ASSIGNMENT NO:6 **DATE:05/09/2015**

PROGRAM TITLE:Write a C++ Program to find the max width and mirror image of a Binary Search Tree in a non-recursive way. Create and print the B.S.T also in non-recursive way. Make the code generic.

THEORY:A Binary Search Tree is a Binary Tree which hold all the normal properties of a Binary Tree along with a extra property which is that the elements in the left subtree should always be less than that of the root node and the elements in the right subtree should always be less than that of the root node. This property holds true for all the nodes of the tree.

PROGRAM ALGORITHM:

```
//root is the root of the B.S.T,front and rear are the pointers of the
queue,width is set as 1 in the beginning
Algo_findwidth(root)
{
      initialise a queue with the root in the data part of the 1st cell;
      insert a NULL node (the separator) into the queue;
      while(front not equal to rear)
      {
            if(data(front)=NULL)
            {
                  initialise w as 0;
                  calculate the number of cells between front and rear and
store it in w;
                  if(w>width)
                        set width as w;
                  insert a NULL node (the separator) into the queue;
            }
            else
            {
                  insert the left child of the front node if it is not
empty;
                  insert the right child of the front node if it is not
empty;
            }
            delete an element from the queue;
      }
}

Algo_mirror()
{
      Push the root onto the stack;
      while(stack is not empty)
      {
            curr=pop from stack;//curr is a temporary node
            swap the children of curr;
            push the children onto the stack;
      }
}
```

