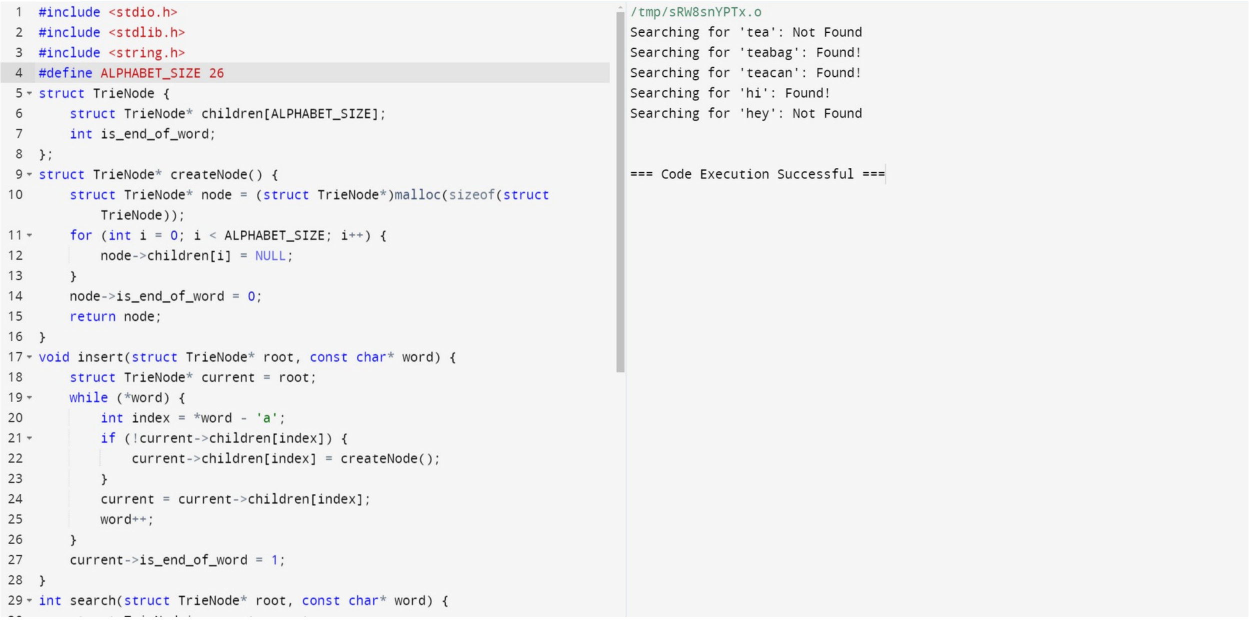
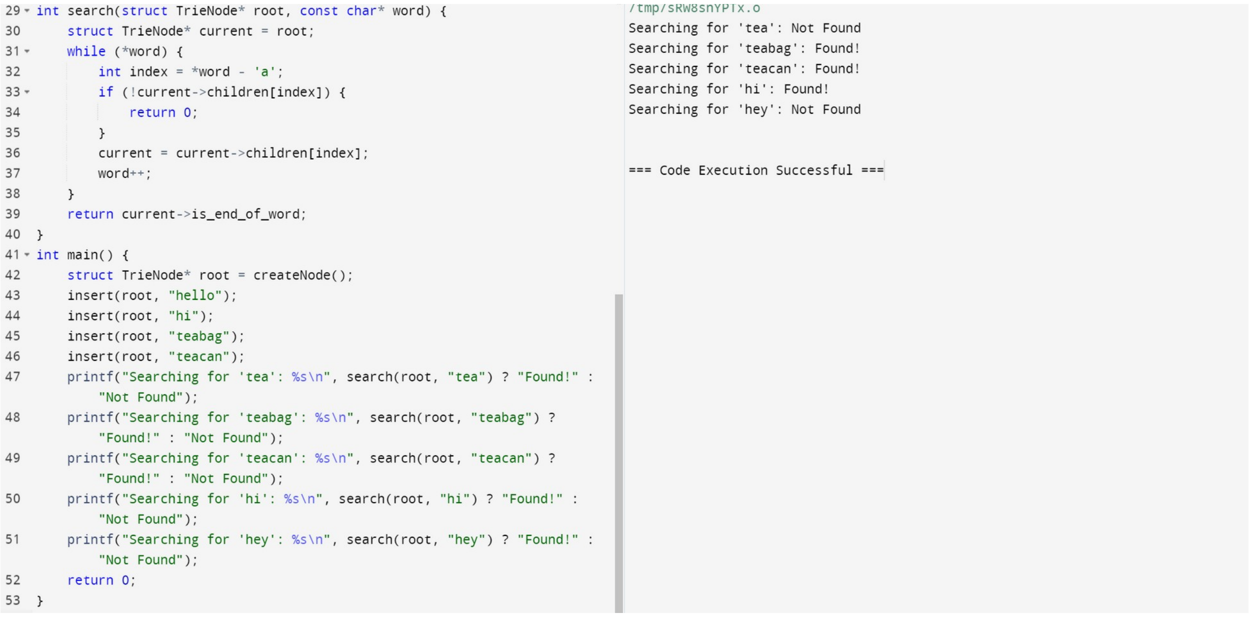
1.Write a c program for TRIE.



