UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, APRIL 2017

CSC 190 H1S — COMPUTER ALGORITHMS AND DATA STRUCTURES

Exam Type: A — NO calculator allowed
Duration: 2.5 hours
Examiner(s): Pirathayini Srikantha

Student Number: |_|_|_|_|_|_|_|_|_|_|_|

Family Name(s): _____

Given Name(s): _____

Lecture Section: ☐ LEC 01 (Mon 11 am - 12 pm) ☐ LEC 02 (Mon 5 pm - 6 pm)

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully.*

This final examination paper consists of 5 questions on 18 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the final examination is complete and fill in the identification section above.*

Answer each question directly on the examination paper, in the space provided. If you need more space for one of your solutions, use one of the extra "blank" pages at the end of the examination and *indicate clearly the part of your work that should be marked.*

Write up your solutions carefully! In particular, use notation and terminology correctly and explain what you are trying to do—part marks *will* be given for showing that you know the general structure of an answer, even if your solution is incomplete.

When writing code, comments are not required, although these may help us mark your answers.

MARKING GUIDE

1: _____/30
2: _____/15
3: _____/15
4: _____/15
5: _____/15

TOTAL: _____/90

Good Luck!

OVER...

Question 1. [30 MARKS]

This section consists of 20 short answer questions. Please fill in your answers in the space provided under each question. You can list your answers concisely in point-form.

Part (a) [1.5 MARK] List an advantage of each of the following:

- Using header file(s)
- Dividing your C code into multiple files
- Using existing libraries (e.g. `stdio.h`)

Part (b) [1.5 MARK] What value will be returned by the following function if argument `a` takes the value 3? Explain why.

```
float func(float a)
{
   int b=a/4;
   return b;
}
```

Part (c) [1.5 MARK] When and why may it be advantageous to specify the type of variable in a program as `unsigned` (e.g. `unsigned short int`)? In a `signed` variable, which bit is reserved for indicating whether the value stored in the variable is positive or negative?

Part (d) [1.5 MARK] Identify two issues in the following code snippet that may lead to undefined behaviour of the program.

```
int ** func( int a)
{
   int * b = (int *)malloc(sizeof(int));
   int * c = b;
   b=a+2;
   return &c;
}
```

Part (e) [1.5 MARK] Assume that a function is declared as follows: `int circle(int j);`. Write code for the following:

- Declare a function pointer variable and set it to the function `circle`
- Print the result of calling the function via the function pointer variable where the argument is set to `2` (output formatting will not matter)

Part (f) [1.5 MARK] Consider the following code snippet. Assume that `short int` requires 16 bits for storage. What value will be stored in `b` at the end of the second line (express answer in decimal)?

```
short int a = 13;
short int b = a | (1<<5);
```

Part (g) [1.5 MARK] Suppose structure `Sample` is defined as follows. Write the `if`-statement that compares whether the values stored in two structure variables `var1` and `var2` of type `Sample` are the same. If the values are the same, `1` is returned. Otherwise, `0` is returned.

```
struct Sample{
   int a;
   int b;
};
```

Part (h) [1.5 MARK] If an algorithm has a complexity of $O(n)$, what does this mean in terms of the formal definition of the big-oh notation?

Part (i) [1.5 MARK] How are recursive function calls processed in memory (similar to what ADT)? What is the insertion and deletion policy of this ADT?

Part (j) [1.5 MARK] Suppose that nodes with key values $5, 19, 2, 4, 20, 3$ are inserted in that order (from left to right) into a binary search tree. Illustrate the resulting tree.

Part (k) [1.5 MARK] For the tree constructed in the previous question:

- What is the height of the tree (assume that this is an extended tree)?
- What is the result of an in-order traversal of the tree?
- What is the balance factor of the root node?

Refer to Fig. 1 for the next 2 sub-questions. The root of each tree is the top-most node.