PROGRAM CODE:

```
#include <iostream>
#include <cstdlib>
using namespace std;
```

```
template <class T>
struct node
{
      T data;
      struct node<T> *left;
      struct node<T> *right;
};

/*************************Converting into a
Mirror*************************/
template <class T>
struct snode
{
      struct node<T> *data;
      struct snode<T> *next;
};
template <class T>
class Tree
{
      struct node<T>* root;
      public:
            Tree()
            {
                  root=NULL;
            }
            void insert();
            void indisplay();
            void mirror();
            void findwidth();
            void menu();
};
template <class T>
class Stack
{
      struct snode<T>* top;
      public:
            Stack();
            struct snode<T>* allocnode(struct node<T>* item);
            void push(struct node<T> *inp);
            struct node<T>* pop();
            int empty()
            {
                  if(top==NULL)
                        return 1;
                  return 0;
            }
};
template <class T>
Stack<T>::Stack()
{
      top=NULL;
}
template <class T>
struct snode<T>* Stack<T>::allocnode(struct node<T>* item)
{
      struct snode<T>* temp = (struct snode<T>*)malloc(sizeof(struct
snode<T>));
      temp->data=item;
```

```
      temp->next=NULL;
      return temp;
}
template <class T>
void Stack<T>::push(struct node<T>* item)
{
      struct snode<T> *inter=allocnode(item);
            if(inter==NULL)
      {
                  cout<<"\n\tMemory couldnt be allocated from the heap.
Overflow occurs.";
      }
      if(top==NULL)
      {
            top=inter;
      }
      else
      {
            inter->next=top;
            top=inter;
      }
}
template <class T>
struct node<T>* Stack<T>::pop()
{
      struct snode<T>* p = top;
      struct node<T>* item;
      if(top==NULL)
      {
            cout<<"\n\tUnderflow Occurs.";
            return NULL;
      }
      else
      {
            top=top->next;
            item=p->data;
            delete p;
            return item;
      }
}

/* Mirror the tree using stack*/
template <class T>
void Tree<T>::mirror()
{
      if(root==NULL)
            return;
      Stack<T> s;
      s.push(root);
      while(!s.empty())
      {
            struct node<T> *curr=s.pop();

            //Swap the children
            struct node<T> *temp=curr->right;
            curr->right=curr->left;
            curr->left=temp;

            //Push the children on the stack
```

```
            if(curr->right)
                  s.push(curr->right);
            if(curr->left)
                  s.push(curr->left);
      }
}

/**********************Finding the width of the
tree************************/
template <class T>
struct qnode
{
      struct node<T> *data;
      struct qnode<T> *next;
};
template <class T>
int queuein(struct qnode<T> **rear,struct node<T> *inp)
{
      if(inp!=NULL)
      {
            struct qnode<T> *temp=(struct qnode<T> *)malloc(sizeof(struct
qnode<T>));
            temp->data=inp;
            temp->next=NULL;
            (*rear)->next=temp;
            *rear=temp;
      }
      return 0;
}
template <class T>
int queuedel(struct qnode<T> **front)
{
      struct qnode<T> *p=*front;
      (*front)=(*front)->next;
      free(p);
      p=NULL;
      return 0;
}
template <class T>
void Tree<T>::findwidth()
{
      if(root==NULL)
      {
            cout<<"\n\tThe width is-0"<<endl;
            return;
      }
      static int width=1;
      struct qnode<T> *front=NULL;
      struct qnode<T> *rear=(struct qnode<T> *)malloc(sizeof(struct
qnode<T>));
      rear->data=root;
      rear->next=NULL;
      front=rear;
      rear=(struct qnode<T> *)malloc(sizeof(struct qnode<T>));
      rear->data=NULL;
      rear->next=NULL;
      front->next=rear;
      while(front!=rear)
      {
```

```
            if(front->data==NULL)
            {
                  int w=0;
                  struct qnode<T> *tmp=front;
                  while(tmp!=rear)
                  {
                        w++;
                        tmp=tmp->next;
                  }
                  if(w>width)
                        width=w;
                  tmp=(struct qnode<T> *)malloc(sizeof(struct qnode<T>));
                  tmp->data=NULL;
                  tmp->next=NULL;
                  rear->next=tmp;
                  rear=tmp;
            }
            else
            {
                  queuein(&rear,(front->data)->left);
                  queuein(&rear,(front->data)->right);
            }
            queuedel(&front);
      }
      cout<<"\n\tThe width is-"<<width<<endl;
}

/*********************************Creating the
Tree*************************/
template <class T>
int check(struct node<T> *root)
{
      if(root==NULL)
            return 0;
      return 1;
}
template <class T>
struct node<T>* parent(struct node<T> *root,int item)
{
      struct node<T> *pos=root;
      while((pos!=NULL)&&(pos->data!=item))
      {
            if(pos->data>item)
            {
                  if(check(pos->left))
                        pos=pos->left;
                  else
                     break;

            }
            if(pos->data<item)
            {
                  if(check(pos->right))
                        pos=pos->right;
                  else
                     break;
            }

      }
```

```
      return pos;
}
template <class T>
void Tree<T>::insert()
{
      T item;
      cout<<"\n\tEnter data::";
      cin>>item;
      struct node<T> *temp=(struct node<T> *)malloc(sizeof(struct
node<T>));
      temp->data=item;
      temp->left=NULL;
      temp->right=NULL;
      if(root==NULL)
      {
            root=temp;
      }
      else
      {
            struct node<T> *par=parent(root,item);
            if((par)->data>item)
            {
                  (par)->left=temp;
            }
            else if((par)->data<item)
            {
                  (par)->right=temp;
            }
            else
            {
                  cout<<"\n\tItem already present in tree.";
            }
      }
}

/**********************************Displaying the
Tree**************************/
template <class T>
void Tree<T>::indisplay()
{
      Stack<T> S;
      struct node<T> *temp=root;
      cout<<"\n\tInorder Display of Tree::\n\t";
      while(1)
      {
            while(temp!=NULL)
            {
                  S.push(temp);
                  temp=temp->left;
            }
            if(S.empty())
                  break;
            temp=S.pop();
            cout<<temp->data<<"\t";
            temp=temp->right;
      }
      cout<<endl;
}
template <class T>
```

```
void Tree<T>::menu()
{
      int ch=1;
      while (ch!=0)
      {
            cout<<"\n\tMENU::\n\t1.Insert\n\t2.Inorder
Display\n\t3.Convert Tree into Mirror\n\t4.Find the
Width\n\t0.Exit\n\tEnter Choice::";
            cin>>ch;
            switch(ch)
            {
            case 1:    insert();
                  break;
            case 2:    indisplay();
                  break;
            case 3: mirror();
                  break;
            case 4: findwidth();
                  break;
            case 0: cout<<"\n\tProgram Terminated."<<endl;
                  break;
            default:cout<<"\n\tInvalid Choice.";
            }
      }
}
/**********************************The Main
Function*****************************/
int main()
{
      int ch=1;
      cout<<"\n\tChoose the datatype that you want to work
with::\n\t1.Integer\n\t2.Float\n\t3.Character\n\tEnter Choice::";
      cin>>ch;
      switch(ch)
      {
            case 1:{Tree<int> t;
                  t.menu();}
                  break;
            case 2:    {Tree<float> t;
                  t.menu();}
                  break;
            case 3:    {Tree<char> t;
                  t.menu();}
                  break;
            default:cout<<"\n\tInvalid Choice."<<endl;
      }
      return 0;
}
```

