University of Toronto
Faculty of Applied Science and Engineering

ECE 244F

PROGRAMMING FUNDAMENTALS

Fall 2013

Final Examination

Examiners: T.S. Abdelrahman, V. Betz, M. Stumm and H. Timorabadi

Duration: Two and a Half Hours

This exam is OPEN Textbook and CLOSED notes. The use of computing and/or communicating devices is NOT permitted.

Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted.

The value of each part of each question is indicated. The total value of all questions is 100.

Name: ________________________________
(Underline last name)

Student Number: ________________________________

Q1:		**Q8:**			
Q2:		**Q9:**			
Q3:		**Q10:**			
Q4:		**Q11:**			
Q5:		**Q12:**			
Q6:		**Q13:**			
Q7:		**Q14:**		**Total:**	

Question 1. (12 marks). *General.*

Answer the following questions either by **Yes** or **No**, or by providing a **very brief** and **direct** answer when indicated.

(a) **Yes** or **No**? It is safe for a function to return a pointer to an object that was created on the stack inside the function.

(b) **Yes** or **No**. An `O(log(n))` algorithm may or may not be faster than an `O(n)` algorithm when `n>1`?

(c) **Yes** or **No**. The parameter to a copy constructor must be passed by reference.

(d) **Yes** or **No**. In analyzing the complexity of an algorithm, the "best-case" scenario refers to the situation where the size of the input is relatively small. In contrast, the worst-case scenario refers to the situation where the size of the input is relatively large.

(e) **Yes** or **No**? A derived class constructor must always call the default base class constructor to initialize and possibly allocate memory for the base object.

(f) **Yes** or **No**? If you destroy an object through a pointer to a base class, and the base-class destructor is not virtual, the derived-class destructors are not executed, and the destruction might not be complete.

(g) What is the complexity (in Big-O notation) of the most efficient algorithm that counts the number of nodes in a binary search tree?

(h) **Yes** or **No**? Hash tables can only be used with objects that have integer keys.

(i) **Yes** or **No**? Within a derived class member function, code can directly access the private data members of the base class.

(j) **Yes** or **No**? A function that changes the private variables of a class must be written as a member function.

(k) What is the average computational complexity of inserting an object into a hash table, in big-O notation?

(l) In general, it is not possible to assign a base object (of type `BaseC`) to a derived object (type `DerivedC`). However, it is possible to implement a member function of the derived class that allows this assignment to happen. Write the prototype of this member function.

Question 2. (11 marks). *General.*

(a) **(1 mark)**. Which of the following statements about binary trees is NOT true?

A. Every binary tree has at least one node.
B. Every non-empty tree has exactly one root node.
C. Every node has at most two children.
D. Every non-root node has exactly one parent.

(b) **(1 mark)**. Given a binary search tree, which traversal type would print the values in the nodes in sorted order?

A. Preorder
B. Postorder
C. Inorder
D. None of the above

(c) **(1 mark)**. When should a pointer parameter `p` be a reference parameter? (That is, when would it be more appropriate for a parameter list to be `(myType * & p)` rather than `(myType * p)`?)

A. When the function changes `p`, and you want the change to affect the actual pointer argument.
B. When the function changes `p`, and you do NOT want the change to affect the actual pointer argument.
C. When the function changes `*p`, and you want the change to affect the object that is pointed at.
D. When the function changes `*p`, and you do NOT want the change to affect the object that is pointed at.
E. When the pointer points to a large object.

(d) **(4 marks)**. Consider the following class definition. The numbers listed on the left are not part of the code; they are there for reference.

```
 1: class Strange {
 2:    private:
 3:       char* name;
 4:       int levels;
 5:       int* scores;
 6:   public:
 7:       Strange ();
 8:       Strange (const char * n);
 9:       Strange (const char * n, int g);
10:       Strange (const Strange & g);
11:       ~Strange ();
12: };
```

Which class method is invoked by each of the following statements? Write your answers in the table on the next page.

