University of Toronto
Faculty of Applied Science and Engineering

ECE 244F

PROGRAMMING FUNDAMENTALS

Fall 2015

Final Examination

Examiners: Michael Stumm and Hamid Timorabadi

Duration: Two and a Half Hours

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

Do not remove any sheets from this test. All questions must be answered in the space provided.

Work independently. The value of each question is 5. The total value of all questions is 100.

Write your name and student number in the space below. Do the same on the top of each sheet of this exam book.

Name: ______________________________
(Underline last name)

Student Number: ______________________________

Q1.	________	Q11.	________		
Q2.	________	Q12.	________		
Q3.	________	Q13.	________		
Q4.	________	Q14.	________		
Q5.	________	Q15.	________		
Q6.	________	Q16.	________		
Q7.	________	Q17.	________		
Q8.	________	Q18.	________		
Q9.	________	Q19.	________	Total	
Q10.	________	Q20.	________		

Question 1. (5 marks). *Warmup questions.*

a) How many plusses ('+') does the name of the programming language have that you have been using in this class?

Answer:

b) $2^x = z$. Solve for x:

Answer:

x =

c) What is base 2 log of 128?

Answer:

d) How many members does an object of class B have if defined as below:

Answer:

```
class A{  int a ; } ;
class B : public A{ int a ; } ;
```

e) In the following recursive implementation of Fibonacci, a call to `fib` with 4 as an argument, `fib(4)`, will generate how many calls to `fib`?

Answer:

```
int fib( int n ) {
    if( n==0 ) return 0 ;
    if( n==1 ) return 1 ;
    return fib(n-1)+fib(n-2) ;
}
```

Question 2. (5 marks). *More warmup questions.*

Please circle the correct answer:

a) True or False:

An $O(\log(n))$ algorithm is always faster than an $O(n)$ algorithm when $n > 1$?

b) True or False: The complexity of the following function is $O(2^n)$.

```
int Boo( int n ) {
    if (n == 4) return (2);
    else return (2* Boo(n+1));
}
```

c) True or False:

The complexity of `HashTable::Insert( int v )` which inserts the value `v` into a closed hash table using linear probing is always $O(1)$.

d) Control-C or Control-D:

You are testing a program that reads from the standard input until end of file is reached. When you have completed providing input, what do you type?

e) True or False:

A derived class constructor must always call the default base class constructor to initialize and possibly allocate memory for the base object.

Question 3. (5 marks). *Simple complexity analysis.*

Using big-O notation, what is the complexity of the following algorithms:

a) factorial:

```
int fact( int n ) {
    int f = 1 ;
    for( int i=n; i>1; i-- )
        f *= i ;
    return f ;
}
```

Answer:
T(n) = O(   )

b) summation of the first n numbers (due to Gauss):

```
int sum( int n ) {
    return n * (n+1) / 2 ;
}
```

Answer:
T(n) = O(   )

c) Fibonnaci

```
int fib( int n ) {
    int *fibarray = new int[n] ;
    fibarray[0] = 0 ;
    fibarray[1] = 1 ;
    for( int i=2; i<n+1; i++ )
        fibarray[i] = fibarray[i-1] + fibarray[i-2] ;
    return fibarray[n] ;
}
```

Answer:
T(n) = O(   )

Question 4. (5 marks). *Yet more complexity analysis.*

Suppose that you have an array `A` of integers; the integers are not in any sorted order, and the array may contain duplicate items. Three algorithms are described below that copy all items of array A into another array `B`, but in a way that prevents items that may be duplicated in A from being duplicated in `B`. What is the **average** running time in big-O notation of each algorithm? State any assumptions you make.

a) For each item in array `A`, copy the item into another array `B` unless the item already exists in `B`.

b) Sort `A` using Quicksort, and then for each item in array `A`, copy the item into another array `B` unless the item already exists in `B`, which can now be determined with just one lookup.

c) Insert each element of `A` into a hash table unless a duplicate item already exists in the hash table. At the end, copy all items from the hash table into array `B`.

Question 5. (5 marks). *Complexity analysis of recursive functions. (Some parts of this question may take some time to solve.)*

Assume that the function `lini( int n )` in the algorithms below has a linear running time, $T(n) = O(n)$, so that $T(n) \le c \cdot n$ for some constant c.

