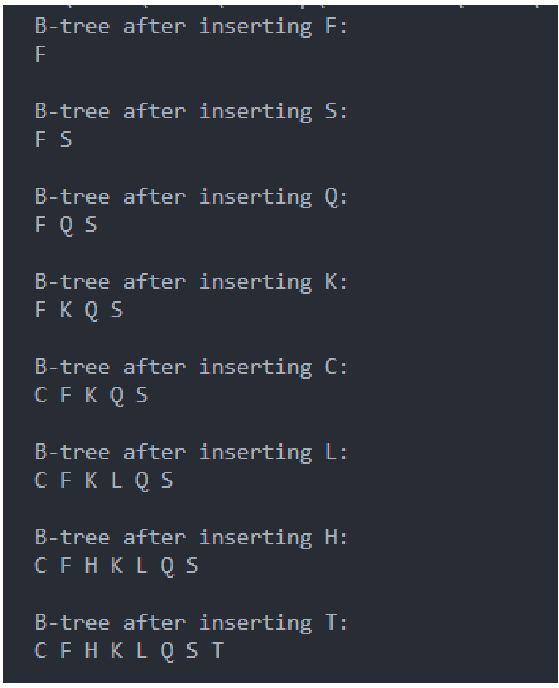
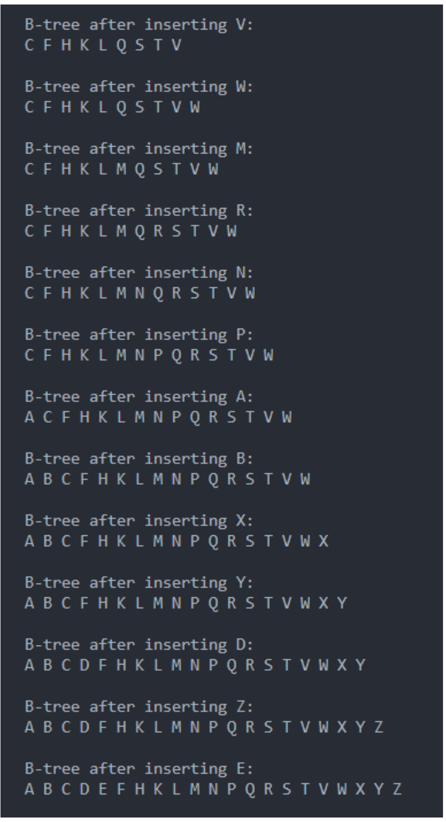
1. Write a C program for creation of B tree having minimum degree t. Show the results of inserting the keys F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E in order into an empty B-tree with minimum degree 2.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MIN_DEGREE 2

typedef struct BTreeNode {
    int *keys;
    int t;
    struct BTreeNode **C;
    int n;
    bool leaf;
} BTreeNode;

typedef struct BTree {
    BTreeNode *root;
    int t;
} BTree;

BTreeNode *createNode(int t, bool leaf) {
    BTreeNode *newNode = (BTreeNode *)malloc(sizeof(BTreeNode));
    newNode->t = t;
    newNode->leaf = leaf;
    newNode->keys = (int *)malloc((2 * t - 1) * sizeof(int));
    newNode->C = (BTreeNode **)malloc((2 * t) * sizeof(BTreeNode *));
    newNode->n = 0;
    return newNode;
}

void traverse(BTreeNode *root) {
    if (root != NULL) {
        int i;
        for (i = 0; i < root->n; i++) {
            if (!root->leaf)
                traverse(root->C[i]);
            printf("%c ", root->keys[i]);
        }
        if (!root->leaf)
            traverse(root->C[i]);
    }
}
```