Statement	**Class Method (write line number)**
`Strange First;`	
`Strange Second (“firstname”);`	
`Strange Third (“hello”, 8);`	
`Strange *p = new Strange ();`	

(e) (4 marks). Consider the following definitions of a base and two derived classes. There is also a code segment that uses the classes. The code segment assumes no objects of type `baseWindow` are allowed to exist. Also, each of the other classes must have a method called `draw()` that is invoked to correctly draw the window. The definitions not complete. Complete the definitions, showing your answer directly on the code below.

```
class baseWindow {

 private:
  WinID* win_id;
  // WinID is a class

 public:
  baseWindow ();
  ~baseWindow ();



};
```

```
class colorWindow :
    public baseWindow {

 private:
  Color* win_color;
  //Color is a class

 public:
  colorWindow ();
  ~colorWindow ();


};
```

```
class bwWindow :
    public baseWindow {

 private:
  Shade* win_shade;
  //Shade is a class

 public:
  bwWindow ();
  ~bwWindow ();


};
```

```
// Draw all windows. No objects of type baseWindow can exist
for (baseWindow* current_window = first_window();
             current_window !=NULL;
             current_window = next_window() ) current_window->draw();
```

Question 3. (4 marks). *Classes and Objects.*

In some C++ programs, it is desired that objects be deleted only using an *objects manager*. Consider the following definition of a C++ class called `instruction` (defined in `instruction.h`) and an object manager class called `ObjectManager`.

```
#include “operand.h”                                        instruction.h
#include “operation.h”
class instruction {
   private:
      operand* left_op;
      operand* right_op;
      operation opn;

   public:
      instruction(operand* l, operand* r, operation opn);
      instruction(const instruction & other);
      ~instruction();
      instruction & operator=(const instruction & rhs);
      void print () const;

};
```

```
#include “instruction.h”                                    ObjectManager.h
class ObjectManager {
   private:
      :
      :

   public:
      ObjectManager();
      ~ObjectManager();
      :
      void delete_obj ();

};
```

It is desired that the only non-member function that can delete objects of type `instruction` be the `delete_obj` method of an `ObjectManager` object. Thus,

```
{
     instruction a;
}
```

in the `main` function should generate a compile time error since the object `a` is deleted when it goes out of scope. The same holds for code that looks like this, also in `main`,

```
instruction * p;
p = new instruction (.........);
:
delete p;
```

since "`delete p`" will delete the dynamic object. What are the code changes needed to the class definition in `instruction.h` to make sure that only the `delete_obj` method of an `ObjectManager` object is able to delete objects of type `instruction` but no other method that is non-member of `instruction` can? Write your answer in the code boxes above.

Question 4. (8 marks). *Objects with Pointers.*

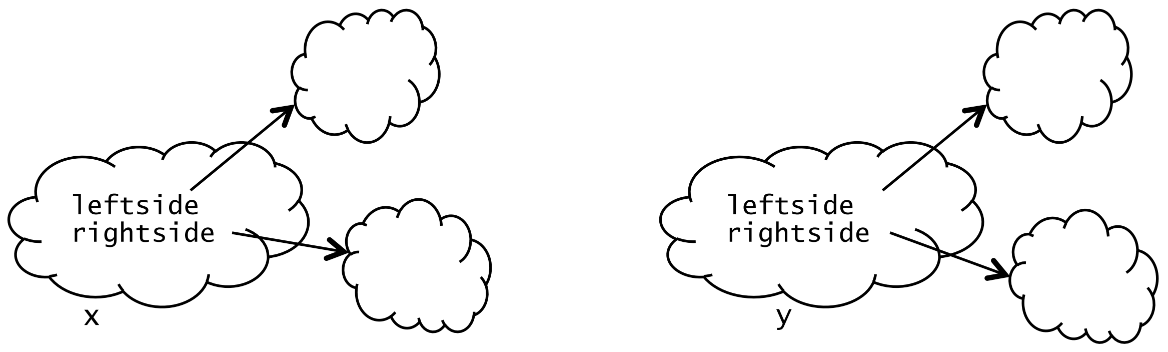
Consider the following definition and implementation of class `Mystery`. The definition and implementation of the classes `Left` and `Right` are not available and you may assume they are correctly defined and implemented.

```
#include “Left.h”
#include “Right.h”

class Mystery {
   private:
       Left*   leftside;
       Right*  rightside;

   public:
       Mystery ();
       ~Mystery ();
       Left* getLeft() const;
       Right* getRight() const;
};


Mystery::Mystery() {
   leftside = new Left();
   rightside = new Right();
}

Mystery::~Mystery() {
   delete leftside;
   delete rightside;
}

Left* Mystery::getLeft() const {return leftside;}

Right* Mystery::getRight() const {return rightside;}
```