Using big-O notation, what is the running time of the function `f( int n)` in the following five cases?

a)
```
void f( int n ) {
    if( n > 0 ) {
        f( n/3.4 ) ;
    }
}
```

Answer:

T(n) = O(   )

b)
```
void f( int n ) {
    if( n>0 ) {
        lini( n ) ;
        f( n/2 ) ;
    }
}
```

Answer:

T(n) = O(   )

c)
```
void f( int n ) {
    if( n > 0 ) {
        lini( n ) ;
        f( n/2 ) ;
        f( n/2 ) ;
    }
}
```

Answer:

T(n) = O(   )

d)
```
void f( int n ) {
    if( n > 0 ) {
        lini( n ) ;
        f( n-1 ) ;
    }
}
```

Answer:

T(n) = O(   )

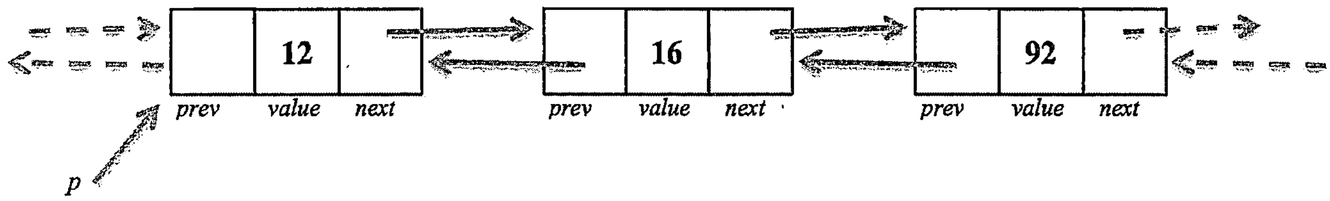
e)
```
void f( int n ) {
    if( n > 0 ) {
        lini( n ) ;
        f( n-1 ) ;
        f( n-1 ) ;
    }
}
```

Answer:

T(n) = O(   )

Question 6. (5 marks). *Pointers and Classes.*

Doubly-linked lists were not covered in the lectures in any detail. Nevertheless, nodes in doubly-linked lists also contain *prev* pointers that point to the previous node in the list (in addition to *next* pointers that point to the next node in the list) as in the following diagram:



A novice programmer wrote the following particularly poor implementation of class `DLNode` that uses casting, where "`(DLNode*) p`" converts the pointer `p` to a point to a `DLNode` object. Moreover, the implementation relies on default destructors:

```
class LLNode {
  private:
    int value ;
    LLNode *next ;
  public:
    LLNode* getNext() { return next ; }
    void setNext( LLNode *n )
      { next = n ; }
    ...
} ;
```

```
class DLNode: public LLNode {
  private:
    DLNode * prev ;
  public:
    DLNode* getNext()
      {return (DLNode *) LLNode::getNext(); }
    DLNode* getPrev() {return prev ; }
    void setPrev( DLNode *p )
      { prev = p ; }
    ...
} ;
```

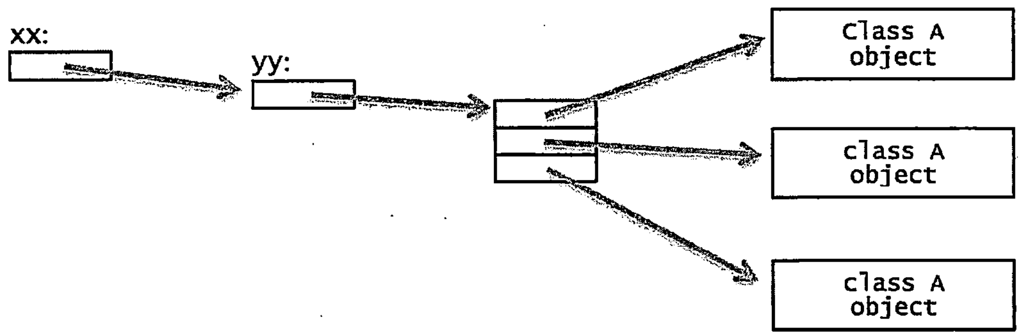
Further, assume you have been passed a `DLNode` pointer, `p`, that points to an element which resides somewhere in the middle of a very large list. Your task is to write the lines of code which would free the space of the node pointed to by p (the node with value 12), using the `delete` operator, after properly removing it from the list. The remaining elements in the list must be properly linked to one another after the deletion.

