```
#include<iostream>
#include<limits.h>
#include<cstdint>
#include<cstring>
#include<math.h>
#include <vector>
using namespace std;

#define MAX 50


int children;
bool exist_flag;
int level;
struct Block{

    int key;
    Block *parent_head;
    int value[MAX];
    Block *child_block[MAX];

    Block()
    {
        key = 0;
        parent_head = NULL;
        for(int i=0; i<MAX; i++){
            value[i] = INT_MAX;
            child_block[i] = NULL;
        }
    }
};

Block *root_block = new Block();


void split_leaf_node(Block *curr_block)
{
    int temp, i, j;

    if(children%2)
        temp = (children+1)/2;
    else temp = children/2;
```

```
Block *right_block = new Block();

curr_block->key = temp;
right_block->key = children-temp;
right_block->parent_head = curr_block->parent_head;

for(i=temp, j=0; i<children; i++, j++){
   right_block->value[j] = curr_block->value[i];
   curr_block->value[i] = INT_MAX;
}
int val = right_block->value[0];

if(curr_block->parent_head==NULL)
{

   Block *parent_head = new Block();
   parent_head->parent_head = NULL;
   parent_head->key=1;
   parent_head->value[0] = val;
   parent_head->child_block[0] = curr_block;
   parent_head->child_block[1] = right_block;
   curr_block->parent_head = right_block->parent_head = parent_head;
   root_block = parent_head;
   return;
}
else
{
   curr_block = curr_block->parent_head;

   Block *new_child = new Block();
   new_child = right_block;

   for(i=0; i<=curr_block->key; i++){
      if(val < curr_block->value[i]){
         swap(curr_block->value[i], val);
      }
   }

   curr_block->key++;

      for(i=0; i<curr_block->key; i++){
      if(new_child->value[0] < curr_block->child_block[i]->value[0]){
         swap(curr_block->child_block[i], new_child);
      }
```

```
        }
        curr_block->child_block[i] = new_child;
           for(i=0;curr_block->child_block[i]!=NULL;i++){
           curr_block->child_block[i]->parent_head = curr_block;
        }
    }

}


void split_non_leaf_node(Block *curr_block)
{
    int temp, i, j;
    temp = children/2;

    Block *right_block = new Block();
    curr_block->key = temp;

    right_block->key = children-temp-1;

    right_block->parent_head = curr_block->parent_head;


    for(i=temp, j=0; i<=children; i++, j++)
    {

        right_block->value[j] = curr_block->value[i];
        right_block->child_block[j] = curr_block->child_block[i];
        curr_block->value[i] = INT_MAX;
        if(i!=temp)
            curr_block->child_block[i] = NULL;
    }

    int val = right_block->value[0];
    memcpy(&right_block->value, &right_block->value[1], sizeof(int)*(right_block->key+1));
    memcpy(&right_block->child_block, &right_block->child_block[1],
sizeof(root_block)*(right_block->key+1));

    for(i=0;curr_block->child_block[i]!=NULL;i++)
    {
        curr_block->child_block[i]->parent_head = curr_block;
    }

    for(i=0;right_block->child_block[i]!=NULL;i++)
```

```
{
   right_block->child_block[i]->parent_head = right_block;
}


if(curr_block->parent_head==NULL)
{

   Block *parent_head = new Block();
   parent_head->parent_head = NULL;
   parent_head->key=1;
   parent_head->value[0] = val;
   parent_head->child_block[0] = curr_block;
   parent_head->child_block[1] = right_block;

   curr_block->parent_head = right_block->parent_head = parent_head;

   root_block = parent_head;
   return;
}
else
{
   curr_block = curr_block->parent_head;
   Block *new_child = new Block();
   new_child = right_block;
   for(i=0; i<=curr_block->key; i++)
   {
      if(val < curr_block->value[i])
      {
         swap(curr_block->value[i], val);
      }
   }

   curr_block->key++;
   for(i=0; i<curr_block->key; i++)
   {
      if(new_child->value[0] < curr_block->child_block[i]->value[0]){
         swap(curr_block->child_block[i], new_child);
      }
   }
   curr_block->child_block[i] = new_child;

    for(i=0;curr_block->child_block[i]!=NULL;i++)
    {
```

```
            curr_block->child_block[i]->parent_head = curr_block;