Now consider the following segment of code in the `main` function.

```
#include “Mystery.h”

int main() {
    Mystery x;
    Mystery y;

    // point A
    x = y;
    // point B

    return (0);
}
```

(a) (2 marks). The figure below shows the two objects `x` and `y` at point A in the code. Adjust the diagram to show the two objects at point B in the code.



(b) (6 marks). There are one or more problems with the code. What member method(s) would you add to the class `Mystery` to address these problem(s)? Write below the implementation of each method you add.

Question 5. (8 marks). *Classes and Objects.*

Consider the two classes, `Album` and `Track`, as well as the function `albumLength()` below:

```
class Album{

 private:
    string title ;
    string artist ;
    int numTracks
    Track* playList ;

public:
    Album( string t, string a, int numT ) {
        title = t ;
        artist = a ;
        numTracks = numT ;
        playList = new Track[numT] ;
    }
    bool addTrack( Track t ) ;
    Track getTrack( int position ) const ;
    ~Album() {
        delete [] playList ;
    }
} ;
```

```
class Track {
 private:
    string title ;
    int length ;
 public:
    Track( string t, int l ) ;
    string getTitle() const ;
    int getLength() const ;
} ;

int albumLength( Album a ) {
 .
 . // code that calculates and
 . // returns the sum of the
 . // length of each album a's
 . // tracks. Assume that this
 . // code is correct.
 .
}
int main() {
    Album x;
        :
    return albumLength(x);
}
```

Describe the major error in the code above in simple terms. **Hint**: Draw a picture.

There are two different ways in which you can fix the error with one of them being more comprehensive and would prevent errors in other similar situations. The other fixes just this specific error. In the boxes below, provide the two fixes:

More comprehensive:

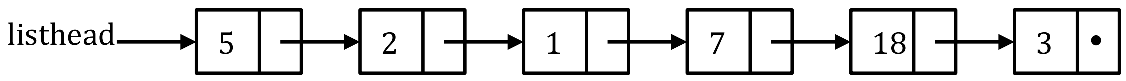
Specific:

Question 6. (6 marks). *Linked Lists.*

Consider the following class definition of nodes of a linked list. It appears in the file "`listNode.h`".

```
class listNode {
    private:
        int key;
        listNode* next;
    public:
        // Constructors & Destructor
        :
        // Accessors
        int getKey();               // Returns key of node
        listNode* getNext();        // Returns pointer to next node

        // Mutators
        :
};
```

Write a **recursive** function `void rPrint(Node* head)` that prints the keys of the linked list in reverse order to that in which the keys appear on the list. Thus, in the example below, the invocation of the function as `rPrint(listhead)` prints: 3 18 7 1 2 5.