Note that you are **not** being asked to write a complete function: just the lines deleting the appropriate node (i.e., the `DLNode` with value=12) from the list that is shown above, starting only with `p`. You may <u>**not**</u> allocate or use any additional variables (including pointers). Assume the code does not belong to a member function; i.e., private data members and methods cannot be accessed.

Your code:

Question 7. (5 marks) *Pointers.*

Consider the following data elements. Note that some have a name at the top-left (i.e., xx and yy), while others do not. All of the elements are pointers, except the objects depicted on the right hand side and denoted as class `A` objects. Assume that class `A` has been declared already.

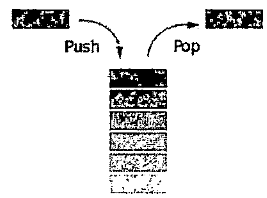


In the space below, provide the declarations and statements needed to create the data structure shown above. The number of statements used should be minimal; i.e., marks will be deducted for extra statements. Elements without a name in the figure may not have a name in your code.

Answer:

Question 8. (5 marks). *Linked Lists.*

Linked lists can be used to implement various data structures. Here, we consider the use of linked lists to implement a stack. A stack is a LIFO (last in, first out) abstract data type which is used to maintain a collection of objects. The stack has two principal operations: *push*, which adds an element to the collection at the top of the stack, and *pop*, which removes the element at the top of the stack.



Assume class Node has been defined as in class with two private data members, `int value` and `Node * next`, and with the following constructors:

```
Node() ;
Node( int v ) ;
Node( int v, Node* n ) ;
```

Moreover, class Stack is declared as a friend. Assume class Stack is defined as:

```
class Stack {
  private:
    Node * sp ;
  public:
    Stack() { sp = NULL ; }
    ~Stack() { delete sp ; }
    void push( int value ) { sp = new Node( value, sp ) ; }
    Node * pop() ;
} ;
```

In the space below, implement `Stack::pop()` that removes the node at the top of the stack and returns a pointer to the node if it exists and returns `NULL` if it does not exist. You are to implement this by adding at most five lines of code in the space below:

Answer:

```
Node * Stack::pop()  {



}
```

Question 9. (5 marks). *Argument passing.*

What does the following program output:

```
void func(int*& a, int* b, int* c) {
    a[0] = 5;   a[1] = 6;
    c[0] = 7;   c[1] = 8;

    a = c;

    a[0] = 9;   a[1] = 10;
    b[0] = 11;  b[1] = 12;
}

int main() {
    int* m = new int[2];
    int* n = new int[2];

    m[0] = 1;    m[1] = 2;
    n[0] = 3;    n[1] = 4;

    func(m, m, n);

    cout << m[0] << "-" << m[1] << "-" << n[0] << "-" << n[1] << endl;
}
```

Answer:

Question 10. (5 marks). *Memory Management.*

Assume that variables of type `int` or `float` each consume 4 bytes of memory and that each pointer also consumes 4 bytes of memory. How much memory space (in bytes) will the following program have allocated – whether on the stack or dynamically, and whether leaked or not – by the time we reach **`Point A`** of the program, and how much memory space will have leaked by the time the program reaches **`point A`**:

```
class A {
    int numItems;
    int* items;
    A( int n ) {
        numItems = n ;
        items = new int[numItems] ;
    }
} ;
int main() {
    A* firstObject;
    for( int i=1; i<4; i++ ) {
      firstObject = new A( i ) ;
    }
    // Point A
}
```

Total memory allocated by time we reach point A: bytes	*Leaked memory by the time we reach point A:* bytes

Question 11. (5 marks). *Operator Overloading.*

One of the disadvantages of C and C++ arrays is that they have no bounds checking. This results in erroneous program behavior and crashes when the program contains off-by-one errors when indexing into arrays. For this reason, you wish to implement a "smart array" of objects of type `T`. The goal is to automatically check the bounds of the array when indexing into the array. The declaration of your smart array starts as follows:

```
class SmartTarray {
  private:
    T *array;
    int size;
  public:
    SmartTarray( int arraySize ) ;
    ...
};
```

You may assume that class `T` and the constructors/destructor of class `SmartTarray` are properly implemented. Write the `operator[]` function that is used to index into the array and returns a copy of the indexed `T` object if successful. If the array is accessed with an index out of bounds, it should output the error message "`out of bound access`", and in that case any value can be returned.