        }
    }

}

void INSERT(Block *curr_block, int val){

    for(int i=0; i<=curr_block->key; i++)
    {
        if(val < curr_block->value[i] && curr_block->child_block[i]!=NULL)
        {
            INSERT(curr_block->child_block[i], val);
            if(curr_block->key==children)
                split_non_leaf_node(curr_block);
            return;
        }
        else if(val < curr_block->value[i] && curr_block->child_block[i]==NULL)
        {
            swap(curr_block->value[i], val);

            if(i==curr_block->key)
            {
                    curr_block->key++;
                    break;
            }
        }
    }

    if(curr_block->key==children)
            split_leaf_node(curr_block);

}


void rearrange(Block *left_block, Block *right_block, bool isLeaf, int posOfLeftBlock, int
curr_block_flag)
{

    int right_sib_first = right_block->value[0];


    if(curr_block_flag==0)
    {
```

```
      if(!isLeaf)
      {

         left_block->value[left_block->key] = left_block->parent_head->value[posOfLeftBlock];
         left_block->child_block[left_block->key+1] = right_block->child_block[0];
         left_block->key++;
         left_block->parent_head->value[posOfLeftBlock] = right_block->value[0];
         memcpy(&right_block->value[0], &right_block->value[1],
sizeof(int)*(right_block->key+1));
         memcpy(&right_block->child_block[0], &right_block->child_block[1],
sizeof(root_block)*(right_block->key+1));
         right_block->key--;

      }
      else
      {

         left_block->value[left_block->key] = right_block->value[0];
         left_block->key++;

         memcpy(&right_block->value[0], &right_block->value[1],
sizeof(int)*(right_block->key+1));

         right_block->key--;
         left_block->parent_head->value[posOfLeftBlock] = right_block->value[0];
      }



   }
   else
   {

      if(!isLeaf)
      {

         memcpy(&right_block->value[1], &right_block->value[0],
sizeof(int)*(right_block->key+1));
         memcpy(&right_block->child_block[1], &right_block->child_block[0],
sizeof(root_block)*(right_block->key+1));

         right_block->value[0] = left_block->parent_head->value[posOfLeftBlock];

         right_block->child_block[0] = left_block->child_block[left_block->key];
```

```
            right_block->key++;

            left_block->parent_head->value[posOfLeftBlock] = left_block->value[left_block->key-1];

            left_block->value[left_block->key-1] = INT_MAX;
            left_block->child_block[left_block->key] = NULL;
            left_block->key--;

        }
        else
        {

            memcpy(&right_block->value[1], &right_block->value[0],
sizeof(int)*(right_block->key+1));
            right_block->value[0] = left_block->value[left_block->key-1];
            right_block->key++;
            left_block->value[left_block->key-1] = INT_MAX;
            left_block->key--;
            left_block->parent_head->value[posOfLeftBlock] = right_block->value[0];
        }
    }
}

void merge(Block *left_block, Block *right_block, bool isLeaf, int posOfRightBlock){

    if(!isLeaf){

        left_block->value[left_block->key] = left_block->parent_head->value[posOfRightBlock-1];
        left_block->key++;
    }

    memcpy(&left_block->value[left_block->key], &right_block->value[0],
sizeof(int)*(right_block->key+1));
    memcpy(&left_block->child_block[left_block->key], &right_block->child_block[0],
sizeof(root_block)*(right_block->key+1));


    left_block->key += right_block->key;


    memcpy(&left_block->parent_head->value[posOfRightBlock-1],
&left_block->parent_head->value[posOfRightBlock],
sizeof(int)*(left_block->parent_head->key+1));
```

```
   memcpy(&left_block->parent_head->child_block[posOfRightBlock],
&left_block->parent_head->child_block[posOfRightBlock+1],
sizeof(root_block)*(left_block->parent_head->key+1));
   left_block->parent_head->key--;

    for(int i=0;left_block->child_block[i]!=NULL;i++){
      left_block->child_block[i]->parent_head = left_block;
   }



}