```c
BTreeNode *search(BTreeNode *root, int k) {
    int i = 0;
    while (i < root->n && k > root->keys[i])
        i++;

    if (root->keys[i] == k)
        return root;

    if (root->leaf)
        return NULL;

    return search(root->C[i], k);
}

void splitChild(BTreeNode *x, int i, BTreeNode *y) {
    int t = y->t;
    BTreeNode *z = createNode(t, y->leaf);
    z->n = t - 1;

    for (int j = 0; j < t - 1; j++)
        z->keys[j] = y->keys[j + t];

    if (!y->leaf) {
        for (int j = 0; j < t; j++)
            z->C[j] = y->C[j + t];
    }

    y->n = t - 1;

    for (int j = x->n; j >= i + 1; j--)
        x->C[j + 1] = x->C[j];

    x->C[i + 1] = z;

    for (int j = x->n - 1; j >= i; j--)
        x->keys[j + 1] = x->keys[j];

    x->keys[i] = y->keys[t - 1];
    x->n = x->n + 1;
}

void insertNonFull(BTreeNode *x, int k) {
    int i = x->n - 1;

    if (x->leaf) {
        while (i >= 0 && x->keys[i] > k) {
            x->keys[i + 1] = x->keys[i];
            i--;
        }

        x->keys[i + 1] = k;
        x->n = x->n + 1;
    } else {
        while (i >= 0 && x->keys[i] > k)
            i--;

        if (x->C[i + 1]->n == 2 * x->t - 1) {
            splitChild(x, i + 1, x->C[i + 1]);

            if (x->keys[i + 1] < k)
                i++;
        }
        insertNonFull(x->C[i + 1], k);
    }
}

void insert(BTree *tree, int k) {
    if (tree->root == NULL) {
        tree->root = createNode(tree->t, true);
        tree->root->keys[0] = k;
        tree->root->n = 1;
    } else {
        if (tree->root->n == 2 * tree->t - 1) {
            BTreeNode *s = createNode(tree->t, false);
            s->C[0] = tree->root;
            splitChild(s, 0, tree->root);
            int i = 0;
            if (s->keys[0] < k)
                i++;
            insertNonFull(s->C[i], k);
            tree->root = s;
        } else {
            insertNonFull(tree->root, k);
        }
    }
}

int main() {
    BTree tree;
    tree.root = NULL;
    tree.t = MIN_DEGREE;
    char keys[] = {'F', 'S', 'Q', 'K', 'C', 'L', 'H', 'T', 'V', 'W',
'M', 'R', 'N', 'P', 'A', 'B', 'X', 'Y', 'D', 'Z', 'E'};
    int n = sizeof(keys) / sizeof(keys[0]);

    for (int i = 0; i < n; i++) {
        insert(&tree, keys[i]);
        printf("B-tree after inserting %c:\n", keys[i]);
        traverse(tree.root);
        printf("\n\n");
    }

