**# Tree :-**

***1. WRITE A PROGRAM TO IMPLEMENT BINARY TREE.***

***// Binary Tree***

***// This tree node having info -1 is considered as a termination point***

**#include <stdio.h>**

**#include <stdlib.h>**

**struct Tree\_node**

**{**

**int info;**

**struct Tree\_node \*left;**

**struct Tree\_node \*right;**

**};**

**typedef struct Tree\_node node;**

**node\* getNode();**

**node\* Create\_node(node\*);**

**void PreOrder(node\*);**

**void PostOrder(node\*);**

**void InOrder(node\*);**

**int main()**

**{**

**int ele,i,n;**

**node \*root;**

**printf("\nLet's create binay tree....!");**

**root=Create\_node(root);**

**printf("\nPre-Order display...\n");**

**PreOrder(root);**

**printf("\nPost-Order display...\n");**

**PostOrder(root);**

**printf("\nIn-Order display...\n");**

**InOrder(root);**

**return 0;**

**}**

**node\* getNode()**

**{**

**node\* newnode;**

**int info;**

**printf("\nEnter the info for the node: ");**

**fflush(stdin);**

**scanf("%d",&info);**

**if(info==-1)**

**{**

**newnode=NULL;**

**}**

**else**

**{**

**newnode=(node\*)malloc(sizeof(node));**

**newnode->info=info;**

**newnode->left=NULL;**

**newnode->right=NULL;**

**}**

**return newnode;**

**}**

**node\* Create\_node(node\* crrptr)**

**{**

**node\* newnode=getNode();**

**crrptr=newnode;**

**if(crrptr==NULL)**

**{**

**return crrptr;**

**}**

**printf("\nFor LEFT of %d node ",crrptr->info);**

**crrptr->left=Create\_node(crrptr->left);**

**printf("\nFor RIGHT of %d node ",crrptr->info);**

**crrptr->right=Create\_node(crrptr->right);**

**return crrptr;**

**}**

**void PreOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**printf("\t%d",crrptr->info);**

**PreOrder(crrptr->left);**

**PreOrder(crrptr->right);**

**}**

**}**

**void PostOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**PostOrder(crrptr->left);**

**PostOrder(crrptr->right);**

**printf("\t%d",crrptr->info);**

**}**

**}**

**void InOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**InOrder(crrptr->left);**

**printf("\t%d",crrptr->info);**

**InOrder(crrptr->right);**

**}**

**}**

***OUTPUT:-***

**PS C:\Users\Khushi\.vscode\Ds> cd Tree**

**PS C:\Users\Khushi\.vscode\Ds\Tree> gcc BinaryTree.cpp**

**PS C:\Users\Khushi\.vscode\Ds\Tree> ./a.exe**

**Let's create binay tree....!**

**Enter the info for the node: 5**

**For LEFT of 5 node**

**Enter the info for the node: 2**

**For LEFT of 2 node**

**Enter the info for the node: 9**

**For LEFT of 9 node**

**Enter the info for the node: 1**

**For LEFT of 1 node**

**Enter the info for the node: -1**

**For RIGHT of 1 node**

**Enter the info for the node: -1**

**For RIGHT of 9 node**

**Enter the info for the node: 8**

**For LEFT of 8 node**

**Enter the info for the node: -1**

**For RIGHT of 8 node**

**Enter the info for the node: -1**

**For RIGHT of 2 node**

**Enter the info for the node: 23**

**For LEFT of 23 node**

**Enter the info for the node: -1**

**For RIGHT of 23 node**

**Enter the info for the node: -1**

**For RIGHT of 5 node**

**Enter the info for the node: 31**

**For LEFT of 31 node**

**Enter the info for the node: 4**

**For LEFT of 4 node**

**Enter the info for the node: 2**

**For LEFT of 2 node**

**Enter the info for the node: -1**

**For RIGHT of 2 