ECE 250
Data Structures and Algorithms
MID-TERM EXAMINATION B

2011-02-15/13:30-14:50
MC-4021/RCH-211

Instructions:

There are 63 marks. It will be marked out of 55.
No aides.
Turn off all electronic media and store them under your desk.
If there is insufficient room, use the back of the previous page.
You may ask only one question during the examination:
"May I go to the washroom?"
Asking **any** other question **will** result in a deduction of 5 marks from the exam grade.
If you think a question is ambiguous, write down your assumptions and continue.
Do not leave during the examination period.
Do not stand up until all exams have been picked up.

Attention:

The questions are in the order of the course material, not in order of difficulty.

Surname/Last Name								
Legal Given/First Name(s)								
UW Student ID Number								
UW User ID								

I have read the above instructions:

Signature: _____

Asking Any Question
-5

Relationships

1*a*. **[1]** Pre-requisites for classes prevent students from taking specific classes until other specified classes are taken. Is this an example of a partial ordering, a hierarchical ordering, or an equivalence relation?

1*b*. **[1]** In Unix, there is a single directory / of which all other directories are subdirectories thereof. What relationship is this an example of?

Runtime Analysis

2. **[4]** Create a binary search tree by placing the following seven items in the given order into an initially empty binary search tree where we use the relationship $f(n) < g(n)$ if $f(n) = \omega(g(n))$. For example, all objects in the left sub-tree should be little-omega of n^2.

$$n^2,\ \sqrt{n}\ ,\ n\ln(n),\ 1,\ 3n,\ \lg(n),\ \ln^2(n)\ n$$

Note: $\ln^2(n) = \ln(n) \times \ln(n)$.

3. **[3]** Determine, using limits, which Landau symbol, **o, O**, **Θ**, **ω**, or **Ω**, best describes the relationship between $n^2 \ln(n)$ and e^n. State your answer in the form $n^2 \ln(n) = \mathbf{?}(\ e^n\ )$. You must use l'Hopital's rule where necessary.

4. **[3]** The following is a skeleton of Gaussian elimination. What is the run time of this code using big-**Θ** notation?

```
for ( int i = 1; i <= n; ++i ) {
      for ( int j = i + 1; j <= n; ++j ) {
            // Do some Theta(1) work
            for ( int k = i; k <= n; ++ k ) {
                  // Do some Theta(1) work
            }
      }
}
```

If it is known that all the entries sufficiently far away from the diagonal are zero, the code simplifies to the following. What is the run time of this code using big-**Θ** notation?

```
for ( int i = 1; i <= n; ++i ) {
      for ( int j = i + 1; j <= i + 2; ++j ) {
            // Do some Theta(1) work
            for ( int k = i; k <= i + 2; ++ k ) {
                  // Do some Theta(1) work
            }
      }
}
```

5. **[3]** Using a proof by induction, show that $\sum_{k=0}^{n}\left(2k+1\right)=\left(n+1\right)^2$.

Arrays and Linked Lists

For your information, these are the declarations of the **`Double_list`** and **`Double_node`** classes from Project 1.

```
template <typename Object>
class Double_list {
    private:
          Double_node<Object> *list_head;
          Double_node<Object> *list_tail;
          int node_count;

    public:
          Double_list();
          Double_list( const Double_list & list );
          ~ Double_list();

          Double_list & operator = ( const Double_list &rhs );

          int size() const;
          bool empty() const;
          Object front() const;
          Object back() const;
          Double_node<Object> *head() const;
          Double_node<Object> *tail() const;

          int count( const Object &obj ) const;

          void push_front( const Object &obj );
          void push_back( const Object &obj );
          Object pop_front();
          int erase( const Object &obj );
};

template <typename Object>
class Double_node {
    private:
          Object       element;
          Double_node *previous_node;
          Double_node *next_node;

    public:
          Double_node( const Object &e = Object(), Double_node *p = 0, Double_node *n = 0 );
          Object retrieve() const;
          Double_node *next() const;
          Double_node *previous() const;

          friend class Double_list<Object>;
};
```

6. **[6]** The following attempts to implement a reversing of the elements in the list; however, the code fails to achieve this. Fix it. You may not create (`new`) or destroy (`delete`) any nodes.

```
template <typename Object>
void Double_list<Object>::reverse() {
      for ( Double_node<Object> *p = head(); p != 0; p = p->next() ) {
            Double_node<Object> *t = p->next();
            p->next_node = p->previous();
            p->previous_node = t;
      }

      Double_node<Object> *t = head();
      list_head = tail();
      list_tail = t;
}


template <typename Object>
void Double_list<Object>::reverse() {
```

7. **[1]** Name an operation on a non-full two-ended array that may run in $\Theta(1)$ time but that must run in $\Theta(n)$ time if it is implemented in a non-full one-ended array.

Stacks and Queues

8. **[4]** A queue may be implemented with a singly linked list or a circular array. What situation must be considered with arrays that does not cause a problem when using linked lists? List three possible solutions.

9. **[2]** Why would you never use a singly linked list for a deque? Be sure to include a discussion of asymptotic run times in your answer.

10. **[3]** The expression `a = (b + c) * d` must be converted to the reverse Polish format `b c + d * &a =` before it can be converted into machine instructions as operands must be loaded into registers before an operation can be performed on those registers. Explain why we must access the address of `a` before executing the assignment operation.

Tree Definitions

11. **[2]** Define the height of a tree in terms of the depth of nodes within a tree..

12. **[3]** Pairs of nodes are said to be *cousins* if they share the same grandparent (the parent of the parent) but they don't share the same parent.

