//Assignment 6a
//Sarvesh suryavanshi
//123B1B065

Problem statement: a. Implement B-Tree of order three and perform following operations: 1. Insert 2. Level order display 3. Delete

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

#define MAX 3
#define MIN 2

struct btNode {
    int keys[MAX];
    btNode* children[MAX + 1];
    int count;
    bool leaf;
};

btNode* createNode(bool leaf) {
    btNode* node = new btNode;
    node->leaf = leaf;
    node->count = 0;
    for (int i = 0; i < MAX + 1; i++) node->children[i] = NULL;
    return node;
}

void levelOrder(btNode* root) {
    if (!root) return;
    queue<btNode*> q;
    q.push(root);
    while (!q.empty()) {
        int n = q.size();
        while (n--) {
            btNode* node = q.front();
            q.pop();
            for (int i = 0; i < node->count; i++) cout << node->keys[i] << " ";
            cout << "| ";
```

```
            if (!node->leaf) {
                for (int i = 0; i <= node->count; i++)
                    if (node->children[i]) q.push(node->children[i]);
            }
        }
        cout << endl;
    }
}

void splitChild(btNode* parent, int i) {
    btNode* fullChild = parent->children[i];
    btNode* newNode = createNode(fullChild->leaf);
    newNode->count = MIN - 1;

    for (int j = 0; j < MIN - 1; j++)
        newNode->keys[j] = fullChild->keys[j + MIN];

    if (!fullChild->leaf) {
        for (int j = 0; j < MIN; j++)
            newNode->children[j] = fullChild->children[j + MIN];
    }

    for (int j = parent->count; j >= i + 1; j--)
        parent->children[j + 1] = parent->children[j];

    parent->children[i + 1] = newNode;

    for (int j = parent->count - 1; j >= i; j--)
        parent->keys[j + 1] = parent->keys[j];

    parent->keys[i] = fullChild->keys[MIN - 1];
    parent->count++;
    fullChild->count = MIN - 1;
}

void insertNonFull(btNode* node, int k) {
    int i = node->count - 1;
    if (node->leaf) {
        while (i >= 0 && k < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = k;
        node->count++;
```

```
    } else {
        while (i >= 0 && k < node->keys[i]) i--;
        i++;
        if (node->children[i]->count == MAX) {
            splitChild(node, i);
            if (k > node->keys[i]) i++;
        }
        insertNonFull(node->children[i], k);
    }
}

void insert(btNode*& root, int k) {
    if (root->count == MAX) {
        btNode* s = createNode(false);
        s->children[0] = root;
        splitChild(s, 0);
        int i = (k < s->keys[0]) ? 0 : 1;
        insertNonFull(s->children[i], k);
        root = s;
    } else {
        insertNonFull(root, k);
    }
}

int main() {
    btNode* root = NULL;
    root = createNode(true);

    int n, val;
    cout << "Enter number of keys to insert: ";
    cin >> n;

    cout << "Enter keys:\n";
    for (int i = 0; i < n; i++) {
        cin >> val;
        insert(root, val);
    }

    cout << "\nLevel Order Traversal of B-Tree:\n";
    levelOrder(root);

    return 0;
}
```

Output

Clear

```
Input:

Enter number of keys to insert: 10
Enter keys:
10 20 5 6 12 30 7 17 3 2

Level Order Traversal of B-Tree:
10 |
5 6 | 20 30 |
2 3 | 7 | 12 | 17 |
```