    return 0;
}
```



```
B-tree after inserting F:
F

B-tree after inserting S:
F S

B-tree after inserting Q:
F Q S

B-tree after inserting K:
F K Q S

B-tree after inserting C:
C F K Q S

B-tree after inserting L:
C F K L Q S

B-tree after inserting H:
C F H K L Q S

B-tree after inserting T:
C F H K L Q S T
```



```
B-tree after inserting V:
C F H K L Q S T V

B-tree after inserting W:
C F H K L Q S T V W

B-tree after inserting M:
C F H K L M Q S T V W

B-tree after inserting R:
C F H K L M Q R S T V W

B-tree after inserting N:
C F H K L M N Q R S T V W

B-tree after inserting P:
C F H K L M N P Q R S T V W

B-tree after inserting A:
A C F H K L M N P Q R S T V W

B-tree after inserting B:
A B C F H K L M N P Q R S T V W

B-tree after inserting X:
A B C F H K L M N P Q R S T V W X

B-tree after inserting Y:
A B C F H K L M N P Q R S T V W X Y

B-tree after inserting D:
A B C D F H K L M N P Q R S T V W X Y

B-tree after inserting Z:
A B C D F H K L M N P Q R S T V W X Y Z

B-tree after inserting E:
A B C D E F H K L M N P Q R S T V W X Y Z
```

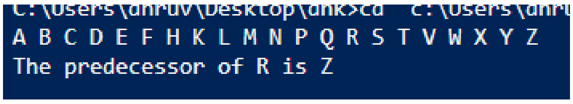
## 2.Write a C program for computing the predecessor of a key in B tree having minimum degree t.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define MIN_DEGREE 2

typedef struct BTreeNode
{
    int *keys;
    int t;
    struct BTreeNode **C;
    int n;
    bool leaf;
} BTreeNode;

typedef struct BTree
{
    BTreeNode *root;
    int t;
} BTree;

BTreeNode *createNode(int t, bool leaf)
{
    BTreeNode *newNode = (BTreeNode *)malloc(sizeof(BTreeNode));
    newNode->t = t;
    newNode->leaf = leaf;
    newNode->keys = (int *)malloc((2 * t - 1) * sizeof(int));
    newNode->C = (BTreeNode **)malloc((2 * t) * sizeof(BTreeNode *));
    newNode->n = 0;
    return newNode;
}

void traverse(BTreeNode *root)
{
    if (root != NULL)
    {
        int i;
        for (i = 0; i < root->n; i++)
        {
            if (!root->leaf)
                traverse(root->C[i]);
            printf("%c ", root->keys[i]);
        }
        if (!root->leaf)
            traverse(root->C[i]);
    }
}

void splitChild(BTreeNode *x, int i, BTreeNode *y)
{
    int t = y->t;
    BTreeNode *z = createNode(t, y->leaf);
    z->n = t - 1;

    for (int j = 0; j < t - 1; j++)
```

```
        z->keys[j] = y->keys[j + t];

    if (!y->leaf)
    {
        for (int j = 0; j < t; j++)
            z->C[j] = y->C[j + t];
    }

    y->n = t - 1;

    for (int j = x->n; j >= i + 1; j--)
        x->C[j + 1] = x->C[j];

    x->C[i + 1] = z;

    for (int j = x->n - 1; j >= i; j--)
        x->keys[j + 1] = x->keys[j];

    x->keys[i] = y->keys[t - 1];
    x->n = x->n + 1;
}

void insertNonFull(BTreeNode *x, int k)
{
    int i = x->n - 1;

    if (x->leaf)
    {
        while (i >= 0 && x->keys[i] > k)
        {
            x->keys[i + 1] = x->keys[i];
            i--;
        }

        x->keys[i + 1] = k;
        x->n = x->n + 1;
    }
    else
    {
        while (i >= 0 && x->keys[i] > k)
            i--;

        if (x->C[i + 1]->n == 2 * x->t - 1)
        {
            splitChild(x, i + 1, x->C[i + 1]);
            if (x->keys[i + 1] < k)
                i++;
        }
        insertNonFull(x->C[i + 1], k);
    }
}

void insert(BTree *tree, int k)
{
    if (tree->root == NULL)
    {
        tree->root = createNode(tree->t, true);
        tree->root->keys[0] = k;
        tree->root->n = 1;
    }
    else
    {
        if (tree->root->n == 2 * tree->t - 1)
        {
            BTreeNode *s = createNode(tree->t, false);
            s->C[0] = tree->root;
            splitChild(s, 0, tree->root);
            int i = 0;
            if (s->keys[0] < k)
                i++;
            insertNonFull(s->C[i], k);
            tree->root = s;
        }
        else
        {
            insertNonFull(tree->root, k);
        }
    }
}

int findPredecessor(BTreeNode *node)
{
    BTreeNode *current = node;
    while (!current->leaf)
    {
        current = current->C[current->n];
    }
    return current->keys[current->n - 1];
}

int predecessor(BTreeNode *node, int k)
```

```
{
    int i = 0;
    while (i < node->n && node->keys[i] < k)
        i++;

    if (i > 0 && node->keys[i - 1] < k)
    {
        if (node->leaf)
        {
            return node->keys[i - 1];
        }
        else
        {
            return findPredecessor(node->C[i]);
        }
    }
    else if (!node->leaf)
    {
        return predecessor(node->C[i], k);
    }

    return -1;
}

int main()
{
    BTree tree;
    tree.root = NULL;
    tree.t = MIN_DEGREE;

    char keys[] = {'F', 'S', 'Q', 'K', 'C', 'L', 'H', 'T', 'V', 'W',
'M',
                   'R', 'N', 'P', 'A', 'B', 'X', 'Y', 'D', 'Z', 'E'};
    int n = sizeof(keys) / sizeof(keys[0]);

    for (int i = 0; i < n; i++)
    {
        insert(&tree, keys[i]);
    }

    traverse(tree.root);
    printf("\n");

    char key = 'R';
    int pred = predecessor(tree.root, key);
    if (pred != -1)
```

```
        printf("The predecessor of %c is %c\n", key, pred);
    else
        printf("The predecessor of %c doesn't exist\n", key);

    return 0;
}
```