OUTPUT:

Set 1:

```
Choose the datatype that you want to work with::
1.Integer
2.Float
3.Character
Enter Choice::1

MENU::
```

```
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::50

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
50

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::25

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
25   50

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::35

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2
```

```
Inorder Display of Tree::
25    35    50

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::40

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::30

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
25    30    35    40    50

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::15

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::20

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
```

```
0.Exit
Enter Choice::2

Inorder Display of Tree::
15    20    25    30    35    40    50

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::4

The width is-3

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::3

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
50    40    35    30    25    20    15

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::0

Program Terminated.
```

Set 2:

```
Choose the datatype that you want to work with::
1.Integer
2.Float
3.Character
Enter Choice::2

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1
```

```
Enter data::10.56

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
10.56

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::15.69

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::0.2569

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::14.12

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::12.14

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
```

```
0.Exit
Enter Choice::2

Inorder Display of Tree::
0.2569     10.56      12.14      14.12      15.69

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::3

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
15.69  14.12     12.14      10.56      0.2569

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::4

The width is-2

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::0

Program Terminated.
```

Set 3:

```
Choose the datatype that you want to work with::
1.Integer
2.Float
3.Character
Enter Choice::3

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1
```

```
Enter data::S

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::O

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
O    S

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::U

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
O    S    U

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::M

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
```

```
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
M    O    S    U

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::I

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
I    M    O    S    U

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::K

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
I    K    M    O    S    U

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::D
```

```
MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
D    I    K    M    O    S    U

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::E

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
D    E    I    K    M    O    S    U

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::d

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::1

Enter data::e

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2
```

```
Inorder Display of Tree::
D    E    I    K    M    O    S    U    d    e

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::4

The width is-2

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::3

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::2

Inorder Display of Tree::
e    d    U    S    O    M    K    I    E    D

MENU::
1.Insert
2.Inorder Display
3.Convert Tree into Mirror
4.Find the Width
0.Exit
Enter Choice::0

Program Terminated.
```

DISCUSSION:

1. We use templates here to make the code generic.
2. While working with char, the data is arranged according to their ASCII codes due to the lack of a specialised method to alphabetically sort them.
3. Both the algorithms for mirror and findwidth traverse all the nodes of the tree, therefore they are both of the O(n).
4. For finding the width, we use a queue where we push all the nodes in the same level into the queue and count and compare them to find the max width.
5. For the mirror and the inorder traversal, we push each node into the stack and then work by popping each individual from the stack until it is empty.
6. Care must be taken to ensure that the scopes of the objects (in the main menu) should not overlap.