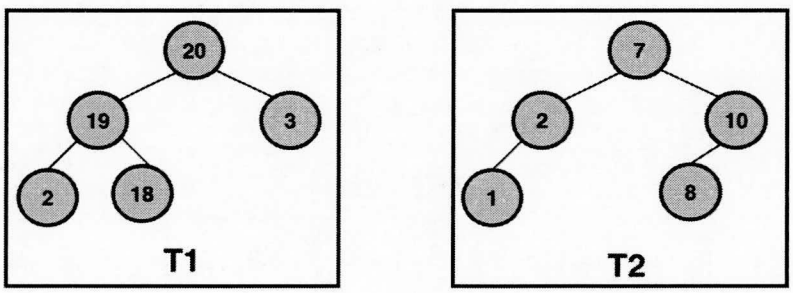


Figure 1: Trees

Part (l) [1.5 MARK] Provide an answer for each of the following based on T1:

- Is T1 a heap, BST and/or an AVL tree?
- Is this a complete and/full tree?
- What is the result of a pre-order traversal of the tree?

Part (m) [1.5 MARK] Provide an answer for each of the following based on T2:

- Is T2 a heap, BST and/or an AVL tree?
- What is the depth of the node containing the value 1?
- What is the result of a post-order traversal of the tree?

Refer to Fig. 2 for the next 6 sub-questions. The root of each graph is the node with the vertex label 0.

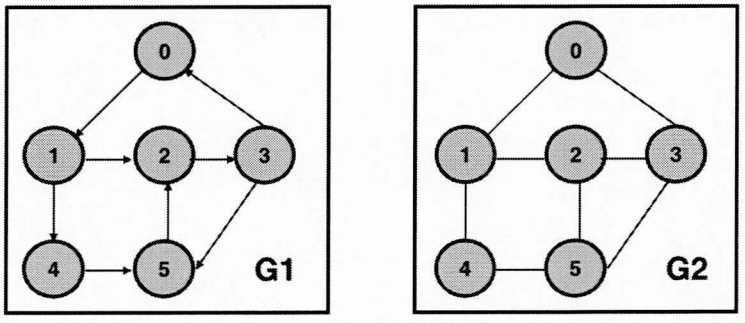


Figure 2: Graphs

Part (n) [1.5 MARK] What is the adjacency matrix representation of G1?

Part (o) [1.5 MARK] What is the adjacency list representation of G2?

Part (p) [1.5 MARK] What is the result of depth-first traversal of G1 starting from node 0? What is the largest in-degree of nodes in G1?

Part (q) [1.5 MARK] What is the result of breadth-first traversal of G2 starting from node 0? How many cycles are there in G2?

Part (r) [1.5 MARK] Is it possible to apply topological ordering to G1? If yes, what will be the result? If not, why not?

Part (s) [1.5 MARK] Assume that all edges have a weight of 1 in G1. Suppose that a minimal spanning tree is to be constructed based on G1. Is there a unique solution? Explain why or why not.

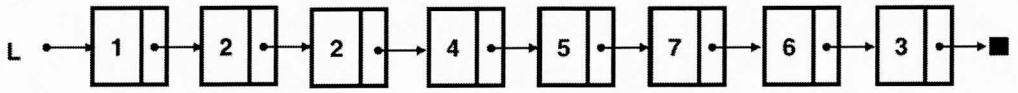
Part (t) [1.5 MARK] What is the advantage of using double hashing over linear probing for collision resolution in a hash table?

Question 2. [15 MARKS]

Given a pointer `L` to a linked list in which nodes of type `Node` (defined below) are dynamically allocated, implement the function:

- `float averageLinkedList(struct Node * L, int * n);`

which computes and returns the average of values stored in the `data` member of all nodes in the linked list using **recursion**. Do not implement additional helper functions. If you do not use recursion, we will still mark the question. However, a mark deduction of 5 will be applied (the highest mark you can get is 10/15). The second argument can be used as a helper variable and you can assume that it points to valid region in memory. In the first function call `*n` is initialized to 0. You can assume that the linked list has at least one node. In the example below, the result of function `averageLinkedList` should be 3.75.



Following is the definition of the node structure used in the linked lists:

```
struct Node{
   int data;
   struct Node * next;
};
```

```
float averageLinkedList(struct Node * L, int * n)
{

}
```

Question 3. [15 MARKS]

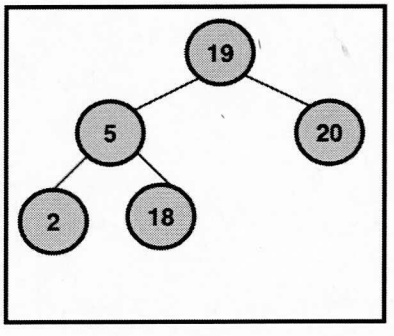
Implement the function `sumBST` that returns the sum of the `data` member of nodes in a binary search tree with values strictly greater than 10. Do not implement additional helper functions. You may or may not use recursion (no restriction on either implementation).