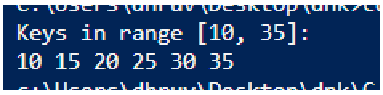


```
C:\Users\dhruv\Desktop\dhk>cd  C:\Users\dhru
A B C D E F H K L M N P Q R S T V W X Y Z
The predecessor of R is Z
```

3. Write a function to find all keys in the range [low, high] in a B-tree.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define MIN_DEGREE 2

typedef struct BTreeNode
{
    int *keys;
    int t;
    struct BTreeNode **C;
    int n;
    bool leaf;
} BTreeNode;

typedef struct BTree
{
    BTreeNode *root;
    int t;
} BTree;

BTreeNode *createNode(int t, bool leaf)
{
    BTreeNode *newNode = (BTreeNode *)malloc(sizeof(BTreeNode));
    newNode->t = t;
    newNode->leaf = leaf;
    newNode->keys = (int *)malloc((2 * t - 1) * sizeof(int));
    newNode->C = (BTreeNode **)malloc((2 * t) * sizeof(BTreeNode *));
    newNode->n = 0;
    return newNode;
}

void splitChild(BTreeNode *x, int i, BTreeNode *y)
{
    int t = y->t;
    BTreeNode *z = createNode(t, y->leaf);
    z->n = t - 1;

    for (int j = 0; j < t - 1; j++)
        z->keys[j] = y->keys[j + t];

    if (!y->leaf)
    {
        for (int j = 0; j < t; j++)
            z->C[j] = y->C[j + t];
    }

    y->n = t - 1;

    for (int j = x->n; j >= i + 1; j--)
        x->C[j + 1] = x->C[j];

    x->C[i + 1] = z;

    for (int j = x->n - 1; j >= i; j--)
        x->keys[j + 1] = x->keys[j];

    x->keys[i] = y->keys[t - 1];
    x->n = x->n + 1;
}

void insertNonFull(BTreeNode *x, int k)
{
    int i = x->n - 1;

    if (x->leaf)
    {
        while (i >= 0 && x->keys[i] > k)
        {
            x->keys[i + 1] = x->keys[i];
            i--;
        }

        x->keys[i + 1] = k;
        x->n = x->n + 1;
    }
    else
    {
        while (i >= 0 && x->keys[i] > k)
            i--;
        if (x->C[i + 1]->n == 2 * x->t - 1)
        {
            splitChild(x, i + 1, x->C[i + 1]);

            if (x->keys[i + 1] < k)
                i++;
        }
        insertNonFull(x->C[i + 1], k);
    }
}

void insert(BTree *tree, int k)
{
    if (tree->root == NULL)
    {
        tree->root = createNode(tree->t, true);
        tree->root->keys[0] = k;
        tree->root->n = 1;
    }
    else
    {
        if (tree->root->n == 2 * tree->t - 1)
        {
            BTreeNode *s = createNode(tree->t, false);
            s->C[0] = tree->root;
            splitChild(s, 0, tree->root);
            int i = 0;
            if (s->keys[0] < k)
                i++;
            insertNonFull(s->C[i], k);
            tree->root = s;
        }
        else
        {
            insertNonFull(tree->root, k);
        }
    }
}

void rangeSearch(BTreeNode *node, int low, int high)
{
    int i = 0;

    while (i < node->n && node->keys[i] < low)
        i++;
```

```c
    while (i < node->n && node->keys[i] <= high)
    {
        if (!node->leaf)
            rangeSearch(node->C[i], low, high);
        printf("%d ", node->keys[i]);
        i++;
    }

    if (!node->leaf)
        rangeSearch(node->C[i], low, high);
}

int main()
{
    BTree tree;
    tree.root = NULL;
    tree.t = MIN_DEGREE;

    int keys[] = {20, 5, 1, 10, 15, 30, 25, 40, 35, 50};
    int n = sizeof(keys) / sizeof(keys[0]);

    for (int i = 0; i < n; i++)
    {
        insert(&tree, keys[i]);
    }

    int low = 10, high = 35;
    printf("Keys in range [%d, %d]:\n", low, high);
    rangeSearch(tree.root, low, high);

    return 0;
}
```



```
C:\Users\dhruv\Desktop\dnk>c
Keys in range [10, 35]:
10 15 20 25 30 35
C:\Users\dhruv\Desktop\dnk\C
```