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define ALPHABET_SIZE 26
struct TrieNode {
    struct TrieNode* children[ALPHABET_SIZE];
    int is_end_of_word;
};
struct TrieNode* createNode() {
    struct TrieNode* node = (struct TrieNode*)malloc(sizeof(struct TrieNode));
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        node->children[i] = NULL;
    }
    node->is_end_of_word = 0;
    return node;
}
void insert(struct TrieNode* root, const char* word) {
    struct TrieNode* current = root;
    while (*word) {
        int index = *word - 'a';
        if (!current->children[index]) {
            current->children[index] = createNode();
        }
        current = current->children[index];
        word++;
    }
    current->is_end_of_word = 1;
}
int search(struct TrieNode* root, const char* word) {
```

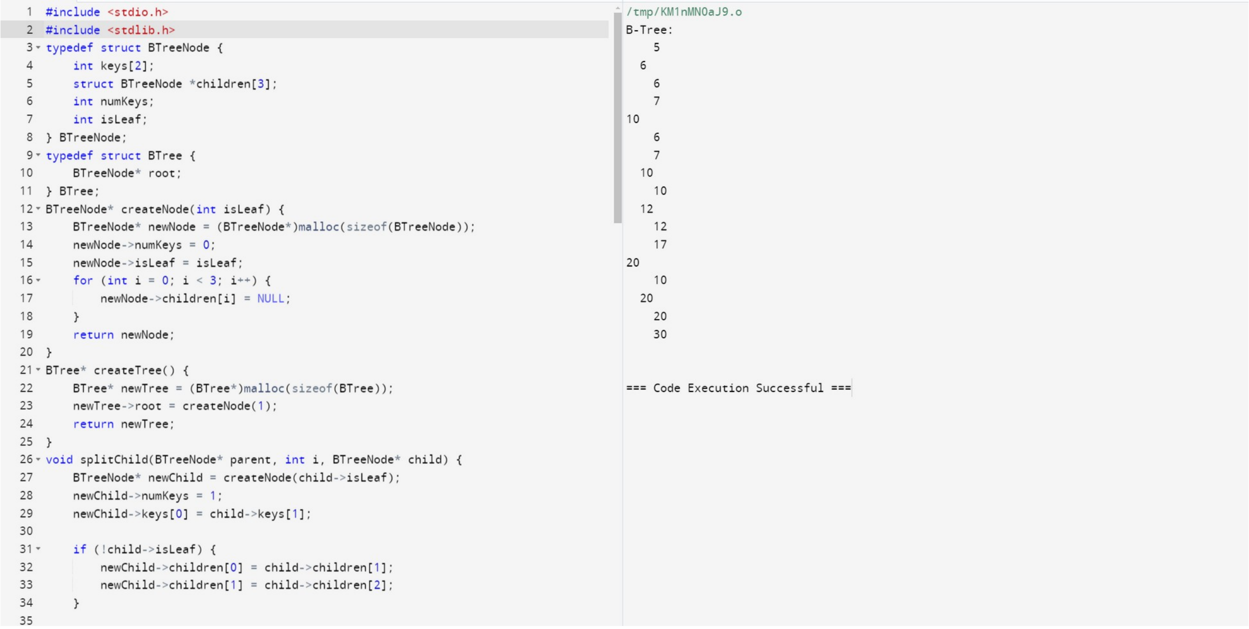
```
/tmp/sRW8snYPTx.o
Searching for 'tea': Not Found
Searching for 'teabag': Found!
Searching for 'teacan': Found!
Searching for 'hi': Found!
Searching for 'hey': Not Found


=== Code Execution Successful ===
```



```c
int search(struct TrieNode* root, const char* word) {
    struct TrieNode* current = root;
    while (*word) {
        int index = *word - 'a';
        if (!current->children[index]) {
            return 0;
        }
        current = current->children[index];
        word++;
    }
    return current->is_end_of_word;
}
int main() {
    struct TrieNode* root = createNode();
    insert(root, "hello");
    insert(root, "hi");
    insert(root, "teabag");
    insert(root, "teacan");
    printf("Searching for 'tea': %s\n", search(root, "tea") ? "Found!" : "Not Found");
    printf("Searching for 'teabag': %s\n", search(root, "teabag") ? "Found!" : "Not Found");
    printf("Searching for 'teacan': %s\n", search(root, "teacan") ? "Found!" : "Not Found");
    printf("Searching for 'hi': %s\n", search(root, "hi") ? "Found!" : "Not Found");
    printf("Searching for 'hey': %s\n", search(root, "hey") ? "Found!" : "Not Found");
    return 0;
}
```

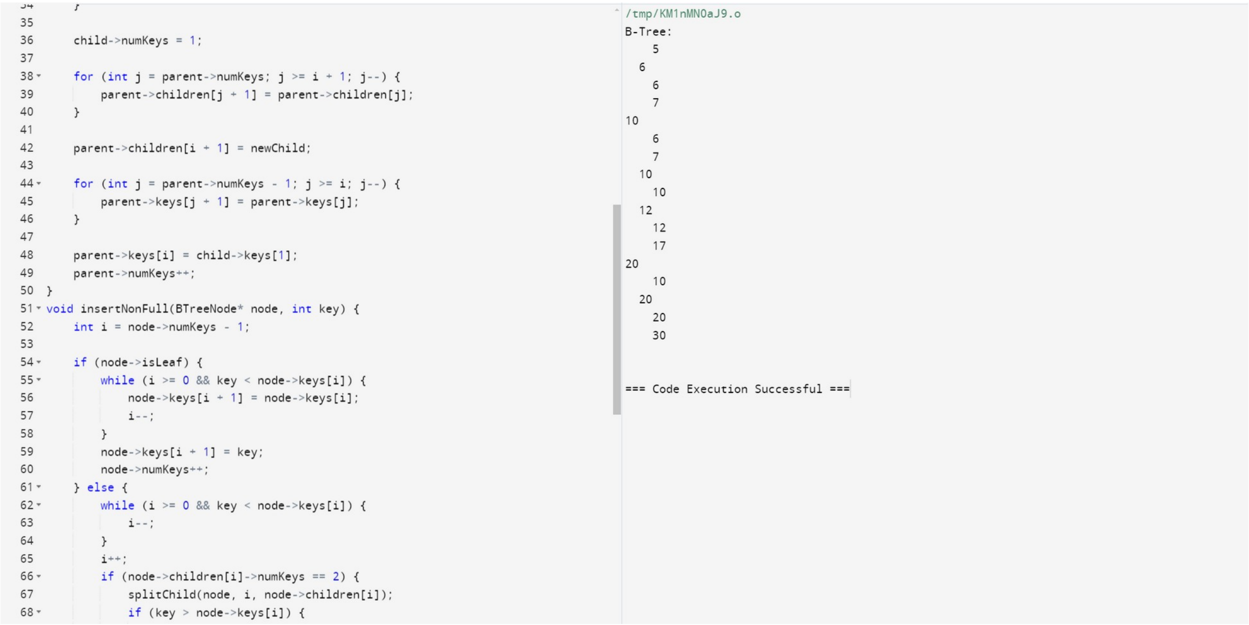
```
/tmp/sRW8snYPTx.o
Searching for 'tea': Not Found
Searching for 'teabag': Found!
Searching for 'teacan': Found!
Searching for 'hi': Found!
Searching for 'hey': Not Found


=== Code Execution Successful ===
```

2.Write a c program for B TREE (2-3).



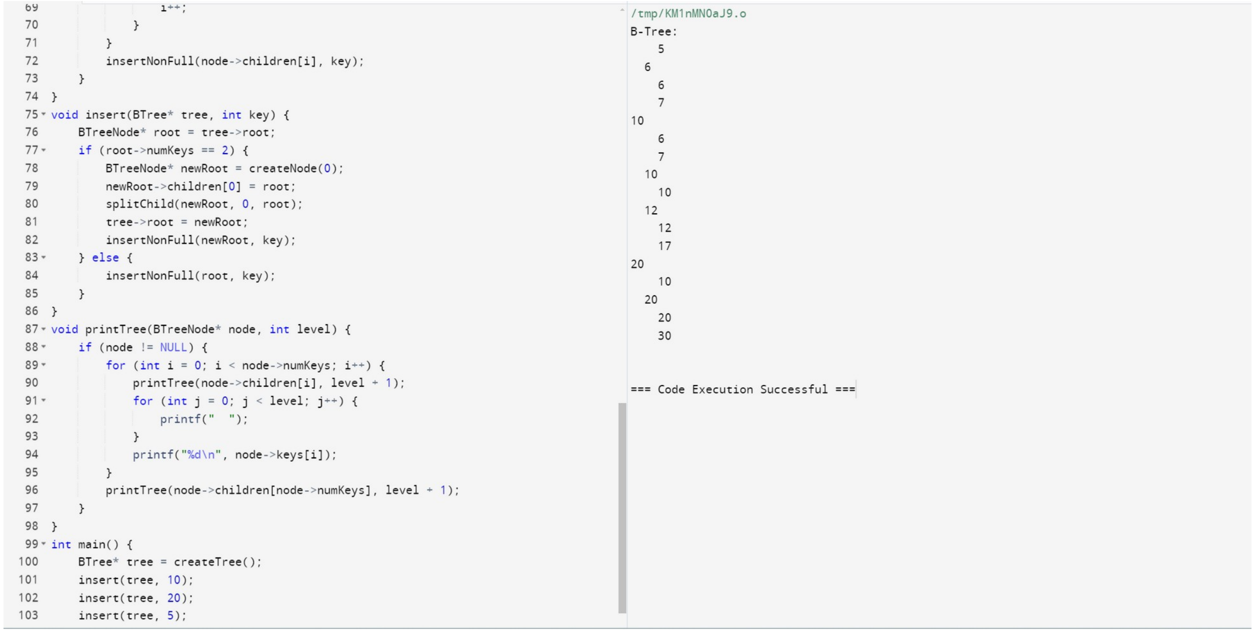
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct BTreeNode {
    int keys[2];
    struct BTreeNode *children[3];
    int numKeys;
    int isLeaf;
} BTreeNode;
typedef struct BTree {
    BTreeNode* root;
} BTree;
BTreeNode* createNode(int isLeaf) {
    BTreeNode* newNode = (BTreeNode*)malloc(sizeof(BTreeNode));
    newNode->numKeys = 0;
    newNode->isLeaf = isLeaf;
    for (int i = 0; i < 3; i++) {
        newNode->children[i] = NULL;
    }
    return newNode;
}
BTree* createTree() {
    BTree* newTree = (BTree*)malloc(sizeof(BTree));
    newTree->root = createNode(1);
    return newTree;
}
void splitChild(BTreeNode* parent, int i, BTreeNode* child) {
    BTreeNode* newChild = createNode(child->isLeaf);
    newChild->numKeys = 1;
    newChild->keys[0] = child->keys[1];

    if (!child->isLeaf) {
        newChild->children[0] = child->children[1];
        newChild->children[1] = child->children[2];
    }

```

```
/tmp/KM1nMN0aJ9.o
B-Tree:
    5
  6
    6
    7
10
    6
    7
  10
    10
  12
    12
    17
20
    10
  20
    20
    30


=== Code Execution Successful ===
```



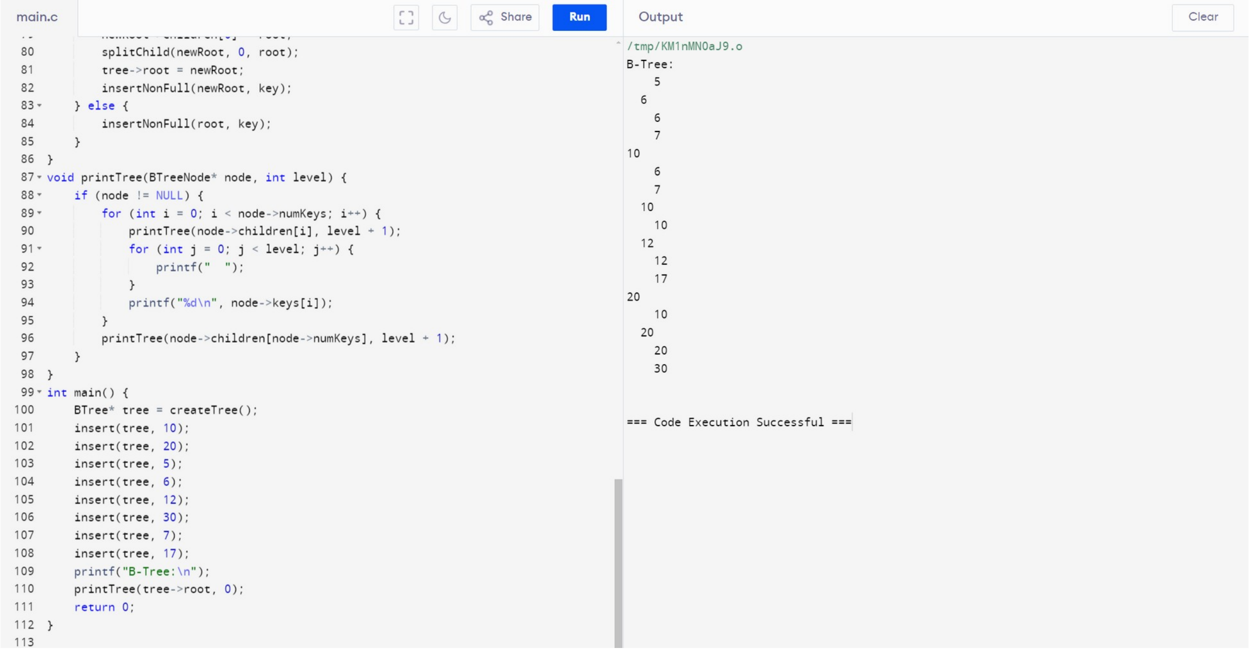
```c
    }

    child->numKeys = 1;

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

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

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

    parent->keys[i] = child->keys[1];
    parent->numKeys++;
}
void insertNonFull(BTreeNode* node, int key) {
    int i = node->numKeys - 1;

    if (node->isLeaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->numKeys++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->numKeys == 2) {
            splitChild(node, i, node->children[i]);
            if (key > node->keys[i]) {
```

```
/tmp/KM1nMN0aJ9.o
B-Tree:
    5
  6
    6
    7
10
    6
    7
  10
    10
  12
    12
    17
20
    10
  20
    20
    30


=== Code Execution Successful ===
```



```c
                i++;
            }
        }
        insertNonFull(node->children[i], key);
    }
}
void insert(BTree* tree, int key) {
    BTreeNode* root = tree->root;
    if (root->numKeys == 2) {
        BTreeNode* newRoot = createNode(0);
        newRoot->children[0] = root;
        splitChild(newRoot, 0, root);
        tree->root = newRoot;
        insertNonFull(newRoot, key);
    } else {
        insertNonFull(root, key);
    }
}
void printTree(BTreeNode* node, int level) {
    if (node != NULL) {
        for (int i = 0; i < node->numKeys; i++) {
            printTree(node->children[i], level + 1);
            for (int j = 0; j < level; j++) {
                printf("  ");
            }
            printf("%d\n", node->keys[i]);
        }
        printTree(node->children[node->numKeys], level + 1);
    }
}
int main() {
    BTree* tree = createTree();
    insert(tree, 10);
    insert(tree, 20);
    insert(tree, 5);
```

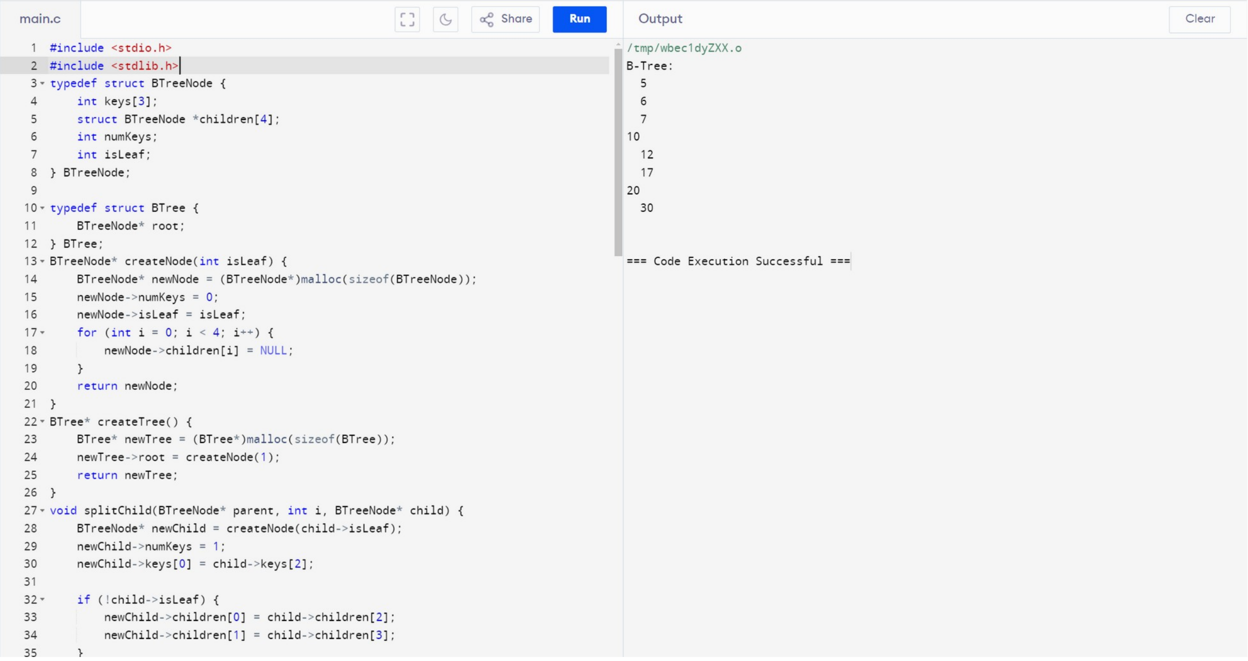
```
/tmp/KM1nMN0aJ9.o
B-Tree:
    5
  6
    6
    7
10
    6
    7
  10
    10
  12
    12
    17
20
    10
  20
    20
    30


=== Code Execution Successful ===
```



```c
        splitChild(newRoot, 0, root);
        tree->root = newRoot;
        insertNonFull(newRoot, key);
    } else {
        insertNonFull(root, key);
    }
}
void printTree(BTreeNode* node, int level) {
    if (node != NULL) {
        for (int i = 0; i < node->numKeys; i++) {
            printTree(node->children[i], level + 1);
            for (int j = 0; j < level; j++) {
                printf("  ");
            }
            printf("%d\n", node->keys[i]);
        }
        printTree(node->children[node->numKeys], level + 1);
    }
}
int main() {
    BTree* tree = createTree();
    insert(tree, 10);
    insert(tree, 20);
    insert(tree, 5);
    insert(tree, 6);
    insert(tree, 12);
    insert(tree, 30);
    insert(tree, 7);
    insert(tree, 17);
    printf("B-Tree:\n");
    printTree(tree->root, 0);
    return 0;
}
```

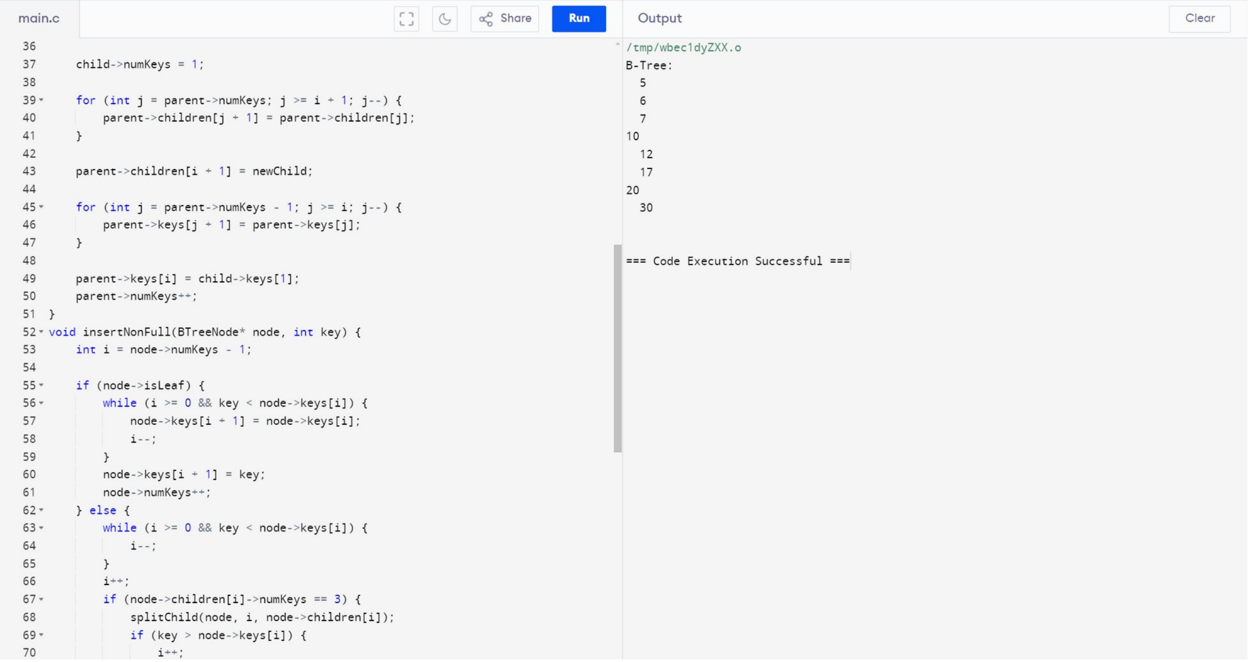
```
/tmp/KM1nMN0aJ9.o
B-Tree:
    5
  6
    6
    7
10
    6
    7
  10
    10
  12
    12
    17
20
    10
  20
    20
    30


=== Code Execution Successful ===
```

3.Write a c program for B TREE (2-3-4).



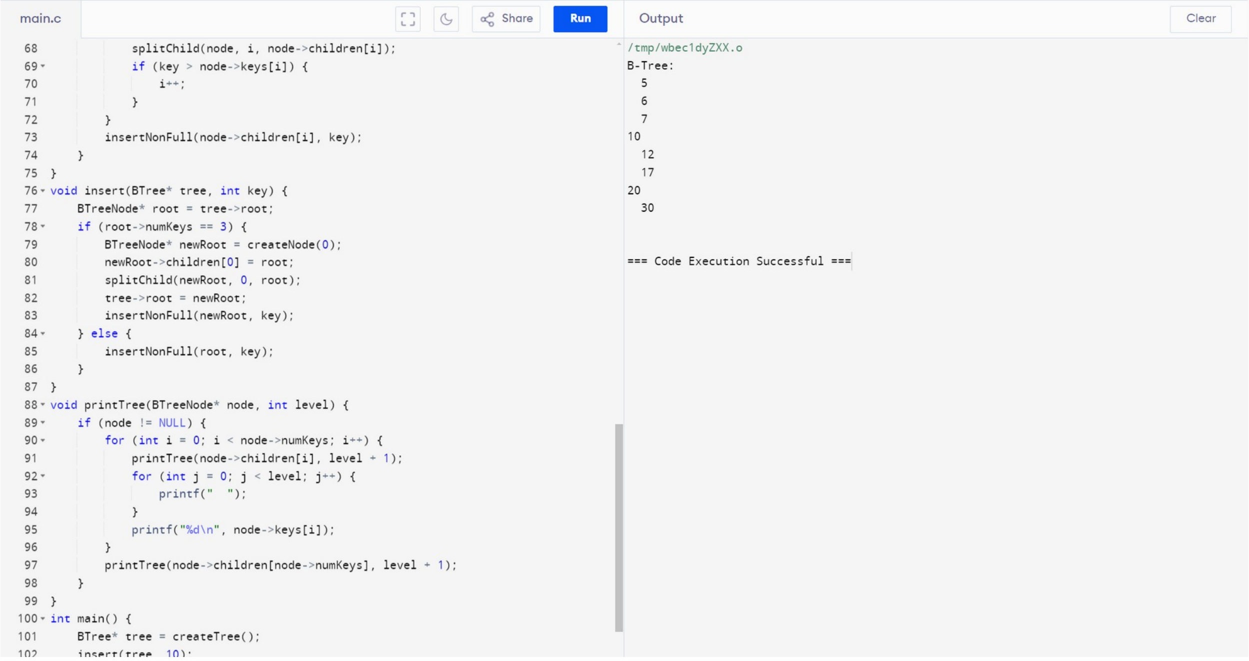
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct BTreeNode {
    int keys[3];
    struct BTreeNode *children[4];
    int numKeys;
    int isLeaf;
} BTreeNode;

typedef struct BTree {
    BTreeNode* root;
} BTree;
BTreeNode* createNode(int isLeaf) {
    BTreeNode* newNode = (BTreeNode*)malloc(sizeof(BTreeNode));
    newNode->numKeys = 0;
    newNode->isLeaf = isLeaf;
    for (int i = 0; i < 4; i++) {
        newNode->children[i] = NULL;
    }
    return newNode;
}
BTree* createTree() {
    BTree* newTree = (BTree*)malloc(sizeof(BTree));
    newTree->root = createNode(1);
    return newTree;
}
void splitChild(BTreeNode* parent, int i, BTreeNode* child) {
    BTreeNode* newChild = createNode(child->isLeaf);
    newChild->numKeys = 1;
    newChild->keys[0] = child->keys[2];

    if (!child->isLeaf) {
        newChild->children[0] = child->children[2];
        newChild->children[1] = child->children[3];
    }
```

