Question 1 (C++ inheritance)

Consider the following program in C++:

```
#include <iostream>
class A {
public:
  A(){ cout<<"A:A() is called"<<endl; }
  void foo1(){ cout<<"A:foo1 is called"<<endl; };
  void foo2(){ cout<<"A:foo2 is called"<<endl; };
  void foo3(){ cout<<"A:foo3 is called"<<endl; };
};

class B : public A {
public:
  B(){ cout<<"B:B() is called"<<endl; };
  void foo1() {
    cout<< "B:foo1 is called" << endl;
    A::foo1();
  };
  void foo2() {
    foo1();
    cout <<"B:foo2 is called"<<endl;
  };
};
class C : public B {    // wow!
public:
  C(){ cout<< "C:C() is called" <<endl; };
  void foo3() { cout << "C:foo3 is called" << endl;};
};
int main()
{
  B b;
  C c;

  cout << ">>> b starts to call its member functions." << endl;
  b.foo1();
  b.foo2();
  b.foo3();
  cout << ">>> Now c calls its member functions." << endl;
  c.foo1();
  c.foo2();
  c.foo3();
}
```

Please write down the output of the program.

Answer (4 marks):

```
A:A() is called
B:B() is called
A:A() is called
B:B() is called
C:C() is called

>>> b starts to call its member functions.
B:foo1 is called
A:foo1 is called
B:foo1 is called
A:foo1 is called
B:foo2 is called
A:foo3 is called

>>> Now c calls its member functions.
B:foo1 is called
A:foo1 is called
B:foo1 is called
A:foo1 is called
B:foo2 is called
C:foo3 is called
```

Question 2 (DoubleLinkList)

Write a member function `deleteNodeAt(int i)` for the class `DoubleLinkList` to delete a node. The function will delete a node in the double link list at position *i*. You can assume that all other functions are implemented except `deleteNodeAt(int)`. Here is the `DoubleLinkList` declarations:

```
template <class T>
class DoubleLinkListNode {
  protected:
    T content_;
    DoubleLinkListNode<T>* next_;
    DoubleLinkListNode<T>* prev_;
    friend class DoubleLinkList<T>;
};

template <class T>
class DoubleLinkList {
  public:
     DoubleLinkList();        // initialize head_ and size_
     void deleteHead();       // delete the node at the head
     void deleteTail();       // delete the node at the tail
     void insertAtHead(T&);   // insert a node at the head
     void insertAtTail(T&);   // insert a node at the tail
     int  size();             // return the number of elements
     void insertNodeAt(int);  // insert a node at a certain
                              // position
     void deleteNodeAt(int);  // delete a node at a certain
                              // position
  protected:
    int size_;
     DoubleLinkListNode<T>* head_;
     DoubleLinkListNode<T>* searchForNode(int);
};
```

You are required to add in the safe-guards also. However, you are not required to use exceptions but you are required to print out error messages. through `cout` or `cerr`.

Answer (3 marks):

```
template <class T>
void DoubleLinkList<T>::deleteNodeAt(int i)
{
  if(i>size() || i<1)
  {
    cout << “ERROR: Delete at invalid position” << endl;
    return;
  } else if(i==1)
    deleteHead();
  else if(i==size())
    deleteTail();
  else {
    DoubleLinkListNode<T> *ptr = searchForNode(i);
    ptr->next_->prev_ = ptr->prev_;
    ptr->prev_->next_ = ptr->next_;
    delete ptr;
  }
}
```

Question 3 (OrderLinkList and Time Complexity)

a) Let there be *n* elements in the list. What is the worst time complexity for the function `searchForNode()` in `OrderLinkList` in the lecture notes? Explain why. (1 mark)

b) Given `OrderLinkList` as in the lecture notes, Mr. Silly implemented the member function `printInReverseOrder()`. The function prints the list from the last node to the head.

```
void OrderLinkList<T>::printInReverseOrder()
{
        int i
        BasicLinkListNode<T>* nodePtr;
        for(i=size();i>0;i--)
        {
                nodePtr = searchForNode(i);
                cout << nodePtr->content_ << " ";
        }
        cout << endl;
}
```