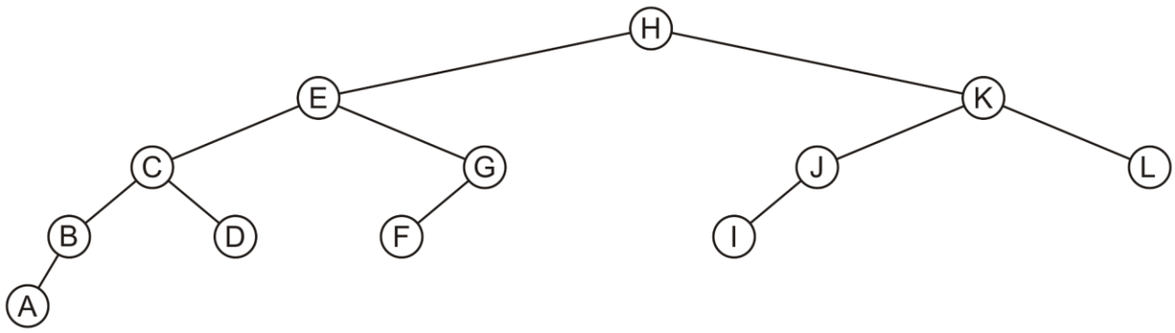

Figure 12. A tree.

List all pairs of cousins found in the tree in Figure 12.

13. **[4]** Implement a recursive function that returns the size (number of nodes) within a tree.

```
template <typename Object>
int General_tree<Object>::size() {



      for (
            Single_node< General_tree<Object> > *p = children.head();
            p != 0;
            p = p->next()
      ) {




      }


      return
}
```

Tree Traversals

13. **[2]** Perform a post-order traversal of the tree shown in Figure 13 and place your answer in Table 13.

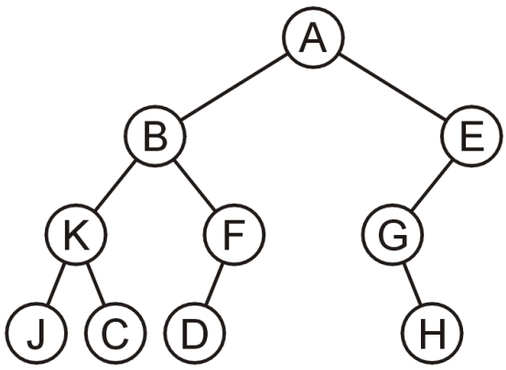

Figure 13. A tree.

Table 13. Post-order depth-first traversal.

14. **[4]** Under what conditions will the pre-order depth-first traversal and the bread-first traversal of a tree be the same?

Binary Search Trees

15. **[4]** In the given order, insert the following integers into a binary search tree:

34, 15, 65, 59, 69, 42, 40, 80, 50, 66

16. **[2]** Remove 34 from the binary search tree in Question 15 copying up an appropriate element from the **right** subtree.

AVL Trees

17. **[6]** The following are three insertions into the AVL tree shown in Figure 16. At each step, you only need to redraw that component of the tree necessary to demonstrate both the insertion and any rotations.

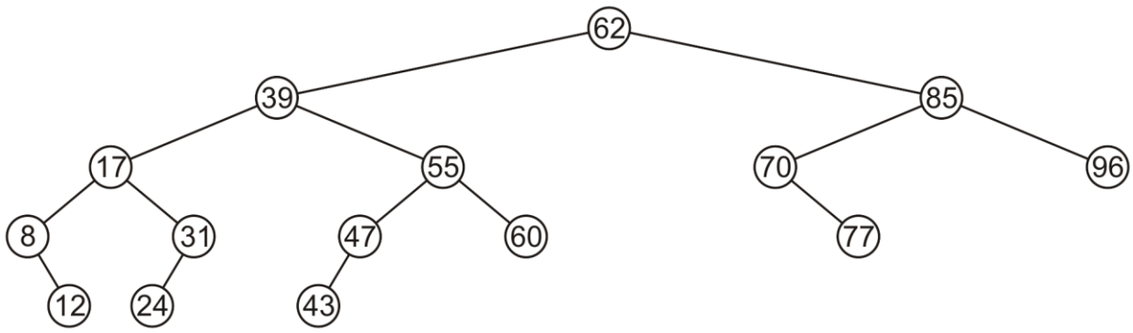

Figure 17. An AVL binary search tree.

a. Given the AVL tree in Figure 17, insert a node containing 45 and perform any rotations necessary to maintain the AVL balance.

b. Given the AVL tree in Figure 17, insert a node containing 6 followed by the insertion of a node containing 5. Perform any rotations necessary to maintain the AVL balance.

c. Given the AVL tree in Figure 17, insert a node containing 79 and perform any rotations necessary to maintain the AVL balance.

Other Questions

18. **[3]** A binary search tree has insert implemented as follows:

```
template <typename Object>
bool Binary_search_tree::insert( const Object &obj ) {
    if ( empty() ) {
        root = new Binary_search_node<Object>( obj );
        return true;
    } else {
        return root->insert( obj );
    }
}
```

If the argument already exists in the tree, it is not inserted and `false` is returned. If the argument is not yet in the tree, it is inserted into the appropriate location and `true` is returned. The constructor of a binary search node creates a node with the pointers to the left and right subtrees set to zero. Correct the following code to implement the desired behaviour.

```
template <typename Comp>
bool Binary_search_node<Comp>::insert( const Comp &obj ) {
    if ( obj < retrieve() ) {
        return left()->insert( obj );
    } else if ( obj > retrieve() ) {
        return right()->insert( obj );
    }

    return false;
}


template <typename Object>
bool Binary_search_node::insert( const Object &obj ) {









}
```

19. **[2]** What traversal of an expression tree must be performed in order to convert the expression tree into the format necessary for conversion of the expression tree into machine instructions?