void DELETE(Block *curr_block, int val, int curBlockPosition)
{


   bool isLeaf;
   if(curr_block->child_block[0]==NULL)
      isLeaf = true;
   else
         isLeaf = false;

   int prev_sib_first = curr_block->value[0];



   for(int i=0;exist_flag==false &&  i<=curr_block->key; i++)
   {
      if(val < curr_block->value[i] && curr_block->child_block[i] != NULL)
      {
         DELETE(curr_block->child_block[i], val, i);

      }

      else if(val == curr_block->value[i] && curr_block->child_block[i] == NULL){

         memcpy(&curr_block->value[i], &curr_block->value[i+1],
sizeof(int)*(curr_block->key+1));

         curr_block->key--;
         exist_flag = true;
         break;
      }
```

```
    }

    if(curr_block->parent_head == NULL && curr_block->child_block[0] == NULL)
      return;



  if(curr_block->parent_head==NULL && curr_block->child_block[0] != NULL &&
curr_block->key == 0)
  {
     root_block = curr_block->child_block[0];
     root_block->parent_head = NULL;
     return;
  }

  if(isLeaf && curr_block->parent_head!=NULL)
  {

     if(curBlockPosition==0)
     {
        Block *right_block = new Block();
        right_block = curr_block->parent_head->child_block[1];

        if(right_block!=NULL && right_block->key > (children+1)/2){

              rearrange(curr_block, right_block, isLeaf, 0, 0);
        }
        else if (right_block!=NULL && curr_block->key+right_block->key < children){

              merge(curr_block, right_block, isLeaf, 1);
        }
     }

     else{


        Block *left_block = new Block();
        Block *right_block = new Block();


        left_block = curr_block->parent_head->child_block[curBlockPosition-1];

        right_block = curr_block->parent_head->child_block[curBlockPosition+1];
        if(left_block!=NULL && left_block->key > (children+1)/2)
```

```
        {
            rearrange(left_block, curr_block, isLeaf, curBlockPosition-1, 1);
        }
        else if(right_block!=NULL && right_block->key > (children+1)/2)
        {
           rearrange(curr_block, right_block, isLeaf, curBlockPosition, 0);
        }
        else if (left_block!=NULL && curr_block->key+left_block->key < children)
        {
           merge(left_block, curr_block, isLeaf, curBlockPosition);
        }
        else if (right_block!=NULL && curr_block->key+right_block->key < children)
        {
          merge(curr_block, right_block, isLeaf, curBlockPosition+1);
        }
    }
}
else if(!isLeaf && curr_block->parent_head!=NULL)
{

    if(curBlockPosition==0){
        Block *right_block = new Block();
        right_block = curr_block->parent_head->child_block[1];

        if( right_block!=NULL && right_block->key-1 >= ceil((children-1)/2))
        {
           rearrange(curr_block, right_block, isLeaf, 0, 0);
        }

        else if (right_block!=NULL && curr_block->key+right_block->key < children - 1)
        {
           merge(curr_block, right_block, isLeaf, 1);
        }
    }

    else{


        Block *left_block = new Block();
        Block *right_block = new Block();


        left_block = curr_block->parent_head->child_block[curBlockPosition-1];
```

```
            right_block = curr_block->parent_head->child_block[curBlockPosition+1];

            if( left_block!=NULL && left_block->key-1 >= ceil((children-1)/2))
            {
                rearrange(left_block, curr_block, isLeaf, curBlockPosition-1, 1);
            }
            else if(right_block!=NULL && right_block->key-1 >=  ceil((children-1)/2))
            {
               rearrange(curr_block, right_block, isLeaf, curBlockPosition, 0);
            }

            else if ( left_block!=NULL && curr_block->key+left_block->key < children-1)
            {
               merge(left_block, curr_block, isLeaf, curBlockPosition);
            }
            else if ( right_block!=NULL && curr_block->key+right_block->key < children-1)
            {
              merge(curr_block, right_block, isLeaf, curBlockPosition+1);
            }
         }

