

## SG220 3D Game Engine Architecture

### KGA#2 - Sample Code Handout

Instructors/Students: This is an example of what the coded deliverable should look like.

```
//=====//
// Program name: Binary Tree creation and traversals //
// //
// Program Author: Author //
// //
// Program Description: This program creates a binary tree and performs a inorder, preorder, and //
// //
// // postorder traversals. //
// //
// Program Creation: //
// Date: May 13, 2006 //
// Time: 10:42 AM //
// //
// Note: Visual C++ Console Application. //
//=====//

//=====//
// Include files: //
// //
// iostream: This include files contains objects that control reading and writing to the standard streams. //
// //
// stdafx: This file includes files and for project-specific include files that are used frequently. //
// //
// fstream: This include file is an iostream derivative. It is for file input and output. //
//=====//

#include <iostream>
#include "stdafx.h"
#include <fstream>

// This directive allows the names in the namespace std to be used without explicit qualification.

using namespace std;

//===== Program data =====//

char buffer; // used in a cin command to keep the console active.

struct node { // Structure to simulate a node in the binary tree.
    char data; // Node data.
    struct node* left; // Node pointer to the left child.
    struct node* right; // Node pointer to the right child.
};

//=====//
```

```

//===== Program functions =====//

//=====//
// Function name: NewNode. //
// //
// Function description: This function creates a new binary tree node. //
// //
// Function arguments: int data - this integer contains the node data //
// //
// Function return: This function returns a pointer to the newly created node. //
//=====//

struct node* NewNode(int data)
{
    struct node* node = new(struct node); // instantiate new node.
    node->data = data; // load node data.
    node->left = NULL; // initialize left child pointer.
    node->right = NULL; // initialize right child pointer.

    return(node); // return pointer
                  // to the newly created node
}

//=====//
// Function name: Insert. //
// //
// Function description: This function insert a node into a binary tree. //
// //
// Function arguments: struct node* node - pointer to the parent node //
// int data - child node data (left or right) //
// //
// Function return: This function returns a pointer to parent node. //
//=====//

struct node* insert(struct node* node, int data)
{
    if (node == NULL) // if binary tree is empty - create new node.
        return(NewNode(data));
    else // create left child.
    {
        if (data <= node->data) node->left = insert(node->left, data);
        // create right child.
        else
            node->right = insert(node->right, data);
    }
    return(node);
}

```

```
//=====//  
// Function name: Search. //  
// // //  
// Function description: This function search for a particular node in the binary tree. //  
// // //  
// Function arguments: struct node* node - pointer to the top node. //  
// int data - data to be searched //  
// // //  
// Function return: This function returns a pointer to parent node. //  
//=====//
```

```
struct node* search(struct node* node, int data)  
{  
    if (node->data == data) // found match and it is the top node.  
        return (node);  
    else  
    {  
        if (node->data < data)  
            search(node->right, data); // search the right branch.  
        else  
            search(node->left, data); // search the left branch.  
    }  
}
```

```
//=====//  
// Function name: InOrder. //  
// // //  
// Function description: This function performs InOrder traversal in the binary tree. //  
// // //  
// Function arguments: struct node* node - pointer to the top node of the binary tree. //  
// // //  
// Function return: void - none. //  
//=====//
```

```
void InOrder (struct node* node)  
{  
    if (node)  
    {  
        InOrder (node->left); // traverse(visit) left branch.  
        cout<<node->data; // traverse(visit) node.  
        cout<<" ";  
        InOrder (node->right); // traverse(visit) right branch.  
    }  
}
```

```
//=====//  
// Function name: InOrder. //  
// //  
// Function description: This function performs PreOrder traversal in the binary tree. //  
// //  
// Function arguments: struct node* node - pointer to the top node //  
// //  
// Function return: void - none. //  
//=====//
```

```
void PreOrder(struct node* node)  
{  
    if (node)  
    {  
        cout<<node->data; // traverse(visit) node.  
        cout<<" ";  
        PreOrder (node->left); // traverse(visit) left branch.  
        PreOrder (node->right); // traverse(visit) righth branch.  
    }  
}
```

```
//=====//  
// Function name: InOrder. //  
// //  
// Function description: This function performs PostOrder traversal in the binary tree. //  
// //  
// Function arguments: struct node* node - pointer to the top node //  
// //  
// Function return: void - none. //  
//=====//
```

```
void PostOrder(struct node* node)  
{  
    if (node)  
    {  
        PostOrder (node->left); // traverse(visit) left branch.  
        PostOrder (node->right); // traverse(visit) right branch.  
        cout<<node->data; // traverse(visit) node.  
        cout<<" ";  
    }  
}
```

```
//=====//  
// Function name: PrintTree. //  
// //  
// Function description: This function "display" the binary tree. //  
// //  
// Function arguments: struct node* node - pointer to the top node //  
// //  
// Function return: void - none. //  
// //  
//=====//
```

```
void printTree(struct node* node)  
{  
    cout<<node->data<<endl;  
  
    if (node->left)  
    {  
        // Display left child.  
        cout<<"Left child of node ";  
        cout<<node->data;  
        cout<<" is node ";  
  
        printTree(node->left);  
    }  
  
    if (node->right)  
    {  
        // Display right child.  
        cout<<"Right child of node ";  
        cout<<node->data;  
        cout<<" is node ";  
  
        printTree(node->right);  
    }  
}
```

```
//=====//
// Program main.                                     //
//=====//

void main()
{
    node *myNode;
    node *searchMyNode;

    myNode = NewNode('I');           // Create a new node containing 'I'
    insert (myNode, 'L');           // Insert node 'L'

    searchMyNode = search(myNode, 'L'); // Search for node 'L'
    insert (searchMyNode, 'K');     // Insert node 'K'
    insert (searchMyNode, 'M');     // Insert node 'M'

    searchMyNode = search(myNode, 'M'); // Search for node 'M'
    insert (searchMyNode, 'P');     // Insert node 'P'
    insert (myNode, 'D');           // Insert node 'D'

    searchMyNode = search(myNode, 'D'); // Search for node 'D'
    insert (searchMyNode, 'A');     // Insert node 'A'
    insert (searchMyNode, 'F');     // Insert node 'F'

    searchMyNode = search(searchMyNode, 'F'); // Search for node 'F'
    insert (searchMyNode, 'H');     // Insert node 'H'

    cout<<"Binary Tree:"<<endl<<endl;

    printTree(myNode);             // Display the binary tree.

    cout<<endl<<endl<<"In-Order Traversal-> ";

    InOrder(myNode);              // InOrder traversal.

    cout<<endl<<endl<<"Pre-Order Traversal-> ";

    PreOrder(myNode);             // PreOrder traversal.

    cout<<endl<<endl<<"Post-Order Traversal-> ";

    PostOrder(myNode);           // PostOrder traversal.

    cin >> buffer;               // Keep the console active(open).
}
```