- `int sumBST(struct Node * root);`

The pointer to the root of the tree is passed as the argument to the first function call. Following is the definition of the node structure used in the binary search tree:

```
struct Node {
    int data;
    struct Node * lChild;
    struct Node * rChild;
};
```

Assume that all nodes in the tree are created via dynamic allocation. If there are no nodes with `data` values greater than 10, then return 0. In the following example, the result of the function call should be 57 (i.e. 18+19+20=57).



```
int sumBST(struct Node * root)
{

}
```

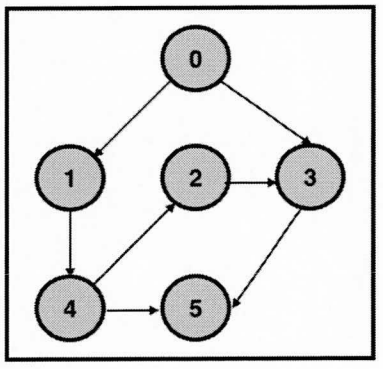
Question 4. [15 MARKS]

Implement the function `dfTraversal` which performs depth-first traversal of a directed graph represented by `aM` starting from the node labelled `n` and prints out the labels of nodes as these are being traversed. Do not implement additional helper functions. You may or may not use recursion (no restriction on either implementation). Note that the provided underlying graph representation is an *adjacency matrix*. You may assume that the following definitions and declarations are available to you:

```
#define NODES 6
struct adjMat
{
   int matrix[NODES][NODES];
   int vNodes[NODES];
};
struct Data{
   int value;
};
struct Stack * initStack();
void push(struct Stack * s, struct Data d);
struct Data pop(struct Stack * s);
int isEmpty(struct Stack * s);
void deleteStack(struct Stack * s);
```

`NODES` represents the total number of vertices in the graph. Labels assigned to vertices in the graph take values in the set [0, NODES-1]. You can assume that the `adjMat` structure is populated with appropriate values representing the graph before being passed as an argument to the function `dfTraversal`. The stack interface functions can be used as helper functions for the depth-first traversal implementation. You may assume that these interface definitions are available (i.e. you do not have to implement these).

As an example, consider the graph illustrated in the following figure. If the initial starting node `n` is 0, then the function `dfTraversal` should traverse the graph starting from node labelled `0` in depth-first order and print out the nodes as these are being traversed as follows: `0,1,4,2,3,5`.



```
void dfTraversal(struct adjMat * aM, int n)
{

}
```

Question 5. [15 MARKS]

Implement in-order traversal of a tree **without** using recursion. Each node in the tree has the following structure:

```
struct Node{
    int vLabel;
    struct Node * lChild;
    struct Node * rChild;
};
```

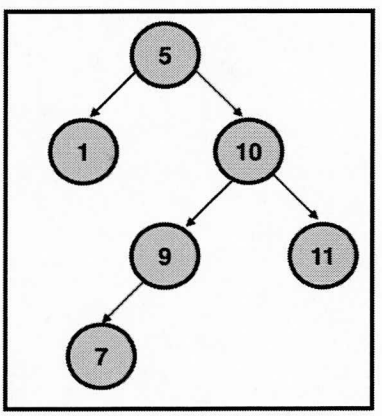
The function you will implement is:

- `void inOrderTraversal(struct Node * root);`

This function will print out the labels of nodes as these are being traversed in the tree, starting from the `root` node. You can use the `Stack` interface functions provided in the previous question for this. The only difference is the definition of the structure `Data` which is stored in each node of the stack:

```
struct Data{
    struct Node * n;
    int mark;
};
```

In the following example, in-order traversal will result in 1-5-7-9-10-11



```
void inOrderTraversal(struct Node * root)
{

}
```

[Use the space on this page for rough work. Indicate clearly any work you want us to mark.]

[Use the space on this page for rough work. Indicate clearly any work you want us to mark.]

[Use the space on this page for rough work. Indicate clearly any work you want us to mark.]

[Use the space on this page for rough work. Indicate clearly any work you want us to mark.]

Total Marks = 90

END OF FINAL EXAMINATION