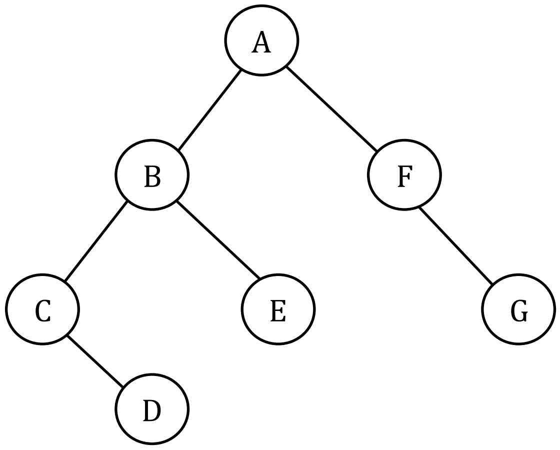


Write your answer in the box below. Keep you code as simple as possible, no more than 3-5 lines of code. Longer code will be penalized.

```
#include <iostream>
using namespace std;
#include "listNode.h"

void rPrint(Node* head) {













}
```

Question 7. (3 marks). *Trees.*

The following is a binary tree. Give the pre-order, in-order and post-order traversals of the tree.



Write your answers in the boxes below.

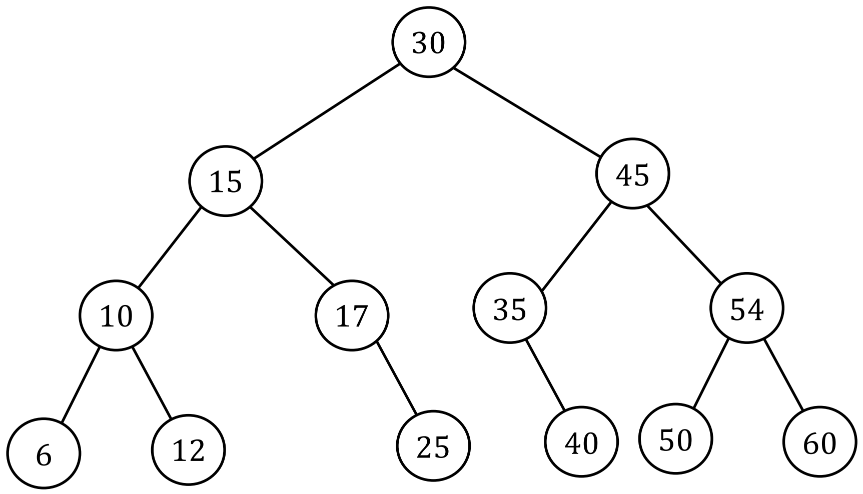
Pre-order traversal:

In-order traversal:

Post-order traversal:

Question 8. (3 marks). *Binary Search Trees.*

The following is a binary search tree.



The node with the key "15" is deleted from the tree. It is then re-inserted as a new node, also with the key "15". Draw changes to the tree above that result from the deletion followed by the insertion.

Question 9. (11 marks). *Inheritance.*

Consider the following definitions of base and derived classes.

```
class circuitElement {
   private:
       int ID;
   public:
       circuitElement();
       ~circuitElement();
       int getID();
       virtual void print() const = 0;
};

class resistor : public circuitElement {
   private:
       int resistance;
   public:
       resistor();
       ~resistor();
       int getResistance() const;
       void setResistance(int r);
       virtual void print() const;
};

class powerResistor : public resistor {
   private:
       float power;
   public:
       powerResistor();
       ~powerResistor();
       float getPower() const;
       void setPower(float p);
       virtual void print() const;
};

class capacitor : public circuitElement {
   private:
      int capacitance;
   public:
       capacitor();
       ~capacitor();
       int getCapacitance() const;
       void setCapacitance(int r);
       virtual void print() const;
};

class powerCapacitor : public capacitor {
   private:
       float power;
   public:
       powerCapacitor();
       ~powerCapacitor();
       int getPower() const;
       void setPower(float p);
       virtual void print() const;
};
```

Assume the implementation of these classes is correct even though the implementation code is not shown here.