      }


      Block *tempBlock = new Block();
      tempBlock = curr_block->parent_head;
      while(tempBlock!=NULL){
            for(int i=0; i<tempBlock->key;i++)
            {
               if(tempBlock->value[i]==prev_sib_first)
               {
                  tempBlock->value[i] = curr_block->value[0];
                  break;
               }
         }
         tempBlock = tempBlock->parent_head;
      }

   }

   void PRINT_TREE(vector < Block* > Blocks){
      vector < Block* > newBlocks;
      for(int i=0; i<Blocks.size(); i++)
      {
```

```
        Block *curr_block = Blocks[i];

        cout <<"[|";
        int j;
        for(j=0; j<curr_block->key; j++)
        {
            cout << char(curr_block->value[j]) << "|";
            if(curr_block->child_block[j]!=NULL)
            newBlocks.push_back(curr_block->child_block[j]);
        }
        if(curr_block->value[j]==INT_MAX && curr_block->child_block[j]!=NULL)
            newBlocks.push_back(curr_block->child_block[j]);

        cout << "]  ";
    }

    if(newBlocks.size()==0)
    {
        //puts("");
        puts("");
        Blocks.clear();
    }
    else
    {
        //puts("");
        puts("");
        Blocks.clear();
        PRINT_TREE(newBlocks);
    }
}

void SEARCH(Block *curr_block, int val, int curBlockPosition){


     bool isLeaf;
     if(curr_block->child_block[0]==NULL)
        isLeaf = true;
     else isLeaf = false;

     int prev_sib_first = curr_block->value[0];

     for(int i=0;exist_flag==false &&  i<=curr_block->key; i++)
     {
        if(val < curr_block->value[i] && curr_block->child_block[i] != NULL)
```

```
        {
            level++;
            SEARCH(curr_block->child_block[i], val, i);

        }
        else if(val == curr_block->value[i] && curr_block->child_block[i] == NULL)
        {

            exist_flag = true;
            break;
        }
     }
}

int main(){
    char num[100];


    //cout<<"Enter Maximum Degree of the B+ Tree: ";
    //cin>>children;
    children=4;

    vector < Block* > Blocks;


    char ch;
    int i = 0;
    int total_vals = 0;

    cout<<"i:Insert a value\np:Print the tree\ns:Search for a value\ne:Exit\n";
    cin>>ch;
    while(ch!='e')
    {

        switch(ch)
        {
            case 'i':
                                cin>>num[i];
                                cout<<"Inserted "<<num[i]<<"\n";
                                INSERT(root_block, int(num[i]));
                                        i++;
                                        total_vals++;
                                        break;
                    case 'p':
```

```
                        Blocks.clear();
                        Blocks.push_back(root_block);
                        PRINT_TREE(Blocks);
                        //puts("");
                        break;
                case 'd':
                        int del;
                        cin>>del;
                        if(total_vals==0)
                        {
                                cout<<"Tree is empty\n";
                                break;
                        }

                        exist_flag = false;
                        DELETE(root_block, del, 0);
                        if (exist_flag==true)
                        {
                                total_vals--;
                                cout<<"Deleted "<<del<<"\n";
                        }
                        else
                                cout<<"ERROR\n";

                        //cout<<exist_flag<<endl;
                        break;
                case 's':       char ser;
                        cin>>ser;
                        exist_flag = false;
                        level=0;
                        SEARCH(root_block, int(ser), 0);
                        if(exist_flag==true)
                                cout<<"TRUE"<<"\n";
                        else
                                cout<<"FALSE\n";

    }

    cin>>ch;
}



return 0;
```

```
}
```