node**

**Enter the info for the node: -1**

**For RIGHT of 4 node**

**Enter the info for the node: 8**

**For LEFT of 8 node**

**Enter the info for the node: 1**

**For LEFT of 1 node**

**Enter the info for the node: -1**

**For RIGHT of 1 node**

**Enter the info for the node: -1**

**For RIGHT of 8 node**

**Enter the info for the node: 2**

**For LEFT of 2 node**

**Enter the info for the node: -1**

**For RIGHT of 2 node**

**Enter the info for the node: -1**

**For RIGHT of 31 node**

**Enter the info for the node: -1**

**Pre-Order display...**

**5 2 9 1 8 23 31 4 2 8 1 2**

**Post-Order display...**

**1 8 9 23 2 2 1 2 8 4 31 5**

**In-Order display...**

**1 9 8 2 23 5 2 4 1 8 2 31**

**PS C:\Users\Khushi\.vscode\Ds\Tree>**

***2. WRITE A PROGRAM TO IMPLEMENT BINARY SEARCH TREE.***

**// Write a 'C' program for inserting a node,**

**//preorder, inorder, postorder traversal,**

**//counting total number of nodes for binary search tree**

**# include <stdio.h>**

**#include <stdlib.h>**

**struct Binary\_Search\_Tree\_node**

**{**

**int info;**

**struct Binary\_Search\_Tree\_node \*left;**

**struct Binary\_Search\_Tree\_node \*right;**

**};**

**typedef struct Binary\_Search\_Tree\_node node;**

**node\* root;**

**node\* getnode();**

**void Create();**

**void PreOrder(node\*);**

**void InOrder(node\*);**

**void PostOrder(node\*);**

**int main()**

**{**

**int n;**

**root = NULL;**

**printf("Enter number of nodes: ");**

**scanf("%d",&n);**

**for(int i=0; i<n; i++)**

**{**

**Create();**

**}**

**printf("\nBinary Search Tree traversal...\n");**

**printf("Pre-Order traversal...\n");**

**PreOrder(root);**

**printf("\nIn-Order traversal...\n");**

**InOrder(root);**

**printf("\nPost-Order traversal...\n");**

**PostOrder(root);**

**return 0;**

**}**

**node\* getnode()**

**{**

**node\* newnode;**

**newnode = (node\*)malloc(sizeof(node));**

**printf("Enter the node info: ");**

**scanf("%d",&newnode->info);**

**newnode->left = NULL;**

**newnode->right = NULL;**

**return newnode;**

**}**

**void Create()**

**{**

**node\* newnode = getnode();**

**node\* crrptr=root;**

**if(root==NULL)**

**{**

**root = newnode;**

**}**

**else**

**{**

**while(crrptr!=NULL)**

**{**

**if(newnode->info > crrptr->info)**

**{**

**if(crrptr->right!=NULL)**

**{**

**crrptr = crrptr->right;**

**}**

**else**

**{**

**break;**

**}**

**}**

**else**

**{**

**if(crrptr->left!=NULL)**

**{**

**crrptr = crrptr->left;**

**}**

**else**

**{**

**break;**

**}**

**}**

**}**

**if(crrptr->info < newnode->info)**

**{**

**crrptr->right = newnode;**

**}**

**else**

**{**

**crrptr->left = newnode;**

**}**

**}**

**}**

**void PreOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**printf("%d\t",crrptr->info);**

**PreOrder(crrptr->left);**

**PreOrder(crrptr->right);**

**}**

**}**

**void InOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**InOrder(crrptr->left);**

**printf("%d\t",crrptr->info);**

**InOrder(crrptr->right);**

**}**

**}**

**void PostOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**PostOrder(crrptr->left);**

**PostOrder(crrptr->right);**

**printf("%d\t",crrptr->info);**

**}**

**}**

***OUTPUT:-***

**PS C:\Users\Khushi\.vscode\Ds> cd Tree**

**PS C:\Users\Khushi\.vscode\Ds\Tree> gcc BinarySearchTree.cpp**

**PS C:\Users\Khushi\.vscode\Ds\Tree> ./a.exe**

**Enter number of nodes: 6**

**Enter the node info: 2**

**Enter the node info: 7**

**Enter the node info: 4**

**Enter the node info: 1**

**Enter the node info: 9**

**Enter the node info: 12**

**Binary Search Tree traversal...**

**Pre-Order traversal...**

**2 1 7 4 9 12**

**In-Order traversal...**

**1 2 4 7 9 12**

**Post-Order traversal...**

**1 4 12 9 7 2**

**PS C:\Users\Khushi\.vscode\Ds\Tree>**

***3. WRITE A PROGRAM TO IMPLEMENT BINARY AVL TREE.***

**// Inplementation of AVL tree**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct AVL\_Tree\_Node**

**{**

**int key;**

**struct AVL\_Tree\_Node\* left;**

**struct AVL\_Tree\_Node\* right;**

**};**

**typedef struct AVL\_Tree\_Node node;**

**node\* getNode();**

**int height(node\*);**

**int max(int,int);**

**int balance\_factor(node\*);**

**node\* right\_Rotation(node\*);**

**node\* left\_Rotation(node\*);**

**node\* left\_right\_Rotation(node\*);**

**node\* right\_left\_Rotation(node\*);**

**node\* Insert\_node(node\*);**

**node\* balance\_Tree(node\*);**

**void PreOrder(node\*);**

**void InOrder(node\*);**

**void PostOrder(node\*);**

**node\* search\_node(node\*,int);**

**node\* search\_before\_node(node\*,int);**

**node\* delete\_node(node\*);**

**int Inorder\_successor(node\*);**

**int Inorder\_predeccessor(node\*);**

**int main()**

**{**

**int n;**

**node\* root = NULL;**

**// Insertion of node**

**printf("\nEnter the number of nodes: ");**

**scanf("%d",&n);**

**for(int i=0; i<n; i++)**

**{**

**root = Insert\_node(root);**

**}**

**printf("\nBinary Search Tree traversal...\n");**

**printf("Pre-Order traversal...\n");**

**PreOrder(root);**

**printf("\nPost-Order traversal...\n");**

**PostOrder(root);**

**printf("\nIn-Order traversal...\n");**

**InOrder(root);**

**// Search operation**

**int search;**

**node\* searchNode;**

**printf("\nEnter the node to search: ");**

**scanf("%d",&search);**

**searchNode = search\_node(root,search);**

**if(searchNode==NULL)**

**{**

**printf("\nNode not found...!");**

**}**

**else**

**{**

**printf("\nNode found...!\n%d",searchNode->key);**

**}**

**// Deletion operation**

**root = delete\_node(root);**

**printf("\nBinary Search Tree traversal...\n");**

**printf("Pre-Order traversal...\n");**

**PreOrder(root);**

**printf("\nPost-Order traversal...\n");**

**PostOrder(root);**

**printf("\nIn-Order traversal...\n");**

**InOrder(root);**

**return 0;**

**}**

**node\* getNode()**

**{**

**node\* newnode = (node\*) malloc(sizeof(node));**

**printf("\nEnter the info for node: ");**

**scanf("%d",&newnode->key);**

**newnode->left=NULL;**

**newnode->right=NULL;**

**return newnode;**

**}**

**int max (int a,int b)**

**{**

**if(a>b)**

**{**

**return a;**

**}**

**else**

**{**

**return b;**

**}**

**}**

**int height (node\* crrptr)**

**{**

**int height1=0,height2=0;**

**if(crrptr==NULL)**

**{**

**return -1;**

**}**

**else if(crrptr->left == NULL && crrptr->right==NULL)**

**{**

**return 0;**

**}**

**else**

**{**

**if(crrptr->left!=NULL)**

**{**

**height1 = height(crrptr->left)+1;**

**}**

**if(crrptr->right!