CSCI 104 Spring 2020 Midterm 1 Exam

DO NOT OPEN EXAM UNTIL INSTRUCTED TO DO SO

PLEASE TURN OFF ALL CELL PHONES

Name:

ID:

Date and time:

- You will have 110 minutes for the exam.
- This exam is individual effort.
- No electronic devices are permitted or necessary.
- You may use one handwritten 8.5" by 11" sheet of notes on both sides that you have prepared. These notes will be collected with your exam.

Question	Score	Points
1		10
2		10
3		10
4		10
5		12
6		18
Total Score		70

General Notes on Data Structures
For some problems, you are explicitly allowed and encouraged to draw on and use data structures from class. Below is the syntax we gave in class (more or less), which you are welcome to use. This page is not exhaustive. **You may also use C++ STL syntax and functions.**

- All data structures provide the following two constant-time functions:

```
bool empty () const; // returns whether the structure is empty
unsigned int size() const; // returns the number of elements
```

- Maps, referenced as $map < K, V >$. The functions you can assume are the following:

```
void insert (const K & key, const V & value); // does nothing if key is already in the map
void remove (const K & key); // does nothing if key is not in the map
iterator find (const K & key); //returns iterator to pair if key found and end otherwise
V & operator[] (const K & key); /* if key was not in the map, adds it to the map
                                   with a garbage value assigned to it. */
```

  All operations take $\Theta(\log n)$ time.

- Vectors, referenced as $vector < T >$. The functions you may assume and their running times are as follows:

```
T & operator[] (int loc); \\ Takes O(1) worst-case time.
void push_back (const T & item); \\ Takes O(1) amortized runtime.
```

- Stacks, referenced as $stack < T >$. The functions you may assume are the following, all running in time $O(1)$.

```
void push (const T & item);
void pop ();
const T & top () const;
```

- Queues, referenced as $queue < T >$. The functions you may assume are the following, all running in time $O(1)$.

```
void enqueue (const T & item);
void dequeue ();
const T & front () const;
```

- Of course, you are also allowed to use arrays (static or dynamically allocated, as you see fit), and define your own structs or classes if it helps you.

If you need these data structures or algorithms — either as member variables or in any form of inheritance — you may just use them without worrying about **#include** or how they are implemented themselves, apart from the information we gave you above.

1. **Multiple Choice Questions**

```
// class SampleClass defined in SampleClass.h
SampleClass A, B; /*1*/
B = A; /*2*/
SampleClass C = B; /*3*/
```

(a) In the line above labeled /*2*/ what function is called? Select one.

i. Copy constructor
ii. Assignment operator
iii. Default constructor

(b) In the line above labeled /*3*/ what function is called? Select one.

i. Copy constructor
ii. Assignment operator
iii. Default constructor

(c) Now consider new classes: ClassA, ClassB, and ClassC described in the following problems. The following is the prototype for the copy constructor for ClassA:

```
ClassA(const ClassA& copy);
```

Select all statements that are true about this function prototype.

i. This function prohibits the argument object from being modified.
ii. The objected referenced by 'this' cannot be modified.
iii. The argument object is not copied by this function.
iv. The prototype permits assignments of ClassA objects to be chained.

(d) ClassB implements a default constructor that dynamically allocates an array of integers and a copy constructor. Select all additional statements that are true about ClassB.

i. ClassB should implement a destructor to prevent memory leaks.
ii. If no assignment operator is defined, double frees are possible.
iii. If an assignment operator is defined, it may need to delete the array.

(e) ClassB inherits from its base class ClassA using protected inheritance. ClassC inherits from ClassB using public inheritance. Select all statements that are true about these classes.

i. ClassC may access all of the protected and public functions and data of ClassA.
ii. ClassB may access all of the protected and public functions and data of ClassA.
iii. Inside main() public functions of both ClassC and ClassB may be called using an instance of a ClassC object.

(f) ClassA is an abstract base class. ClassB is derived from ClassA publicly and implements all pure virtual functions. ClassC inherits from ClassB publicly. Select all statements that are true.

i. Objects of ClassA, ClassB, and ClassC may be instantiated.
ii. Objects of ClassB and ClassC may be instantiated.
iii. Pointers to ClassB and ClassC may be assigned to pointers to ClassA.
iv. Pointers to ClassB may be assigned to pointers to ClassC.

2. In this exercise you will trace a mystery recursive function on singly linked lists call **m**.

```
#include <iostream>
using namespace std;

typedef struct Item {
  int val;
  Item* next;
  Item(int v, Item* n) : val(v), next(n) {}
} Item;

Item* m(Item* a, Item * b)
{
  if(a == NULL){
    return b;
  }
  else if(b == NULL){
    return a;
  }
  else {
    a->next = m(b, a->next);
    return a;
  }
}
int main()
{
    Item * first = new Item(19,new Item(28, NULL));
    Item * second = new Item(3,new Item(5,new Item(7, NULL)));
    Item *third = m(first, second);
    Item *tmp = third;
    while (tmp!= NULL){
        cout << tmp->val << " ";
        tmp = tmp->next;
    }
    cout << endl;
    // assume code here to prevent memory leaks
    return 0;
}
```

Print what will be output by main here:

3. Write a function, **isParenthesized**, that given an expression string will return true if and only if the string is properly parenthesized and false otherwise. Recall from lecture that an expression is properly parenthesized if a left parentheses such as '(' and '[' have a matching right parentheses. Symbols such as '+', '*', and numbers may be ignored. You may assume expression strings will only contain '(', ')", '[', ']', '+', '*', or numbers. **Your implementation must be iterative.** *You may use STL container classes.*

```
 bool isParenthesized(string exp);
/** isParenthesized("(7+8)") and isParenthesized("[[7+3]*(6+8)]") should return true **/
/** isParenthesized("[7+8") and isParenthesized("[7+8)") should return false **/
/** isParenthesized("(7+[8)-5]") should return false **/
/* You may begin your implementation here*/
 bool isParenthesized(string exp) {
```

4. In this exercise, we will implement removal for a **doubly** linked list. The function **remove** will be given a head to the doubly linked list by reference and an integer target value. All elements of the doubly linked list with the integer target value should be removed from the list and the head should be updated if necessary. *Assume that all items on the list had been dynamically allocated.* Your solution must be **iterative**.

```
struct DLItem{
  int val;
  DLItem *next;
  DLItem *prev;
  DLItem(DLItem *n, DLItem *p, int v):next(n), prev(p), val(v){}
};

void remove(DLItem*& head, int target) {
 /* You may begin your implementation here */
```

5. Trace through main() and write what would be printed as a result of running main() in the box below.

```
#include <iostream>
#include <string>
using namespace std;
class ClassA {
  public:
    ClassA()  { cout << endl << "A()"; }
    ClassA(string name): list_name(name)  {cout << endl << "A() " << list_name << " ";}
    ~ClassA()  { cout << "~A() " << list_name << " " << endl; }
    virtual  void print()  { cout << "A: " << list_name << endl; }
  private:
    string list_name;
};
class ClassB: public ClassA {
  public:
    ClassB(): ClassA("default list"), count(0)  {cout << "B() "; }
    ClassB(string list_name, int c) : ClassA(list_name), count(c)  {cout << "B() ";}
    virtual ~ClassB() { cout << "~B() ";}
     void print() { cout << endl << "B: "; ClassA::print();}
  protected:
    int count;
};
class ClassC: private ClassB {
  public:
    ClassC(): ClassB()  {cout << "C() "; }
    ClassC(int k, string name): ClassB(name, k)  {cout << "C()";}
    ~ClassC() {cout << "~C()" ; }
    void print() { cout << endl << "C: "; ClassA::print(); }
};
int main() {
   ClassA *t2 = new ClassB;
   ClassB *t3 = new ClassB ("orange",5);
   ClassC *t4 = new ClassC(10, "grape");
   ClassC t5(*t4);
   t2->print();
   delete t4;
   delete t3;
   delete t2;
   return 0;
}
```

Print what will be output by main below: (Use - for blank line)

6. In this problem you will help National Geographic magazine collectors maintain their collection. They would like to keep track of their magazines so that the most recent issue by publication date that they have is most easily accessible to them at the top of their collection. Every issue has a unique string identifier. They would like to keep track of the total value of their collection and also be able to sell issues as necessary. In order to sell issues, they need to be able to get the cost of an issue quickly, but it is fine if searching for the actual issue is not very efficient (i.e. linear). You are to complete the implementation of the class **MagazineCollection**, with the specified runtime requirements. Hint: You may also use STL classes in your implementations if necessary. We have provided a simplified STL syntax on the first page of your exam that you may also use.

```
class MagazineCollection {
   /* Let n be the number of magazines in the collection*/
   public:
   MagazineCollection();
   /* constructor initializes an empty collection with zero value. Worst case runtime: O(1)*/

   ~MagazineCollection (); // destructor

   void addMagazine(string issue, int value);
   /* adds the magazine issue with given value. Assume that this function will be called by the
      collectors with magazine issues in the order that the issues are released by publication
      date. Assume value is greater than 0. If the magazine is already in the collection, this
      function should do nothing. Worst case runtime: O(n)  */

   int getMagazineValue(string issue) const;
   /* returns the value of any magazine in the collection. Returns 0 if the issue is not in the
       collection. Worst case runtime: O(log n) */

   int getValueOfRecentM(int m) const;
   /* returns value of the m most recent magazine issues in the collection. If n < m, return
       value of entire collection. Worst case runtime: O(n log n) */

   void sellMagazine(string issue);
   /* this function should remove the magazine with the given issue identifier from the
      collection of magazines and its value from the collection value. If the magazine issue is
      not in the collection, this function should do nothing. Worst case runtime: O(n)*/

   int getCollectionValue() const;
    /* Return the value of the entire magazine collection. Worst case runtime: O(1) */

   private:
   // feel free to add private member variables or functions




};
```

```
MagazineCollection::MagazineCollection()   /* Worst case runtime: O(1) */
{



}
MagazineCollection::~MagazineCollection () {



}
void MagazineCollection::addMagazine(string issue, int value){ /*Worst case runtime: O(n)*/
```

```
}
int MagazineCollection::getMagazineValue(string issue) const  /*Worst case runtime: O(log n)*/
{
```

```
}
int MagazineCollectioon::getCollectionValue() const {  /*Worst case runtime: O(1)*/



}
```

```
int MagazineCollection::getValueOfRecentM(int m) const /*Worst case runtime: O(n*log(n))*/
{









}
void MagazineCollection::sellMagazine(string issue) /*Worst case runtime: O(n)*/
{









}
```

This page is left intentionally blank for **UNGRADED scratch work**. No name is required but please turn it in with your exam.