Write your answer in the box below.

Answer:

Question 12. (5 marks). *Tree Traversals.*

Assume TreeNode is a class as defined in the lectures with data members consisting of `value`, `left` and `right`. Further, assume the following function that traverses a binary tree:

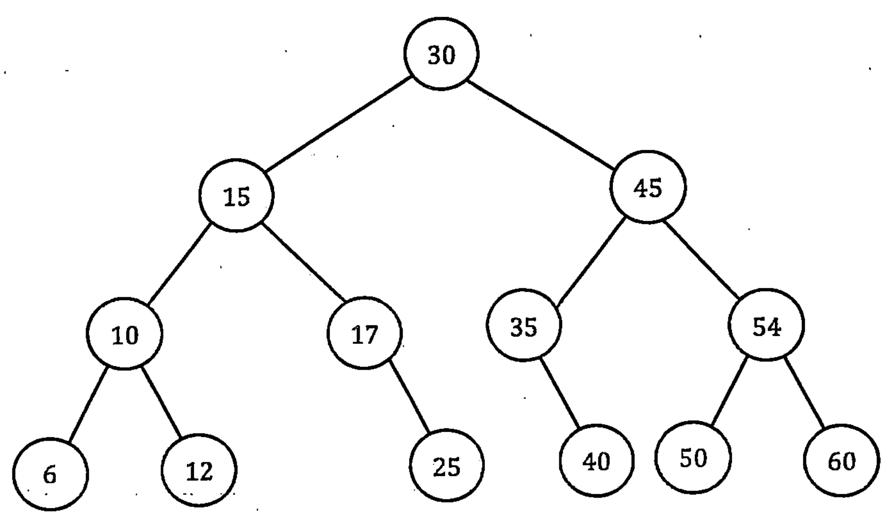
```
void TreeNode::wtraverse() {
    cout << this->value << " - " ;
    if( right != NULL ) right->wtraverse()  ;
    if( left != NULL ) left->wtraverse() ;
}
```

What is output when the function `wtraverse()` is called on the node labeled "15" in the tree shown below (in the next question, before any modifications to the tree)?

Answer:

Question 13. (5 marks). *Binary Search Trees.*

The following is a binary search tree.



Assume that the node with the key "45" is deleted from the tree and then subsequently re-inserted as a new node (also with the key "45"). Draw changes to the tree above that result from the deletion followed by the insertion. You can make the changes directly in the diagram of the tree above.

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

Consider the following code fragment. First cross out (~~like this~~) all lines that are incorrect; i.e., lines that might cause a compiler error or a run-time error. Then determine what output generated by the following code?

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

class B {
  public:
     B() ;
     virtual void Foo() ;
} ;

class D : public B {
   public:
        D() ;
        void Foo() ;
} ;

B::B() { this->Foo() ; }

void B::Foo() { cout << "B::Foo()" << endl ; }

D::D() { this->Foo(); }

void D::Foo() { cout << "D::Foo()" << endl; }

int main() { D objectD; }
```

Output:

Question 15. (5 marks). *More inheritance.*

Consider the following code fragment. First cross out (~~like this~~) all lines in `main()` that are incorrect; i.e., lines that might cause a compiler error or a run-time error. Then, provide the output generated by the code (assuming the crossed-out lines are not executed). Note that the three classes have no errors.

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

class B {
  protected:
    virtual void Foo() { cout << "B::Foo()" << endl ; }
  public:
    void Boo() { Foo() ;  cout << "B::Boo()" << endl ; }
    virtual void Hoo() { /* deliberately left empty */ }
} ;

class D : public B {
  protected:
    virtual void Foo() { cout << "D::Foo()" << endl ; }
  public:
    void Boo() { Foo() ;  cout << "D::Boo()" << endl ; }
    virtual void Hoo() = 0 ;
} ;

class E : public D {
  public:
    void Foo() { cout << "E::Foo()" << endl ; }
    void Boo() { Foo() ; cout << "E::Boo()" << endl ; }
    void Hoo() { /* deliberately left empty */ }
} ;

int main() {
    B bObj, *bPtr ;
    D dObj, *dPtr ;
    E eObj, *ePtr ;

    bPtr = &eObj ;
    ePtr = &eObj ;

    bPtr -> Boo() ;
    bPtr -> Hoo() ;

    eObj.Boo() ;
    eObj.Hoo() ;
}
```

Output:

Question 16. (5 marks). *Hash tables.*

Assume you are given a closed hash table with size M = 11, and the following hash function:

`h(key) = (key % M)` (Note: the "%" is the modulus operator in C++)

Assume that **quadratic probing** is used to resolve key collisions, and that the hash table is initially empty. If i is the initial hash index generated by the hash function above, quadratic probing examines locations $i + k^2$, where $k = 0, 1, 2, 3, \ldots$ until an empty location is found.