```
/tmp/wbec1dyZXX.o
B-Tree:
  5
  6
  7
10
  12
  17
20
  30


=== Code Execution Successful ===
```



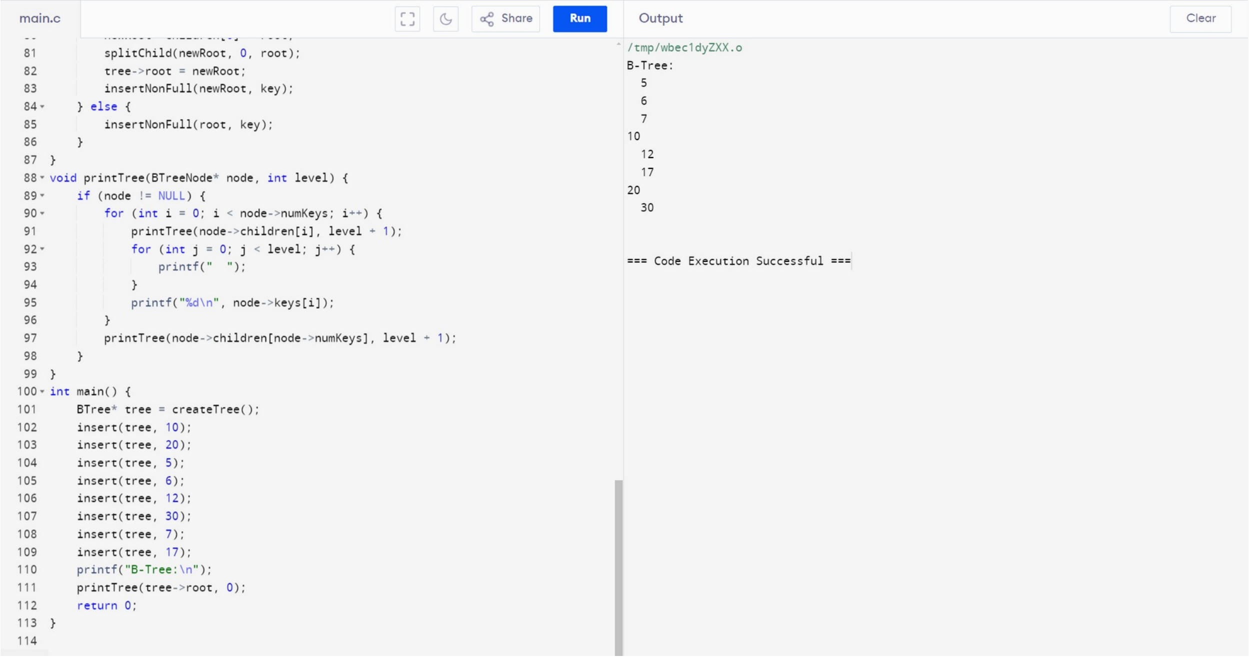
```c

    child->numKeys = 1;

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

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

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

    parent->keys[i] = child->keys[1];
    parent->numKeys++;
}
void insertNonFull(BTreeNode* node, int key) {
    int i = node->numKeys - 1;

    if (node->isLeaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->numKeys++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->numKeys == 3) {
            splitChild(node, i, node->children[i]);
            if (key > node->keys[i]) {
                i++;
```

```
/tmp/wbec1dyZXX.o
B-Tree:
  5
  6
  7
10
  12
  17
20
  30


=== Code Execution Successful ===
```



```c
            splitChild(node, i, node->children[i]);
            if (key > node->keys[i]) {
                i++;
            }
        }
        insertNonFull(node->children[i], key);
    }
}
void insert(BTree* tree, int key) {
    BTreeNode* root = tree->root;
    if (root->numKeys == 3) {
        BTreeNode* newRoot = createNode(0);
        newRoot->children[0] = root;
        splitChild(newRoot, 0, root);
        tree->root = newRoot;
        insertNonFull(newRoot, key);
    } else {
        insertNonFull(root, key);
    }
}
void printTree(BTreeNode* node, int level) {
    if (node != NULL) {
        for (int i = 0; i < node->numKeys; i++) {
            printTree(node->children[i], level + 1);
            for (int j = 0; j < level; j++) {
                printf("  ");
            }
            printf("%d\n", node->keys[i]);
        }
        printTree(node->children[node->numKeys], level + 1);
    }
}
int main() {
    BTree* tree = createTree();
    insert(tree, 10);
```

```
/tmp/wbec1dyZXX.o
B-Tree:
  5
  6
  7
10
  12
  17
20
  30


=== Code Execution Successful ===
```

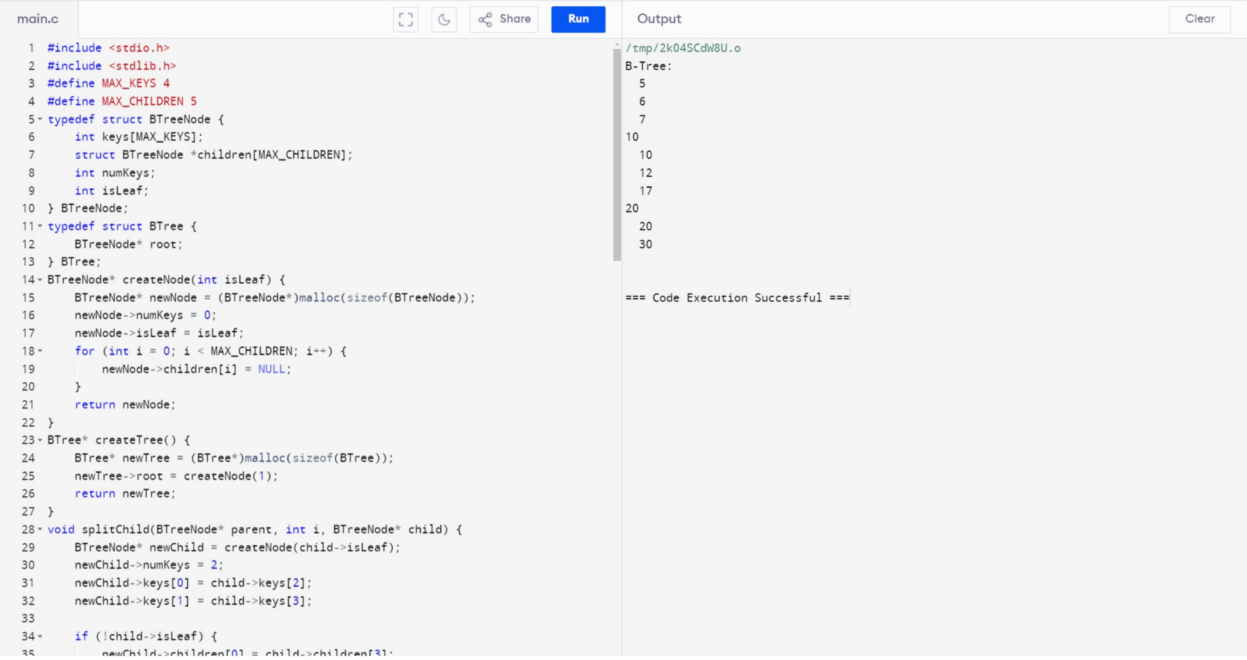


```c
        splitChild(newRoot, 0, root);
        tree->root = newRoot;
        insertNonFull(newRoot, key);
    } else {
        insertNonFull(root, key);
    }
}
void printTree(BTreeNode* node, int level) {
    if (node != NULL) {
        for (int i = 0; i < node->numKeys; i++) {
            printTree(node->children[i], level + 1);
            for (int j = 0; j < level; j++) {
                printf("  ");
            }
            printf("%d\n", node->keys[i]);
        }
        printTree(node->children[node->numKeys], level + 1);
    }
}
int main() {
    BTree* tree = createTree();
    insert(tree, 10);
```