In the table below, indicate whether each group of statements (i.e., a row in the table) generate a compiler error or not by placing an `X` in the appropriate column. <u>All statements appear in the `main` function of a program. Consider each group of statements (i.e., row in the table) by itself.</u>

Statement	Error?	Not Error?
`circuitElement e;`		

`resistor r;`
`powerResistor pr;`
`pr = r;`		

`resistor r;`
`powerResistor pr;`
`r = pr;`		

`resistor r;`
`capacitor c;`
`r = c;`		

`powerCapacitor pc;`
`int x = pc.getID();`		

`circuitElement* elm_p;`		

`resistor* r_p;`
`r_p = new resistor();`		

`circuitElement* elm_p = new resistor();`		

`powerResistor* pr_p;`
`resistor* r_p = new resistor();`
`pr_p = r_p;`		

`powerResistor* pr_p = new powerResistor();`
`resistor* r_p;`
`r_p = pr_p;`
`r_p->getPower();`		

`powerResistor* pr_p = new powerResistor();`
`resistor* r_p;`
`r_p = pr_p;`
`r_p->print();`		

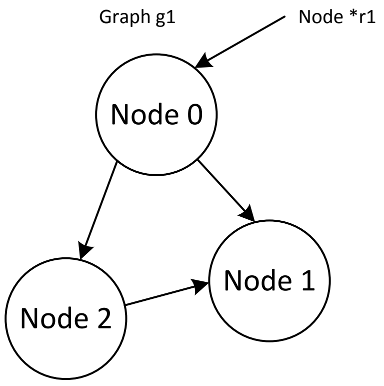
Question 10. (12 marks). *Trees and Graphs.*

Consider the two classes below, which are used to store a graph.

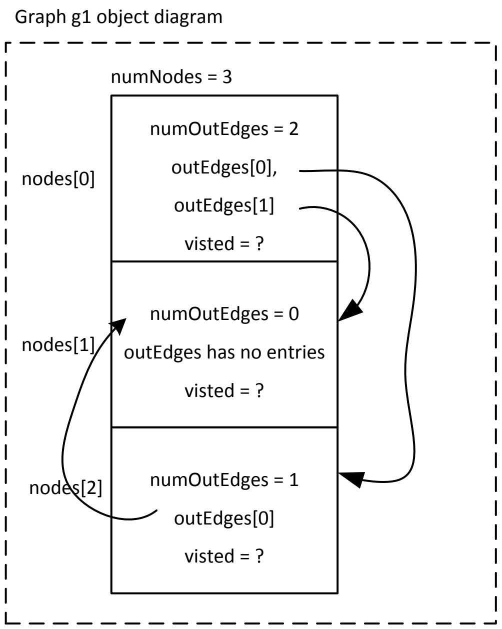
```
class Node {
private:
   int numOutEdges;   // Number of edges leaving this node.
   Node** outEdges;   // array [0..numOutEdges-1] of Node*
   bool visited;      // Temporary data you can use.
Public:
   int getNumOutEdges () const {return (numOutEdges); }
   Node* getOutEdge (int iedge) const {return (outEdges[iedge]); }
   bool getVisited () const {return (visited); }
   void setVisited (bool val) { visited = val; }
   ...   // Constructors not shown
};


class Graph {
private:
   int numNodes;  // Total number of nodes in the graph
   Node* nodes;   // array [0..numNodes-1] of the nodes in the graph
   // You can make private helper functions if you wish.
public:
   bool isTree (Node *root);
   ... // Constructors and destructor not shown.
};
```

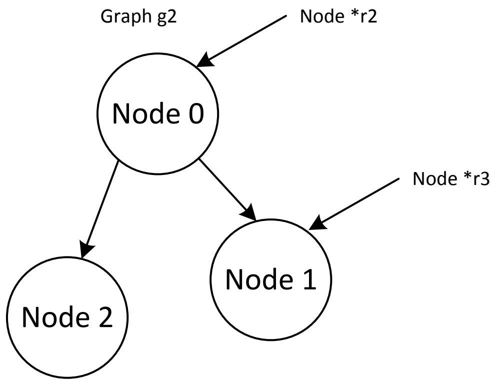
Write the code for the member function `isTree`. This function should return true if the **entire** graph is a tree rooted at the node pointed to by `root`, and false otherwise. You can add private helper functions to class `Graph` if you wish, but do not need to (and should not) add any data or public functions to class `Graph`. You also should not make any changes to class `Node`. Your solution must be no more than O(n) complexity, where n is the number of nodes in the graph.

