Spring 2013: CS 3323 Midterm Exam 1

Total Time: 40 minutes Total Points: 45

Write your name clearly. Answer all the questions.

Reead all the questions. If you need more space, use the other side of your question sheet.

Name:_____ **Date:**_____

****

Question 1. True/False **[10]**

1. Garbage collection is a form of memory management. True
2. Deleted dynamic elements make garbage collection a tougher task. False
3. List can grow indefinitely. False
4. Array based implementation of list may require shifting of elements when elements are added/deleted. True
5. List is a LIFO based data structure. False
6. By default array data structure has inbuilt out of bound error checking. False
7. Node element in linked list implementation has members for storing value and storing addresses. Trye
8. Stack is not a good choice for conversion of infix operation into postfix operation. False
9. Deep copy constructor should not be written for dynamic elements. False
10. Elements are added and deleted at the end (top) of the stack. True

Question 3. Analyzing code segment. **[12]**

Observe the following declarations:

```
class Node{ public:
     int data;
     Node * next
}; Node *p1 = new Node, *p2 = new Node, *p3 = new Node;
```

Tell what will be displayed:

`p1->data = 12; p2->data = 34;`
 	
`p1->next = p2;  p2->next = p1;`
 	
`cout << p2->data<< " " << p2->next->data<<endl;`
 	
`cout << P1->next->data<< " " << P1->next->next->data;`
	34 12
	
	34 12

P1->data = 12; p2->data = 34; *P1 = *P2; cout << p2->data<< " " << p2->next->data<<endl; cout << P1->next->data<< " " << P1->next->next->data;	 34 34 34 34
P1->data = 12; p2->data = 34; p3->data = 34; P1->next = p2; P2->next = p3; P3->next = 0; cout << P1->data<< " " << p1->next->data<< " " << p1->next->next->data << " "<<p3->data;	 12 34 34 34

For next set of problems, assume that `Stack` is the class implemented by using static arrays and can hold integer values (the one we discussed in the class). The capacity of stack is set to 5. **Give the value of `myTop` and the contents of the array referred to by `myArray` in the stack `s` after the code segment is executed, or indicate why an error occurs.**

Stack s;
 	
s.push(123);
 	
s.push(456);
 	
s.pop();
 	
s.push(789);
 	
s.pop(); s.pop();
	
	myTop = -1

Stack s;
 	
s.push(222);
 	
int i = s.top();
 	
s.pop();
 	
s.push(i);
 	
s.pop();
	
	myTop = -1
	
	myArray[-1] return an error

Stack s;
 	
for(int i = 1; i < 5; i++) s.push(i*i);
 	
s.pop();
	
	myTop = 4
	myArray[0] =1
	myArray[1] =2
	myArray[2] = 3
	myArray[3] = 4

Question 4. Explain the following concepts: **[6]**

a. **Stack-ADT:** Considered a linear data structure, can only add or delete add the top element. Last in first out operation

b. **List-ADT:** Is an abstract data type that describes a linear collection of data items in some order, in that each element occupies a specific position in the list

c. **Role of Activation Record in recursive function calls:**

Question 5. Convert the following infix expressions into postfix expressions **[5]**

a. `(a + b)*(c * (d - e)/f)`

ab+cde-*f/*

b. `(7 * 8 -(2+3)) % 2 + 2`

78*23+-2%2+

Question 4. Observe following declaration and description for a list: **[3+3]**

```
class list {
class Node{
public:
     datatype value;
     Node *next;
     Node():next(0){}
     Node(datatype val):value(val),next(0){}
     } ;
Node *first;
int mySize;

public:
~List();
/*--------------------------------------------------------------------
Destructor
Precondition: This list's lifetime is over.
Postcondition: This list has been destroyed.
--------------------------------------------------------------------*/
```

```
int NodeCount();
/*-------------------------------------------------------------------------
NodeCount: Counts the number of nodes in List object
Precondition: None.
Postcondition: None.
-------------------------------------------------------------------------*/   };
```

Following concepts described in the class, provide a destructor and and a Nodecount function that returns numbers of nodes in List for this list class.

```
~List()
{
   while (first != 0)
    {
      Node* temp = first;
      first = first -> next;
      delete temp;
    }
}
```

```
int NodeCount()
{
   int count =0;
   Node* temp = first;
   while (temp != 0)
   {
    count++;
    temp = temp->next;
    }
   return Count;
}
```

Question 4. Observe following declaration for a Stack: **[3+3]**

```
class Stack {
 public:
  /***** Function Members *****/
  /***** Constructors *****/
  Stack();

  Stack(const Stack & original);
  /*-----------------------------------------------------------
    Copy Constructor
  -----------------------------------------------------------*/
 /***** Destructor *****/
  ~Stack();

  void pop();
  /*-----------------------------------------------------------
    Remove value at top of stack (if any).

    Precondition:  Stack is nonempty.
    Postcondition: Value at top of stack has been removed,
    -----------------------------------------------------------*/
  void push(int value);
  /*-----------------------------------------------------------
    Add the value at top of stack
    Precondition:  None
    Postcondition: Value has been added at top of stack.
    -----------------------------------------------------------*/

  private:
    /*** Node class ***/
   class Node
   {
    public:
       int data;
      Node * next;
      //--- Node constructor
      Node(StackElement value, Node * link = 0)
      { data = value; next = link; }

  };

  /***** Data Members *****/
  NodePointer myTop;      // pointer to top of stack
};
```

Following concepts described in the class, provide push and pop methods

```
void pop()
{
    node* temp= first;
    while (temp->next -> next != 0)
    {
       temp = temp -> next;
    }
    node* temp2 = temp -> next ->next;
    temp -> next = 0;
    delete temp2;
}
```

```
void push(int value)
{
   node* temp = first;
   while (temp -> next != 0)
   {
      temp = temp -> next;
   }
    temp -> next = new Node (value);
}
```

[Bonus Question] Observe following declaration for a polynomial class [6]

```
class Poly {
  int Degree;
  int Cofact[capacity]
 public:
  /***** Function Members *****/

  /***** Constructor *****/

  };
```

With the above definitions, provide an overloaded operator for addition operation, in other words overload '+' operator for this class.

```
Poly operator+ (const Poly& p)
{
   if (Degree => p.Degree)
   {
      for (int i=0 ; i< p.Degree; i++)
       {
          Cofact[i] += p.Cofact[i];
       }
   }
   else
   {
        for (int i=0 ; i< p.Degree; i++)
       {
          p.Cofact[i] += Cofact[i];
       }
   delete[] Cofact;
   Cofact = new int [p.Degree + 1];

   for (int i=0 ; i < p.Degree; i++)
   {
     Cofact[i] = p.Cofact[i];
   }
```