What is the worst time complexity for this function? Explain why. (1 mark)

c) Suggest **another** type of implementation to help Mr. Silly to improve the time complexity of `printInReverseOrder()` and state the new time complexity. You are required to describe the method, but not necessary to give the detail implementation. (1 mark)

Answers (1+1+1=3 marks):

a) O(n) *because the function will traverse all n nodes in the worst case*
b) O(n^2) *because the loop will loop n times and each searchForNode is O(n)*
c) Use recursion, *recurs into another level of function before print out the current node.*

Question 4 (ADT:Implementing Queues using TailLinkList)

Assume that you are given the `tailLinkList` data structure and its implementation:

```
template <class T>
class TailLinkList : public OrderLinkList<T>
{
  public:
    TailLinkList();
    void deleteHead();
    void insertAtHead(T);
    void insertAtTail(T);
    void deleteTail();
    int  size();
    T    contentOfHead() {return head_->content_;}; // assume that we can
    T    contentOfTail() {return tail_->content_;}; // access 'content_' also
};
```

Implemented as in the lecture notes

Implement the **7** function bodies of the data structure Queue below, following the template declaration **USING** `TailLinkList`:

```
template <class T>
class Queue {
  protected:
    TailLinkList<T> tll_;
  public:
     Queue();
     void enqueue(T&);   // put an element into the queue
     void dequeue();     // remove an element from the queue
     T    front();       // return the element in the queue front
     T    back();        // return the element in the back
     bool isEmpty();     // return true if the queue is empty
     int  size();        // return the number of elements
};
```

You are required to give all the safe-guards. Namely, you should check for errors and print error messages in your implementation through `cout` or `cerr`. However, you are not required to use Exceptions. (Please indicate where your answers are if you do not have enough spaces in this page and write in other pages.)

Answers (5 marks):

```
template <class T>
Queue<T>::Queue() {};

template <class T>
void Queue::enqueue(T& item)
{ tll_.addAtTail(item);};

template <class T>
void Queue<T>::dequeue()
{
  if(tll.size()==0)
     cout << "ERROR: The list is empty" << endl;
  else
     tll_.deleteHead();
};

template <class T>
T Queue<T>::front()
{
  T item;
  if(tll_.size()==0)
    cout << "ERROR: The list is empty" << endl;
  else
    item = tll_.contentOfHead();
  return item;
}
```

```
template <class T>
T Queue<T>::back()
{
  T item;
  if(tll_.size()==0)
    cout << "ERROR: The list is empty" << endl;
  else
    item = tll_.contentOfTail();
  return item;
}

template <class T>
bool Queue<T>::isEmpty()
{ return tll_.size() == 0; };

template <class T>
int Queue<T>::size()
{ return tll_.size(); };
```

Question 5 (Stacks)

a) By using the Infix to Postfix code given in the lecture, please show step by step how to convert the expression:

(5 - (7 – 6 / 2) + 3 * 4)

from the Infix notation into the Postfix notation by using a stack.

Answers (part a: 3 marks):

ch	stack	postfix expression
(	(	
5	(	5
-	(-	5
(	(- (	5
7	(- (	57
-	(- (-	57
6	(- (-	576
/	(- (- /	576
2	(- (- /	5762
)	(-	5762/-
+	(+	5762/--
3	(+	5762/--3
*	(+ *	5762/--3
4	(+ *	5762/--34
)		5762/--34*+

Question 6 (Stacks) (cont.)

b) Evaluate the postfix expression in 6a) using a stack and show the steps.

Answers (part b: 2 marks):

Copy your postfix expression here first : _5762/--34*+_____

Stacks: Bottom >>>> >> top	characters:	Stacks: Bottom >>>> >> top	characters:
5	5	134	4
57	7	1 12	*
576	6	13	+
5762	2		
573	/		
54	-		
1	-		
13	3		

- END OF PAPER -