The diagrams below show an example of the data stored in a Graph object for a simple example, and show several graphs and what `isTree(root)` should return for each.



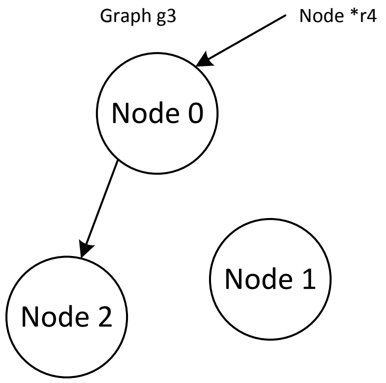
g1.isTree (r1) returns false





g2.isTree (r2) returns true

g2.isTree (r3) returns false



g3.isTree (r4) returns false

Your answer:

Question 11. (8 marks). *Trees, recursion, complexity.*

Consider the following class used for nodes in an arbitrary shape binary tree:

```
class TreeNode {
  private:
    int value ;
    TreeNode * left ;
    TreeNode* right ;
  public:
    .
    .
    .
} ;
```

Now consider the following method of `TreeNode` that calculates the height of the sub-tree where the node on which the method is invoked is the root of the sub-tree:

```
int TreeNode::height() {
  int rheight = 0 ;
  int lheight = 0 ;
  if( left != NULL ) lheight = left->height() ;
  if( right != NULL ) rheight = right->height() ;
  return max( rheight, lheight ) + 1 ;
}
```

and the following method of `TreeNode` that determines whether the sub-tree is balanced where the node on which the method is invoked is the root of the sub-tree:

```
bool TreeNode::balanced() {
  int lheight = 0 ;
  int rheight = 0 ;
  if( left != NULL ) {
    if( ! left->balanced() ) return false ;
    lheight = left->height() ;
  }
  if( right != NULL ) {
    if( !right->balanced() ) return false ;
    rheight = right->height() ;
  }
  if( abs( lheight - rheight ) > 1 ) return false ;
  else return true ;
}
```

(a) (2 marks). What is the best case complexity of `height()`?

T(n) = O (   )

(b) (2 marks). What is the worst case complexity of `height()`?

T(n) = O (   )

(c) (2 marks). What is the best case complexity of `balanced()`?

T(n) = O (   )

(d) (2 marks). What is the worst case complexity of `balanced()`?

T(n) = O (   )

Question 12. (6 marks). *Complexity Analysis*.

(a) (3 marks). Give the time complexity of the following segment of code.

```
float sum = 0;
for (int i = 0; i < n; i++) {
   for (int j = i; j > 0; j--)
       sum += j * i;
}
```

T(n) = O (   )

(b) (3 marks). Give the time complexity of the following segment of code.

```
for(int i=0; i<10; i++)
   for(int j=0; j<N; j++)
      for(int k=N-2; k<N+2; k++)
         cout << i << " " << j << endl;
```

T(n) = O (   )

Question 13. (3 marks). *Hash Tables*.

Assume a hash table with size T = 11 with the following hash function:

h: index = (key mod T) (note: mod is the modulus operator, % in C++)

Show the contents of the hash table after the following operations have been performed. Indicate next to each operation the number of ***probes*** performed. Assume that linear probing is used to resolve collisions and that the hash table is initially empty.

```
1. insert 14
2. insert 26
3. insert 4
4. delete 26
5. search 4
6. insert 37
```

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Question 14. (5 marks). *Debugging.*