Show the final contents of the hash table after all of the following operations have been performed, in the sequence shown. Next to each operation, indicate the number of probes performed when inserting that value, where each required examination of a hash table entry is considered to be a *probe*. (Hence, the first operation, `insert 7`, requires one probe.)

Operation:	Probes:
`insert 7`	1
`insert 3`	
`insert 18`	
`insert 29`	
`insert 14`	
`insert 40`	
`insert 25`	

Hash table after all insertions:

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Question 17. (5 marks). *Surprise: yet another complexity question.*

What is the **worst case** running time in Big-O notation of the function `processArray( int data[], int a, int b )` of the next following Question 18, where n is the size of the data array.

Answer:

T(n) = O(   )

Question 18. (5 marks). *Recursion.*

What does the following code snippet output:

```
void processArrayMore( int data[], int a, int c, int b ) {
    if( data[a] > data[c+1] )
        swap( data[a], data[c+1] ) ;
}

void processArray( int data[], int a, int b ) {
    if( a <= b ) return ;
    int c = (a+b)/2 ;

    processArray( data, a, c ) ;
    processArray( data, c+1, b ) ;
    processArrayMore( data, a, c, b ) ;
}

int main() {
    int data[8] = { 4, 10, 3, 16, 2, 1, 5, 7 } ;

    processArray( data, 0, 7 ) ;

    cout << data[0] << endl ;

    int data2[8] = { 1, 2, 3, 4, 5, 6, 7, 8 } ;

    processArray( data2, 0, 7 ) ;

    cout << data2[0] << endl ;
}
```

(The code `int data[8] = {1, 2, 3...}` initializes array `data` such that the first element (`data[0]`) obtains the value 1, the second element (`data[1]`) obtains the value 2, etc.

Answer:

Question 19. (5 marks). *Constructors.*

Assume that class A has the following constructors and operators defined:

```
A() ;
A( int, float ) ;
A( const A& ) ;
A& operator=( const A& ) ;
```

And assume that class B has the following constructors defined:

```
B() ;
B( string ) ;
B( const E & ) ;
B( const B & ) ;
```

Finally, assume that class D is declared in part as follows:

```
class D : public B {
  protected:
    A a ;
    A *ap ;  // a pointer to a single A object (or NULL).
  public:
    .
    .
    .
} ;
```

Class D does not contain any additional data members beyond a and ap. In the space below, please provide the definition (i.e., implementation) of a copy constructor for class D.

D's Copy Constructor:

Question 20. (5 marks). *Graphs.*

Consider the two classes below that are used to implement an undirected, connected graph:

```
class GNode{
  private:
    int value ;
    int numEdges ;     // number of edges connected to this node
    GNode ** edges ;   // array of edges; i.e., array of GNode *
    bool visited ;     // temporary data for traversing graph
  public:
    int getValue() const { return value ; }
    void setValue( int v ) { value = v ; }
    int getNumEdges() const { return numEdges ; }
    GNode* getEdge( int i ) const { return edges[i] ; }
    bool getVisited() const { return visted ; }
    void setVisited( bool val ) { visited = bal ; }
    . . . // constructors and destructor not shown
} ;

class Graph{
  private:
    GNode *anchor ;
  public:
    GNode* getAnchor() const { return anchor ; }
    void setAnchor( GNode* a ) { anchor = a ; }
    . . . // many other methods, including constructors and destructor
} ;
```

Note that a graph object simply points to some node in the graph (if one exists) that we refer to as the anchor node and that all other nodes of the graph can be reached from the anchor node. In the space below and on the next page, implement a function that is a member function of class Graph and that returns the number of nodes in the graph that have the value v:

```
int Graph::count( int v )
```

You may add helper member functions to either of the GNode or Graph classes or both, but you may not add any new data members and you may not use global variables.