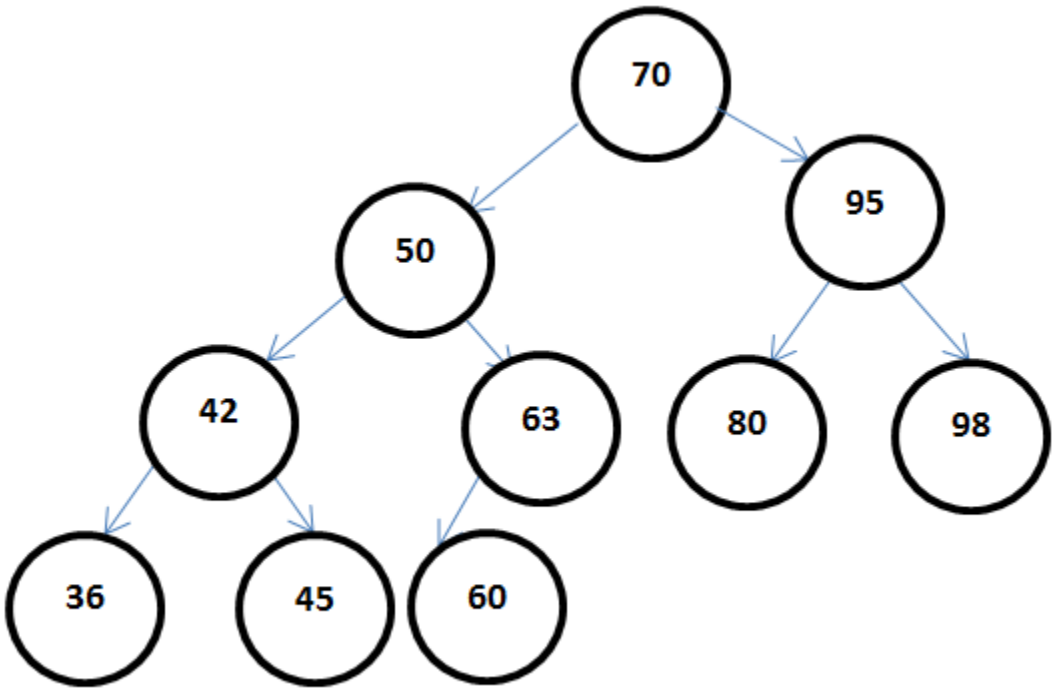
Assignment 2

Problem statement:

Write a program that would
1. Create a binary tree with 10 nodes
2. Traverse the tree Breadth first.
3. Your tree node should NOT have a pointer to the parent.
   Only the pointers to the left child and the right child.

Submit the code and the screen shot of your program run.

Solution:
Considering the following tree as the example:



The Breadth first search works as follows :

1. Initially pointer points to the root node and enqueues the root node and adjacent node.
2. Then Dequeue operation is done.

3. The pointer is moved to the next element in the queue and the traversal continues in the similar steps as mentioned in 1 and 2.

Traversal:
1st iteration :
Enqueue : 70,50,95
Dequeue and display: 70
2nd iteration :
Enqueue: 50,95,42,63
Dequeue and display: 50
3rd iteration :
Enqueue: 95,42,63,80,98
Dequeue and display: 95
4th iteration:
Enqueue: 42,63,80,98,36,45
Dequeue and display: 42
5th iteration:
Enqueue: 63,80,98,36,45,60
Dequeue and display: 63
6th iteration:
Dequeue and display: 80
7th iteration :
Dequeue and display: 98
8th iteration :
Dequeue and display: 36
9th iteration:
Dequeue and display: 45
10th iteration:
Dequeue and display: 60

Expected output: 70->50->95->42->63->80->98->36->45->60

Code :

```
/*
 * tree.cpp
 *
 *  Created on: Sep 21, 2014
 *      Author: apoor_000
 */
#include <cstdlib>
#include <iostream>

using namespace std;

struct node { //Struct Node definition for Binary Tree
	int value;
	node* left;
	node* right;
};

struct node1 //Struct node definition for queue
{
    int data; //data item
    node1* next; //pointer to next link
};

class queue //a list of links
{
private:
    node1* head; //pointer to first link
    node1* tail; //pointer to last link
public:

    queue() //no-argument constructor
    {
        head = NULL;
        tail = NULL;
    } //no first link
    void enqueue(int d); //add data item (one link)
    void display(); //display all link
    int dequeue();
    int front();
};

class Bintree{
private:
	node* rootNode;
public:
	Bintree()    //no-argument constructor
	{
		rootNode = NULL;
	}
	node *insert(node *s, int key);
	void BFS(node *s);
	void preTraverse(node *s);
```

```
};

void queue::enqueue(int d) //add data item
{
       cout << "Enqueue function called" << endl;
       cout << "Enqueue :" << d << endl;
    node1 *newNode = new node1;
    newNode->data = d;
    newNode->next = NULL;

       if ( head == NULL && tail == NULL ) {
              head = tail = newNode;
              return;
       }
    tail->next = newNode;
       tail = newNode;
}

int queue::dequeue() {
       cout << "Dequeue function called" << endl;
    node1 *tmp = head;
    int d;
    if(head == NULL) //If the list is already empty
    {
              cout << "Error : Queue is Empty " << endl;
              return(-1);
       }
       if(head == tail) //If the list gets empty after multiple dequeues
       {
              d = head->data;
              head = tail = NULL;
       } else {
              d = head->data;
              head = head->next;
       }
       delete tmp;
    return(d);
}

node *Bintree::insert(node *rootNode,int key) {

       if(rootNode==NULL) {
              rootNode=new node;
              rootNode->value=key;
              rootNode->left=NULL;
              rootNode->right=NULL;
       } else {
              if (key > rootNode->value) {
                     rootNode->right = insert(rootNode->right,key);
              } else {
                     rootNode->left = insert(rootNode->left,key);
              }
       }
       return (rootNode);
}
```

```
void Bintree::preTraverse(node *rootNode){
	if (rootNode != NULL ){
		cout << rootNode->value << "->";
		preTraverse(rootNode->left);
		preTraverse(rootNode->right);
	}
}

void Bintree::BFS(node *rootNode){
	queue q;
	node *rtn;
	int ret, frnt;
	q.enqueue(rootNode->value);
	q.enqueue(rootNode->left->value);
	q.enqueue(rootNode->right->value);
	ret = q.dequeue();

	if (ret > 0){
		rtn = (node *) ret;
		cout << ret <<endl;
	}
}

int main() {
	Bintree bt;
	node *start = NULL;
	start = bt.insert(start,70);
	start = bt.insert(start,50);
	start = bt.insert(start,95);
	start = bt.insert(start,42);
	start = bt.insert(start,63);
	start = bt.insert(start,80);
	start = bt.insert(start,98);
	start = bt.insert(start,36);
	start = bt.insert(start,45);
	start = bt.insert(start,60);
	bt.preTraverse(start);
	bt.BFS(start);
}
```

Output :

1. Was able to successfully construct a binary tree and insert elements into it.
2. Was able to verify the elements by printing them in PreOrder traversal search.
3. Partially done with Breadthfirst search wherein, was able to perform enqueue and Dequeue operations on the given queue.

Screen shot is as below: