B190839CS

ROSE S JOSE

```
//B190839CS ROSE S JOSE

#include<iostream>
#include<cmath>
struct node{
    int *key;
    node **child;
    bool leaf;
    int n;
    node(int deg)
    {
        key = new int[deg];
        leaf = true;
        child = new node*[deg+1];
        for(int i=0;i<deg;i++)
            child[i] = NULL;
    }

    node* findNode(node *temp, int data, int t, node *root);
    void splitChild(node *temp, int t);
    void traverse();
    bool search(int k);
};

class btree{
private:
    node* root;
    int t;
```

```
public:
        btree(int degree)
        {
                root = NULL;
                t = degree;
        }
        void insert(int data)
        {
                if(root == NULL)
                {
                        root = new node(t);
                        root->leaf = true;
                        root->key[0] = data;
                        root->n = 1;
                }
                else
                {
                        root = root->findNode(root, data, t, root);
                }
        }

        void print()
        {
                if(root!= NULL)
                        root->traverse();
        }
        bool search(int k)
  {
        if(root == NULL)
                return false;
        else
        return root->search(k);
   }
};

node* node::findNode(node *temp, int data, int t, node *root)
{
    if(leaf)
    {
        int i=t;
        //finding the first index from right having data
        while(!key[i-1])
            i--;
        //inserting new key at the correct place
        while(key[i-1] > data && i != 0)
        {
            key[i] = key[i-1];
            i--;
        }
        key[i] = data;
        n+=1;
    }
    else
    {
        int i=0;
        //finding the child where the key should be inserted
        while(i<n && data > key[i])
        {
            i++;
        }
        root = child[i]->findNode(this, data, t, root); //????
    }
    if(n == t)
    {
        if(this == root)
        {
            //creating new node and making the overflowing node its child
            node *s = new node(t);
            s->leaf = false;
            s->child[0] = this;
            s->splitChild(this, t);
            return s;
        }
        else
        {
            temp->splitChild(this, t);
        }
    }
    return root;
}
void node::splitChild(node *fullNode, int t)
{
    node *rightNode = new node(t);
    int i_right=0, move_up = (t-1)/2;
    int n_fullNode = fullNode->n;
    int carry = fullNode->key[move_up];
    for(int i=move_up+1;i < n_fullNode; i++)
    {
        rightNode->key[i_right] = fullNode->key[i];
        i_right++;
        fullNode->n--;
        rightNode->n++;
    }
```

```
//If the node has children
int child_split = move_up+1, i_child = 0;
if(fullNode->leaf == false)
{
    for(int i=child_split; i<= t; i++)
    {
        rightNode->child[i_child]=fullNode->child[i];
        i_child++;
    }
    rightNode->leaf = fullNode->leaf;
}

//insert the new right node right after the fullNode
int j=t-1;
while(child[j] != fullNode)
{
    child[j+1]= child[j];
    j--;
}
child[j+1]=rightNode;

//insert carry to the parent
j = t-1;
while(!key[j-1] && j!=0)
{
        j--;
}
while(key[j-1] > carry && j!=0)
{
    key[j] = key[j-1];
```

```
                        j--;
                }
                key[j] = carry;
                n+=1;
                fullNode->n--;

}
void node::traverse()
{
                int i;
    for (i = 0; i < n; i++)
    {
        // If this is not leaf, then before printing key[i],
        // traverse the subtree rooted with child C[i].
        if (leaf == false)
            child[i]->traverse();
        std::cout << " " << key[i];
    }

    // Print the subtree rooted with last child
    if (leaf == false)
        child[i]->traverse();
}
bool node::search(int k)
{
    // Finding position of k
    int i = 0;
    while (i < n && k > key[i])
        i++;

    // If the found key is equal to k, return this node
```

```
  if (key[i] == k)
    return true;

  // If node is leaf and key not found
  if (leaf == true)
    return false;

  // Go to the correct child
  return child[i]->search(k);
}
int main()
{
        int t, k;
        char c;
        bool i;
        std::cout<<"Order: ";
        std::cin>>t;
        btree T(t);
        do
        {
                std::cin>>c;
                switch(c)
                {
                        case 'i':std::cin>>k;
                                        T.insert(k);
                                        break;
                        case 's':std::cin>>k;
                                        i = T.search(k);
                                        if(i==1)
                                                std::cout<<"TRUE\n";
                                        else
                        std::cout<<"FALSE\n";
                    break;
            case 'p':T.print();
                    break;
            case 'e':break;
        }
        if(c=='e')
            break;
    }while(1);
    return 0;
}
```