C++ Week 6 Trees

Introduction

During this week's practical we will develop our own tree data structure.

Exercise 1

The Binary Tree will store Node objects that contain a single value of type int. Create a new C++ Console Project and add the Node class below to it.

```
// .h file
class Node {
public:
	Node *leftChild;
	Node *rightChild;
	int data;
	Node(int data);
	void Display();
};


// .cpp file
Node::Node(int data) {
		this->data = data;
		leftChild = 0;
		rightChild = 0;
}

void Node::Display() {
		cout << this->data << endl;
}
```

C++ Week 6 Trees

Exercise 2

Now create a Tree class and add the following member variables and functions.

```
// h file
class Tree {
public:
    Node *root;
        Tree();
    Node *Find(int key);
    void DisplayInOrder(Node *localRoot);
};

// cpp File
Tree::Tree(){
        root = 0;
}

Node *Tree::Find(int key) {
        Node * current = root;
        while (current->data != key) {
            if (key < current->data)
                current = current->leftChild;
            else
                current = current->rightChild;
            if (current == 0)
                return 0;
        }
        return current;
}

void Tree::DisplayInOrder(Node *localRoot) {
        if (localRoot != 0) {
            DisplayInOrder(localRoot->leftChild);
                                localRoot->Display();
            DisplayInOrder(localRoot->rightChild);
        }
}
```

C++ Week 6 Trees

Exercise 3

Add a new public member function to Tree named Insert, that takes a single argument (the data to be stored within a Node object). See below for the pseudo code for this method.

```
public void Insert(int data)
        Define a pointer newNode and assign it a new Node object passing data to
        constructor
        If root is 0
                Set root to newNode
        Otherwise
                Define current as a pointer to a Node
                Set current to root
                Define parent as a pointer to a Node
                While(true)
                        Set parent to current
                        If data < current->Data
                                Set current to current->LeftChild
                                If current is 0
                                        Set Parent->LeftChild to newNode
                                        Return
                                End if
                        Otherwise
                                Set current to current->RightChild
                                If current is 0
                                        Parent->RightChild = newNode
                                        Return
                                End if
                        End if
                End loop
        End if
End Function
```

C++ Week 6 Trees

Exercise 4

Within main create an instance of the Tree and insert 6 integers between 1 and 100 in a random order. Now invoke the DisplayInOrder method passing it the trees root.

The Tree class had limited applications, as the Nodes that it managed only stored a single integer value. If we want to use the tree to manage a large number of Players playing within an online game, we would need to create a more complex Node that contained information about the player. The Player class below stores the current level a player is playing on and the number of kills they have achieved since they started playing the game. Add the class to your project.

```
// .h file
class Player {
public:
    Player *leftChild;
    Player *rightChild;
    int kills;
    int level;
    Player(int level, int kills);
    void Display();
};

// .cpp file

Player::Player(int level, int kills) {
            this->kills = kills;
            this->level = level;
            leftChild = 0;
            rightChild = 0;
}

void Player::Display() {
          cout << "Level : " << this->level << " Kills : " << this->kills << endl;
}
```

The design of the Player class raises an issue with how we store it within the tree. Our previous example determined where to save the Node based on the int value stored with it. Since our player class has two values (level & kills), we must decide which one is to be used. In fact we will be using both, the position within the tree will be based on a number that is calculated using the following formulae

Factorial(level) * kills

C++ Week 6 Trees

Rather than creating a new data member within player and updating each time the kills or level change, we shall write a new function that will replace the **<** operator call within the tree's insert member function. Unfortunately it will not be possible to overload the < operator as both operands are pointers.

Note: This is an academic exercise that ignores the fact that two players may have the same integer value!

Exercise 5

Add the factorial function described in Week 1's practical to your player class. Note: This must be converted to a member function.

```
int Factorial(int n){
        if (n == 1)
                return 1;
        return Factorial(n-1) * n;
}
```

Now add a new function called LessThan to the Player class

```
bool Player::LessThan(Player *p1,Player *p2){
        int scoreA =
        int scoreB =
        if (scoreA < scoreB)
            return true;
        else
            return false;
}
```

Complete it by assigning the correct value to scoreA and scoreB based on the formulae Factorial(level) * kills

C++ Week 6 Trees

Exercise 6

Create a new class named PlayerTree and copy all the members from Tree into it. Now modify it so that it manages Player objects rather than Node objects.
Hint: Where you were comparing the data values before using the less than operator you must now use the LessThan member function.

Finally modify the code within program to create a PlayerTree objects and populate it with Player objects, before displaying its contents in order.