```c
    insert(tree, 20);
    insert(tree, 5);
    insert(tree, 6);
    insert(tree, 12);
    insert(tree, 30);
    insert(tree, 7);
    insert(tree, 17);
    printf("B-Tree:\n");
    printTree(tree->root, 0);
    return 0;
}
```

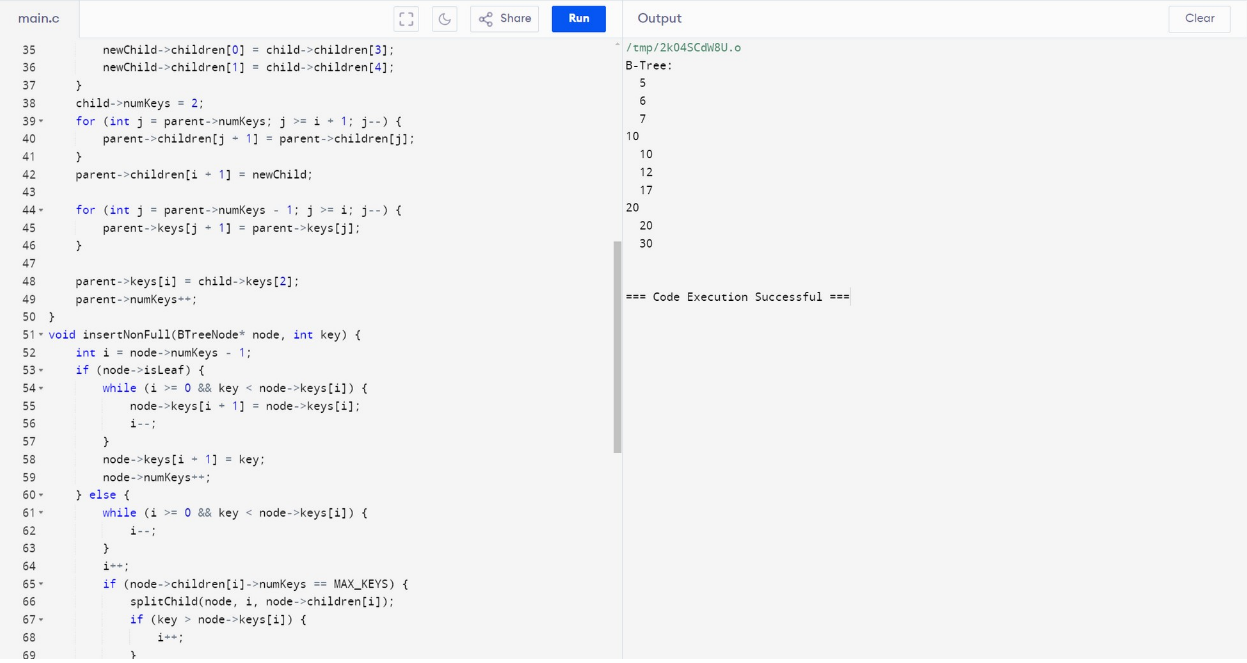
```
/tmp/wbec1dyZXX.o
B-Tree:
  5
  6
  7
10
  12
  17
20
  30


=== Code Execution Successful ===
```

4.Write a c program for B TREE (2-3-4-5).



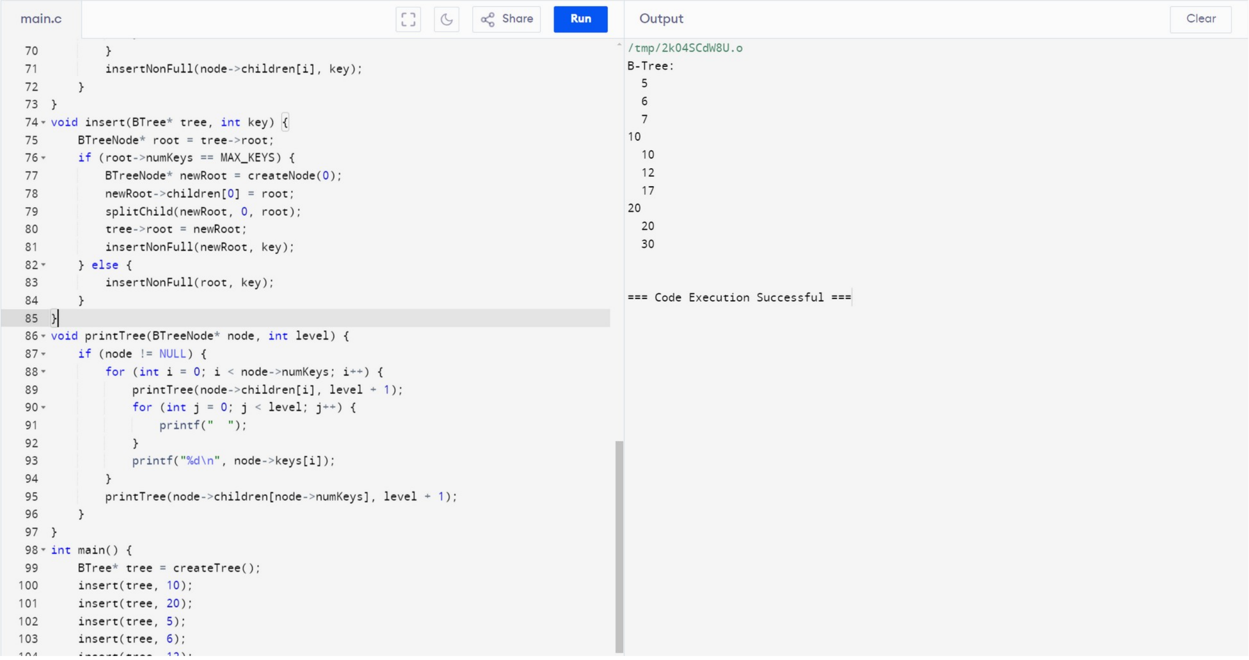
```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_KEYS 4
#define MAX_CHILDREN 5
typedef struct BTreeNode {
    int keys[MAX_KEYS];
    struct BTreeNode *children[MAX_CHILDREN];
    int numKeys;
    int isLeaf;
} BTreeNode;
typedef struct BTree {
    BTreeNode* root;
} BTree;
BTreeNode* createNode(int isLeaf) {
    BTreeNode* newNode = (BTreeNode*)malloc(sizeof(BTreeNode));
    newNode->numKeys = 0;
    newNode->isLeaf = isLeaf;
    for (int i = 0; i < MAX_CHILDREN; i++) {
        newNode->children[i] = NULL;
    }
    return newNode;
}
BTree* createTree() {
    BTree* newTree = (BTree*)malloc(sizeof(BTree));
    newTree->root = createNode(1);
    return newTree;
}
void splitChild(BTreeNode* parent, int i, BTreeNode* child) {
    BTreeNode* newChild = createNode(child->isLeaf);
    newChild->numKeys = 2;
    newChild->keys[0] = child->keys[2];
    newChild->keys[1] = child->keys[3];