The code below implements a doubly-linked list in classes `Node` and `List`. The `main()` function uses this doubly linked list, and the entire program is stored in `test.cpp`. When `test.cpp` is compiled and the resulting executable is run through *valgrind*, memory errors are reported. Fill in the table below indicating how to fix the `Node` and/or `List` classes. You should not make any changes to the `main()` function.

```
 1    #include <iostream>
 2    using namespace std;
 3
 4    class Node {
 5    private:
 6       int key;
 7       Node *next;
 8       Node *prev;
 9    public:
10       Node () {key = 0; next = NULL; prev = NULL; }
11       Node (int _key, Node *_next, Node *_prev);
12       int getKey () { return (key); }
13       void setKey (int _key) { key = _key; }
14       Node *getNext () { return (next); };
15       void setNext (Node *_next) { next = _next; }
16       Node *getPrev () {return (prev); } ;
17       void setPrev (Node *_prev) { prev = _prev; }
18    };
19
20    Node::Node (int _key, Node* _next, Node *_prev) {
21       key = _key;
22       next = _next;
23       prev = _prev;
24    }
25
26
27    class List {
28    private:
29       Node *head;
30       Node *tail;
31    public:
32       List ();
33       ~List ();
34       void insert (int key);
35       void insertBack (int key);
36       Node *find (int key);
37    };
38
39    List::List () {
40       head = NULL;
41    }
42
43    void List::insert (int key) {
44       Node *newNode = new Node (key, head, NULL);
45       if (head != NULL){ // Not an empty list
46          head->setPrev (newNode);
47          head = newNode;
48      }
49      else {  // Empty list case.
50          head = newNode;
51       }
52    }
53
54    void List::insertBack (int key) {
55       Node *newNode = new Node (key, NULL, tail);
56       if (tail != NULL) {  // Not an empty list
57          tail->setNext (newNode);
58          tail = newNode;
59       }
60       else {  // Empty list case.
61          head = newNode;
62          tail = newNode;
63       }
64    }
65
66    List::~List ()
67    {
68       Node *currNode, *prevNode;
69
70       currNode = head;
71       while (currNode != NULL) {
72          prevNode = currNode;
73          delete prevNode;
74          currNode = currNode->getNext ();
75       }
76    }
77
78    int main () {
79       cout << "Starting main\n";
80       List *lptr = new List;
81       lptr->insert (5);
82       delete lptr;
83       List l1;
84       l1.insert (20);
85       l1.insertBack (3);
86       cout << "Ending main\n";
87    }
```

Valgrind output:

```
==3458== Memcheck, a memory error detector
Starting main
==3452== Invalid read of size 4
==3452==    at 0x80489FC: Node::getNext() (test.cpp:14)
==3452==    by 0x80488A3: List::~List() (test.cpp:74)
==3452==    by 0x8048912: main (test.cpp:82)
==3452==  Address 0x42d3064 is 4 bytes inside a block of size 12 free'd
==3452==    at 0x4024851: operator delete(void*) (vg_replace_malloc.c:387)
==3452==    by 0x8048898: List::~List() (test.cpp:73)
==3452==    by 0x8048912: main (test.cpp:82)
==3452==
==3452== Conditional jump or move depends on uninitialised value(s)
==3452==    at 0x8048803: List::insertBack(int) (test.cpp:56)
==3452==    by 0x804894E: main (test.cpp:85)
==3452==
==3452== Use of uninitialised value of size 4
==3452==    at 0x8048A0B: Node::setNext(Node*) (test.cpp:15)
==3452==    by 0x8048819: List::insertBack(int) (test.cpp:57)
==3452==    by 0x804894E: main (test.cpp:85)
==3452==
==3452==
==3452== Process terminating with default action of signal 11 (SIGSEGV)
```

Fix the program by filling in the table below. You may not need to use all the rows. For each row, indicate the line number of the change, whether to remove that line of code, change that line, or add code before that line. For changed or added lines, indicate the new line(s) of code as well.

Line Number	**Remove / Change / Add Before** (choose one)	**New Code** (for Changed or Added Lines)