=NULL)**

**{**

**height2 = height(crrptr->right)+1;**

**}**

**return max(height1,height2);**

**}**

**}**

**int balance\_factor(node\* crrptr)**

**{**

**return (height(crrptr->left)-height(crrptr->right));**

**}**

**node\* right\_Rotation(node\* x)**

**{**

**node\* y = x->left;**

**node\* z = y->left;**

**x->left = y->right;**

**y->right = x;**

**return y;**

**}**

**node\* left\_Rotation(node\* x)**

**{**

**node\* y = x->right;**

**node\* z = y->right;**

**x->right = y->left;**

**y->left = x;**

**return y;**

**}**

**node\* left\_right\_Rotation(node\* x)**

**{**

**x->left = left\_Rotation(x->left);**

**x = right\_Rotation(x);**

**return x;**

**}**

**node\* right\_left\_Rotation(node\* x)**

**{**

**x->right = right\_Rotation(x->right);**

**x = left\_Rotation(x);**

**return x;**

**}**

**node\* Insert\_node(node\* root)**

**{**

**node\* newnode = getNode();**

**node\* crrptr = root;**

**if(crrptr==NULL)**

**{**

**root = newnode;**

**}**

**else**

**{**

**while(crrptr!=NULL)**

**{**

**if(newnode->key > crrptr->key)**

**{**

**if(crrptr->right!=NULL)**

**{**

**crrptr = crrptr->right;**

**}**

**else**

**{**

**break;**

**}**

**}**

**else**

**{**

**if(crrptr->left!=NULL)**

**{**

**crrptr = crrptr->left;**

**}**

**else**

**{**

**break;**

**}**

**}**

**}**

**if(newnode->key > crrptr->key)**

**{**

**crrptr->right = newnode;**

**}**

**else**

**{**

**crrptr->left = newnode;**

**}**

**root = balance\_Tree(root);**

**}**

**return root;**

**}**

**node\* balance\_Tree(node\* crrptr)**

**{**

**if(crrptr==NULL)**

**{**

**return crrptr;**

**}**

**if(crrptr->left!=NULL)**

**{**

**crrptr->left = balance\_Tree(crrptr->left);**

**}**

**if(crrptr->right!=NULL)**

**{**

**crrptr->right = balance\_Tree(crrptr->right);**

**}**

**if(balance\_factor(crrptr) < -1)**

**{**

**if(balance\_factor(crrptr->right)<=0)**

**{**

**crrptr = left\_Rotation(crrptr);**

**}**

**else**

**{**

**crrptr = right\_left\_Rotation(crrptr);**

**}**

**}**

**else if(balance\_factor(crrptr)>1)**

**{**

**if(balance\_factor(crrptr->left)>=0)**

**{**

**crrptr = right\_Rotation(crrptr);**

**}**

**else**

**{**

**crrptr = left\_right\_Rotation(crrptr);**

**}**

**}**

**return crrptr;**

**}**

**void PreOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**printf("%d\t",crrptr->key);**

**PreOrder(crrptr->left);**

**PreOrder(crrptr->right);**

**}**

**}**

**void InOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**InOrder(crrptr->left);**

**printf("%d\t",crrptr->key);**

**InOrder(crrptr->right);**

**}**

**}**

**void PostOrder(node\* crrptr)**

**{**

**if(crrptr!=NULL)**

**{**

**PostOrder(crrptr->left);**

**PostOrder(crrptr->right);**

**printf("%d\t",crrptr->key);**

**}**

**}**

**node\* search\_node(node\* crrptr,int info)**

**{**

**node\* search = NULL;**

**if(crrptr==NULL)**

**{**

**return NULL;**

**}**

**else if(crrptr->key==info)**

**{**

**return crrptr;**

**}**

**else if(crrptr->key > info)**

**{**

**return search\_node(crrptr->left,info);**

**}**

**else**

**{**

**return search\_node(crrptr->right,info);**

**}**

**}**

**node\* search\_before\_node(node\* crrptr,int info)**

**{**

**node\* parent = NULL;**

**while(crrptr!=NULL)**

**{**

**if(crrptr->key == info)**

**{**

**return parent;**

**}**

**else if(crrptr->left!=NULL && crrptr->key > info)**

**{**

**parent = crrptr;**

**crrptr = crrptr->left;**

**}**

**else if(crrptr->right!=NULL && crrptr->key < info)**

**{**

**parent = crrptr;**

**crrptr = crrptr->right;**

**}**

**else**

**{**

**return NULL;**

**}**

**}**

**return NULL;**

**}**

**int  Inorder\_successor(node\* crrptr)**

**{**

**node\* parent = crrptr;**

**crrptr = crrptr->right;**

**if(crrptr->left==NULL)**

**{**

**parent->right = crrptr->right;**

**return crrptr->key;**

**}**

**while(crrptr->left!=NULL)**

**{**

**parent = crrptr;**

**crrptr = crrptr->left;**

**}**

**parent->left = NULL;**

**return crrptr->key;**

**}**

**int Inorder\_predeccessor(node\* crrptr)**

**{**

**node\* parent = crrptr;**

**crrptr = crrptr->left;**

**if(crrptr->right==NULL)**

**{**

**parent->left = crrptr->left;**

**return crrptr->key;**

**}**

**while(crrptr->right!=NULL)**

**{**

**parent = crrptr;**

**crrptr = crrptr->right;**

**}**

**parent->right=NULL;**

**return crrptr->key;**

**}**

**node\* delete\_node(node\* root)**

**{**

**int info;**

**printf("\nEnter the info to delete: ");**

**scanf("%d",&info);**

**node\* search = search\_node(root,info);**

**node\* parent = search\_before\_node(root,info);**

**if(search == NULL)**

**{**

**printf("\nNode not found...!");**

**}**

**else if(search->left==NULL && search->right==NULL)**

**{**

**printf("\nNode found...!");**

**if(parent->left == search)**

**{**

**parent->left = NULL;**

**}**

**else**

**{**

**parent->right = NULL;**

**}**

**}**

**else**

**{**

**printf("\nNode found...!");**

**if(balance\_factor(search)>=0)**

**{**

**search->key = Inorder\_predeccessor(search);**

**}**

**else**

**{**

**search->key = Inorder\_successor(search);**

**}**

**}**

**root = balance\_Tree(root);**

**return root;**

**}**

***OUTPUT:-***

**PS C:\Users\Khushi\.vscode\Ds> cd Tree**

**PS C:\Users\Khushi\.vscode\Ds\Tree> gcc AVL.cpp**

**PS C:\Users\Khushi\.vscode\Ds\Tree> ./a.exe**

**Enter the number of nodes: 6**

**Enter the info for node: 1**

**Enter the info for node: 2**

**Enter the info for node: 3**

**Enter the info for node: 8**

**Enter the info for node: 5**

**Enter the info for node: 0**

**Binary Search Tree traversal...**

**Pre-Order traversal...**

**2 1 0 5 3 8**

**Post-Order traversal...**

**0 1 3 8 5 2**

**In-Order traversal...**

**0 1 2 3 5 8**

**Enter the node to search: 7**

**Node not found...!**

**Enter the info to delete: 2**

**Node found...!**

**Binary Search Tree traversal...**

**Pre-Order traversal...**

**1 0 5 3 8**

**Post-Order traversal...**

**0 3 8 5 1**

**In-Order traversal...**

**0 1 3 5 8**

**PS C:\Users\Khushi\.vscode\Ds\Tree>**

***4. WRITE A PROGRAM TO IMPLEMENT B-TREE.***

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_KEYS 3**

