# **Exam 2**

*This is a closed book and closed notes test.* You are not allowed to have anything on your desk other than pencil and this exam paper during the test; this includes *calculators* or *electronic assistance* of any kind – ***especially smartphones***.

*You may not leave to go to the restroom.* Please go before the exam starts.

*You may not ask questions.* If something is confusing, write a note beside the question and explain your assumptions.

*You must show all of your work on this exam.* You will not be allowed to turn in additional sheets of paper.

*Read and sign the following statement.*  Failure to sign the statement will result in a **zero** on the exam.

*I have neither given nor received unauthorized assistance on this test. I have notified the proctor of any violations of the above policies.*

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **Problem** | **Score** |
| 1 | / 25 |
| 2 | / 25 |
| 3 | / 25 |
| 4 | / 25 |
| **Total** | **/ 100** |

*Points divided evenly among parts of a problem unless otherwise specified.*

1. (25 points) Read all of the instructions below carefully. Consider the following class that implements a singly-linked list.

class List {

private:

// ListNode represents each

// node of the list

class ListNode {

public:

int item; // data in the list

ListNode \*next;

ListNode(Item a, ListNode \*n=NULL)

{

item = a;

next=n; // automatically serves as a list tail

}

};

// add head and tail pointer

ListNode \*head;

ListNode \*tail;

public:

List():head(NULL),tail(NULL) {}

// append a to tail of list

void append(int a);

// remove head, and removed item is returned by reference in r

bool removeHead(int &r);

bool empty() const;

**//Room for your own additional declarations:**

};

Implement the following member functions:

|  |  |  |
| --- | --- | --- |
| **Function** | **Function Declaration** | **Description** |
| (a) Copy constructor | List(const List &copy); | List one;  one.append(1);  one.append(2);  // one contains [1,2]  List two = one; // copy one into two  // two contains [1,2] |
| (b) Assignment operator | List& operator=(const List &rhs); | // assume variables one and two from  // prevoius row  List three;  three.append(5);  three = two; // three becomes equal to 2 |

You may assume that any declared functions in the class are implemented, and you may call them from your new functions if that would be helpful. However, anything not declared you will need to declare and implement them. Your implementation must be legal C++ code to earn full credit.

1. Copy Constructor implementation:

(b) Assignment operator implementation.

2. (25 points) Consider the following sequence of operations on a data structure for integers.

* Insert 923
* Insert 714
* Insert 22
* Insert 6735
* Insert 532
* Remove 6735
* Remove 22

1. [13 points] Show the final state of a hash table using linear probing after performing the operations above. Assume the hash table has length 10 and that the hash function is simply index = data % 10. Assume a probeDistance = 5.
2. [12 points] Show the final state of an AVL Tree for the same operations. Include your work for partial credit.

3. (25 points) Consider the following design requirements for a data structure. For each statement below, explain which data structure you would pick and briefly justify your answer.

1. An arbitrarily large or small number of integers need to be maintained in sorted order. The integers may be inserted in either sorted, partially sorted, or arbitrary order. Irregardless of the order inserted, the data structure maintains their order efficiently.
2. A data structure must hold 100 integers. It’s critical that the integers can be accessed by index, for example, get the 5th or 99th integer, efficiently.
3. A data structure must hold no more than 100 pointers. It must efficiently determine whether or not a given pointer has been inserted.
4. A data structure must hold pointers to the vertices in a graph. On occasion, you will need iterate over all of the vertices. No particular order of traversal is needed.
5. A data structure must hold the edges in a graph. It must support repeatedly and efficiently removing the edge with the smallest weight from the data structure.

4. (25 points) Describe the three cases for removing a node from a binary search tree, and draw an example of each one.

**C++ Keywords**

In common with C:

auto const double float int short struct unsigned  
break continue else for long signed switch void  
case default enum goto register sizeof typedef volatile  
char do extern if return static union while

Unique to C++:

asm dynamic\_cast namespace reinterpret\_cast try  
bool explicit new static\_cast typeid  
catch false operator template typename  
class friend private this using  
const\_cast inline public throw virtual  
delete mutable protected true wchar\_t

Reserved words:

and bitand compl not\_eq or\_eq xor\_eq  
and\_eq bitor not or xor

**ASCII Table**

