactions for products and services. The credit card system requires the card number, name, expiration date, and amount of the purchase. After receiving this information, the credit card system confirms that the transaction is approved or denied. The parts ordering system receives the part code and number of parts needed. It returns the date of parts delivery. The fuel ordering system requires a fuel order description consisting of fuel type, number of litres, station name, and station identification code. It returns the date when the fuel will be delivered.

- The system must record tax and related information, including tax paid by each customer as well as tax per item.

- The station manager must be able to review tax records upon demand.

- The system will send periodic messages to customers, reminding them when their vehicles are due for maintenance. Normally, maintenance is needed every six months.

- The system maintains a repository of account information, accessible by account numbers and by customer name.

- The station manager must be able to review accounting information upon demand.

- The system can report an analysis of prices and discounts to the station manager upon request.

(a) Give a list of the actors and use cases for the Royal Service Station. For this part, no description is required for any use case – just use a meaningful name.

(b) Identify and briefly describe the use case for a customer having some maintenance done on a vehicle (consisting of more than one part and the associated labour costs) that is to be billed at the end of the month. Also, identify and briefly describe what you consider to be the two (2) most important other use cases in the system.

(c) Generate an interaction diagram (either a sequence diagram or a collaboration diagram) for the vehicle maintenance use case given in part (b).

(d) Identify the objects/classes for the Royal Service Station.

(e) Give a composite class diagram for the Royal system.

(f) Describe, using inheritance diagrams (with suitable features), the inheritance taxonomies in your design.

(g) Identify the containers in the system. For each container that would be stored as a file, give its file structure and its primary key.

NOTE: You should make use of suitable 0-0 principles and patterns. The use of these should be identified in your solution. No detailed design is required.

Total 150 **The end**