**struct B\_Tree\_Node**

**{**

**int keys[MAX\_KEYS];**

**struct B\_Tree\_Node\* children[MAX\_KEYS + 1];**

**int num\_keys;**

**int is\_leaf;**

**};**

**typedef struct B\_Tree\_Node node;**

**node\* createNode(int);**

**void splitChild(node\*, int);**

**void insertNonFull(node\*, int);**

**void insert(node\*\*, int);**

**void inorderTraversal(node\*);**

**node\* search(node\*, int);**

**int main()**

**{**

**node\* root = createNode(1);**

**int num\_keys;**

**printf("Enter the number of keys to insert: ");**

**scanf("%d", &num\_keys);**

**printf("Enter the keys to insert:\n");**

**for (int i = 0; i < num\_keys; i++)**

**{**

**int key;**

**scanf("%d", &key);**

**insert(&root, key);**

**}**

**printf("Inorder Traversal: ");**

**inorderTraversal(root);**

**printf("\n");**

**int searchKey;**

**printf("Enter a key to search: ");**

**scanf("%d", &searchKey);**

**node\* result = search(root, searchKey);**

**if (result)**

**{**

**printf("Key %d found in the tree.\n", searchKey);**

**}**

**else**

**{**

**printf("Key %d not found in the tree.\n", searchKey);**

**}**

**return 0;**

**}**

**node\* createNode(int is\_leaf)**

**{**

**node\* newnode = (node\*)malloc(sizeof(node));**

**newnode->num\_keys = 0;**

**newnode->is\_leaf = is\_leaf;**

**for (int i = 0; i < MAX\_KEYS + 1; i++)**

**{**

**newnode->children[i] = NULL;**

**}**

**return newnode;**

**}**

**void splitChild(node\* parent, int index)**

**{**

**node\* child = parent->children[index];**

**node\* newChild = createNode(child->is\_leaf);**

**parent->num\_keys++;**

**for (int i = parent->num\_keys - 1; i > index; i--)**

**{**

**parent->keys[i] = parent->keys[i - 1];**

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

**}**

**parent->keys[index] = child->keys[MAX\_KEYS / 2];**

**parent->children[index + 1] = newChild;**

**for (int i = 0; i < MAX\_KEYS / 2; i++)**

**{**

**newChild->keys[i] = child->keys[i + (MAX\_KEYS / 2) + 1];**

**child->keys[i + (MAX\_KEYS / 2) + 1] = 0;**

**}**

**if (!child->is\_leaf)**

**{**

**for (int i = 0; i < MAX\_KEYS / 2 + 1; i++)**

**{**

**newChild->children[i] = child->children[i + (MAX\_KEYS / 2) + 1];**

**child->children[i + (MAX\_KEYS / 2) + 1] = NULL;**

**}**

**}**

**child->num\_keys = MAX\_KEYS / 2;**

**newChild->num\_keys = MAX\_KEYS / 2;**

**}**

**void insertNonFull(node\* node, int key)**

**{**

**int i = node->num\_keys - 1;**

**if (node->is\_leaf)**

**{**

**while (i >= 0 && key < node->keys[i])**

**{**

**node->keys[i + 1] = node->keys[i];**

**i--;**

**}**

**node->keys[i + 1] = key;**

**node->num\_keys++;**

**}**

**else**

**{**

**while (i >= 0 && key < node->keys[i])**

**{**

**i--;**

**}**

**i++;**

**if (node->children[i]->num\_keys == MAX\_KEYS)**

**{**

**splitChild(node, i);**

**if (key > node->keys[i])**

**{**

**i++;**

**}**

**}**

**insertNonFull(node->children[i], key);**

**}**

**}**

**void insert(node\*\* root, int key)**

**{**

**node\* rootnode = \*root;**

**if (rootnode->num\_keys == MAX\_KEYS)**

**{**

**node\* newRoot = createNode(0);**

**newRoot->children[0] = rootnode;**

**\*root = newRoot;**

**splitChild(newRoot, 0);**

**insertNonFull(newRoot, key);**

**}**

**else**

**{**

**insertNonFull(rootnode, key);**

**}**

**}**

**void inorderTraversal(node\* node)**

**{**

**if (node)**

**{**

**int i;**

**for (i = 0; i < node->num\_keys; i++)**

**{**

**inorderTraversal(node->children[i]);**

**printf("%d ", node->keys[i]);**

**}**

**inorderTraversal(node->children[i]);**

**}**

**}**

**node\* search(node\* node, int key)**

**{**

**int i = 0;**

**while (i < node->num\_keys && key > node->keys[i])**

**{**

**i++;**

**}**

**if (i < node->num\_keys && key == node->keys[i])**