    if (!child->isLeaf) {
        newChild->children[0] = child->children[3];
```

```
/tmp/2k04SCdW8U.o
B-Tree:
  5
  6
  7
10
  10
  12
  17
20
  20
  30


=== Code Execution Successful ===
```



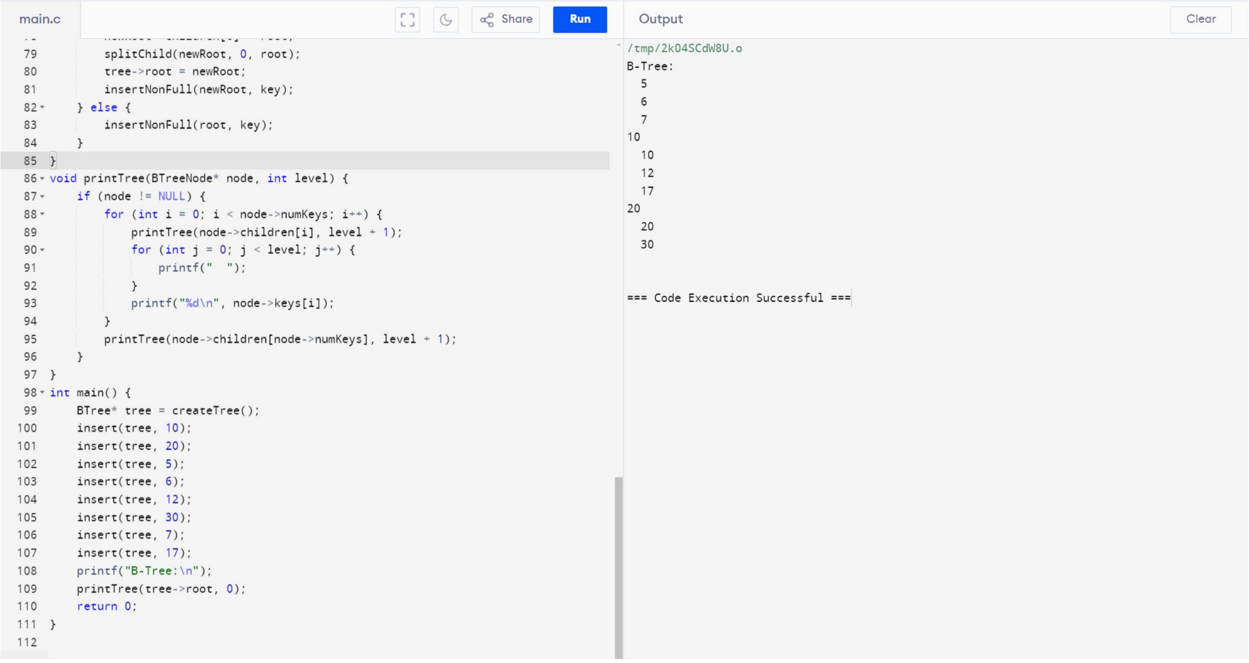
```c
        newChild->children[0] = child->children[3];
        newChild->children[1] = child->children[4];
    }
    child->numKeys = 2;
    for (int j = parent->numKeys; j >= i + 1; j--) {
        parent->children[j + 1] = parent->children[j];
    }
    parent->children[i + 1] = newChild;

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

    parent->keys[i] = child->keys[2];
    parent->numKeys++;
}
void insertNonFull(BTreeNode* node, int key) {
    int i = node->numKeys - 1;
    if (node->isLeaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->numKeys++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->numKeys == MAX_KEYS) {
            splitChild(node, i, node->children[i]);
            if (key > node->keys[i]) {
                i++;
            }
```

```
/tmp/2k04SCdW8U.o
B-Tree:
  5
  6
  7
10
  10
  12
  17
20
  20
  30


=== Code Execution Successful ===
```



```
        }
        insertNonFull(node->children[i], key);
    }
}
void insert(BTree* tree, int key) {
    BTreeNode* root = tree->root;
    if (root->numKeys == MAX_KEYS) {
        BTreeNode* newRoot = createNode(0);
        newRoot->children[0] = root;
        splitChild(newRoot, 0, root);
        tree->root = newRoot;
        insertNonFull(newRoot, key);
    } else {
        insertNonFull(root, key);
    }
}
void printTree(BTreeNode* node, int level) {
    if (node != NULL) {
        for (int i = 0; i < node->numKeys; i++) {
            printTree(node->children[i], level + 1);
            for (int j = 0; j < level; j++) {
                printf("  ");
            }
            printf("%d\n", node->keys[i]);
        }
        printTree(node->children[node->numKeys], level + 1);
    }
}
int main() {
    BTree* tree = createTree();
    insert(tree, 10);
    insert(tree, 20);
    insert(tree, 5);
    insert(tree, 6);
```

```
/tmp/2k04SCdW8U.o
B-Tree:
  5
  6
  7
10
  10
  12
  17
20
  20
  30


=== Code Execution Successful ===
```



```
        splitChild(newRoot, 0, root);
        tree->root = newRoot;
        insertNonFull(newRoot, key);
    } else {
        insertNonFull(root, key);
    }
}
void printTree(BTreeNode* node, int level) {
    if (node != NULL) {
        for (int i = 0; i < node->numKeys; i++) {
            printTree(node->children[i], level + 1);
            for (int j = 0; j < level; j++) {
                printf("  ");
            }
            printf("%d\n", node->keys[i]);
        }
        printTree(node->children[node->numKeys], level + 1);
    }
}
int main() {
    BTree* tree = createTree();
    insert(tree, 10);
    insert(tree, 20);
    insert(tree, 5);
    insert(tree, 6);
    insert(tree, 12);
    insert(tree, 30);
    insert(tree, 7);
    insert(tree, 17);
    printf("B-Tree:\n");
    printTree(tree->root, 0);
    return 0;
}

```

```
/tmp/2k04SCdW8U.o
B-Tree:
  5
  6
  7
10
  10
  12
  17
20
  20
  30


=== Code Execution Successful ===
```