CS 308 Data Structures

Spring 2003 - Dr. George Bebis

Final Exam

Duration: noon - 2:00 pm

Name:

1. **True/False** (3 pts each) **To get credit, you must give brief reasons for your answers !!**

(1.1) **T F** An inorder traversal always processes the elements of a tree in the same order, regardless of the order in which the elements were inserted.

(1.2) **T F** The running time Reheap-Down in $O(logN)$ where N is the number of elements in the heap.

(1.3) **T F** The largest value in a binary search tree is always stored at the root of the tree.

(1.4) **T F** An $O(logN)$ algorithm is slower than an $O(N)$ algorithm.

(1.5) **T F** Inserting an element onto an unsorted list takes $O(1)$ time.

(1.6) **T F** To delete a dynamically allocated tree, the best traversal method is *postorder*

(1.7) **T F** When building a binary search tree, the order in which elements are inserted in the tree is unimportant.

(1.8) **T F** Derived classes have access to the private members of their base classes.

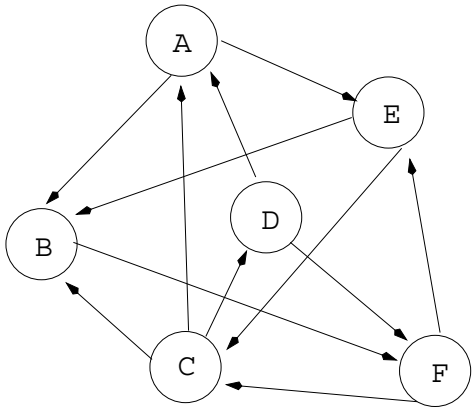
(1.9) **T F** Suppose that a complete binary tree containing 85 elements is stored in an array *tree[]*. The subtree rooted at tree[10] is a full binary tree with four levels.

(1.10) **T F** Implementing a priority queue using a heap is more efficient than using a linked-list.

2. **Short answers** (5 pts each)

(2.1) What is the height L of a full tree with N nodes ? Prove it.

(2.2) Given the graph below, draw its linked-list representation (store the vertices in alphabetical order). Then, apply Breadth First Search (BFS) to find if there is a path from A to D (to get credit, you need to show all the steps clearly).



(2.3) Explain the following terms:

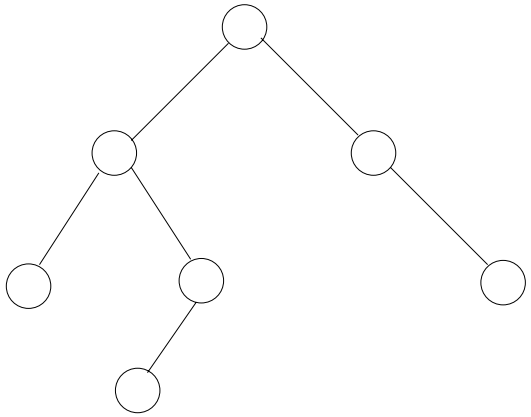
activation record:

run-time stack:

header and trailer nodes:

dynamic binding:

(2.4) Label the following binary tree with numbers from the set {6,22,9,14,13,1,8} so that it is a legal binary search tree (you can choose the numbers in this set in any order).



Show how the tree above would look like after each of the following operations: (i) delete 13 (ii) insert 34 (use the original tree)

(2.5) Compare recursion with iteration in terms of (i) time, (ii) memory, and (iii) efficiency.

(2.6) Trace the function below and describe what it does.

```
template<class ItemType>
int Mystery(TreeType<ItemType> *tree, int &n)
{
 if(tree != NULL) {
   n++;
   Mystery(tree->left, n);
   Mystery(tree->right, n);
 }
}
```

(4) (a) **[10 pts]** Define a new class *LookAheadStack* that is derived from class *StackType* (its definition is shown below). A look-ahead stack differs from the standard stack only in the push operation. An item is added to the stack by the push method only if it is different from the top stack element.

```
struct NodeType {
  int info;
  NodeType* next;
};

class StackType {
 public:
    StackType();
    ~StackType();
    void MakeEmpty();
    bool IsEmpty() const;
    bool IsFull() const;
    void Push(int);
    void Pop(int&);
 private:
    NodeType* topPtr;
};
```

(b) **[10 pts]** Implement the new push function and the derived class's constructor.

(c) **[5 pts]** Which functions and from which class should be declared as *virtual*?

(5) **[15 pts]** Write a **client** boolean function that determines if a binary search tree and an unsorted list contain exactly the same elements. Analyze its running time performance using big-O. The specifications of the binary search tree and unsorted linked-list are given in the next page.

```
template<class ItemType>
class TreeType {
  public:
    TreeType();
    ~TreeType();
    TreeType(const TreeType<ItemType>&);
    void operator=(const TreeType<ItemType>&);
    void MakeEmpty();
    bool IsEmpty() const;
    bool IsFull() const;
    int NumberOfNodes() const;
    void RetrieveItem(ItemType&, bool& found);
    void InsertItem(ItemType);
    void DeleteItem(ItemType);
    void ResetTree(OrderType);
    void GetNextItem(ItemType&, OrderType, bool&);
    void PrintTree(ofstream&) const;
  private:
    TreeNode<ItemType>* root;
};

template<class ItemType>
class UnsortedType {
  public:
    UnsortedType();
    ~UnsortedType();
    void MakeEmpty();
    bool IsFull() const;
    int LengthIs() const;
    void RetrieveItem(ItemType&, bool&);
    void InsertItem(ItemType);
    void DeleteItem(ItemType);
    void ResetList();
    bool IsLastItem() const;
    void GetNextItem(ItemType&);
  private:
    int length;
    NodeType<ItemType>* listData;
    NodeType<ItemType>* currentPos;
};

template<class ItemType>
class NodeType {
  ItemType info;
  NodeType *next;
};
```