**{**

**return node;**

**}**

**else if (node->is\_leaf)**

**{**

**return NULL;**

**}**

**else**

**{**

**return search(node->children[i], key);**

**}**

**}**

***OUTPUT:-***

**PS C:\Users\Khushi\.vscode\Ds\Tree> gcc B.cpp**

**PS C:\Users\Khushi\.vscode\Ds\Tree> ./a.exe**

**Enter the number of keys to insert: 8**

**Enter the keys to insert:**

**5**

**2**

**8**

**1**

**5**

**4**

**34**

**-2**

**Inorder Traversal: -2 1 2 4 5 5 8 34**

**Enter a key to search: 34**

**Key 34 found in the tree.**

**PS C:\Users\Khushi\.vscode\Ds\Tree>**

***5. WRITE A PROGRAM TO IMPLEMENT B+ TREE.***

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_KEYS 4**

**struct B\_Plus\_Tree\_Node**

**{**

**int keys[MAX\_KEYS];**

**struct B\_Plus\_Tree\_Node\* children[MAX\_KEYS + 1];**

**int num\_keys;**

**int is\_leaf;**

**struct B\_Plus\_Tree\_Node\* next;**

**};**

**typedef struct B\_Plus\_Tree\_Node node;**

**node\* createnode(int);**

**void insert(node\*\*, int);**

**void inorderTraversal(node\*);**

**int main()**

**{**

**node\* root = createnode(1);**

**int num\_keys;**

**printf("Enter the number of keys to insert: ");**

**scanf("%d", &num\_keys);**

**printf("Enter the keys to insert:\n");**

**for (int i = 0; i < num\_keys; i++)**

**{**

**int key;**

**scanf("%d", &key);**

**insert(&root, key);**

**}**

**printf("Inorder Traversal: ");**

**inorderTraversal(root);**

**printf("\n");**

**return 0;**

**}**

**node\* createnode(int is\_leaf)**

**{**

**node\* newnode = (node\*)malloc(sizeof(node));**

**newnode->num\_keys = 0;**

**newnode->is\_leaf = is\_leaf;**

**newnode->next = NULL;**

**for (int i = 0; i < MAX\_KEYS + 1; i++)**

**{**

**newnode->children[i] = NULL;**

**}**

**return newnode;**

**}**

**// Insertion function for a B+ tree**

**void insert(node\*\* root, int key)**

**{**

**node\* rootnode = \*root;**

**if (rootnode->is\_leaf)**

**{**

**// Leaf node insertion**

**int i = rootnode->num\_keys - 1;**

**while (i >= 0 && key < rootnode->keys[i])**

**{**

**rootnode->keys[i + 1] = rootnode->keys[i];**

**i--;**

**}**

**rootnode->keys[i + 1] = key;**

**rootnode->num\_keys++;**

**// Update the linked list**

**if (rootnode->next != NULL)**

**{**

**rootnode->next->keys[0] = key;**

**}**

**}**

**else**

**{**

**// Internal node insertion**

**int i = 0;**

**while (i < rootnode->num\_keys && key > rootnode->keys[i])**

**{**

**i++;**

**}**

**insert(&rootnode->children[i], key);**

**}**

**// Split the root if it's full**

**if (rootnode->num\_keys == MAX\_KEYS)**

**{**

**node\* newRoot = createnode(0);**

**newRoot->children[0] = rootnode;**

**\*root = newRoot;**

**}**

**}**

**void inorderTraversal(node\* node)**

**{**

**if (node)**

**{**

**int i;**

**for (i = 0; i < node->num\_keys; i++)**

**{**

**inorderTraversal(node->children[i]);**

**printf("%d ", node->keys[i]);**

**}**

**inorderTraversal(node->children[i]);**

**}**

**}**

***OUTPUT:-***

**PS C:\Users\Khushi\.vscode\Ds> cd Tree**

**PS C:\Users\Khushi\.vscode\Ds\Tree> gcc B+.cpp**

**PS C:\Users\Khushi\.vscode\Ds\Tree> ./a.exe**

**Enter the number of keys to insert: 4**

**Enter the keys to insert:**

**2**

**8**

**3**

**6**

**Inorder Traversal: 2 3 6 8**

**PS C:\Users\Khushi\.vscode\Ds\Tree>**