**DEPARTMENT OF COMPUTER APPLICATION**

**TKM COLLEGE OF ENGINEERING**

**KOLLAM – 691005**

Logo

Description automatically generated

**20MCA135 – DATA STRUCTURES LAB**

PRACTICAL RECORD BOOK

First Semester MCA

2021-2022

**Submitted by:**

NAME : KANNAN R R

ROLL NO : TKM21MCA-2026

**DEPARTMENT OF COMPUTER APPLICATION**

**TKM COLLEGE OF ENGINEERING**

**KOLLAM – 691005**

Logo

Description automatically generated

**Certificate**

This is a bonafide record of the work done by KANNAN R R in the First Semester in Data Structures Lab Course(TKM21MCA-2026) towards the partial fulfillment of the degree of Master of Computer Applications during the academic year 2021-2022.

Staff Member in-charge Examiner

………………………… ………………………..

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Program No :** | **Programs** | | **Page No :** |
|  | **LAB CYCLE 1** | |  |
| 1 | 1.1 |  |  |
| 2 | 1.2 |  |  |
| 3 | 1.3 |  |  |
| 4 | 1.4 |  |  |
| 5 | 1.5 |  |  |
|  |  | |  |
| 6 | 2.1 |  |  |
| 7 | 2.2 |  |  |
| 8 | 2.3 |  |  |
| 9 | 2.4 |  |  |
| 10 | 2.5 |  |  |
| 11 | 2.6 |  |  |
|  |  | |  |
| 12 | 3.1 |  |  |
|  |  | |  |
| 13 | 4.1 |  |  |
| 14 | 4.2 |  |  |
| 15 | 4.3 |  |  |
| 16 | 4.4 |  |  |
|  |  | |  |
| 17 | 5.1 |  |  |
| 18 | 5.2 |  |  |
| 19 | 5.3 |  |  |
| 20 | 5.4 |  |  |

**PROGRAM NO : 1**

**AIM :** Write a C program to implement Stack Operations

**CODE :**

#include<stdio.h>

#define n 5

int s[n],top=-1;

void push();

void pop();

void Top();

void display();

void main() {

int o,c=1;

while(c==1) {

printf("Enter any of the below option number\n1.push\n2.pop\n3.top\n4.display\n");

scanf("%d",&o);

switch(o) {

case 1 : push();

break;

case 2 : pop();

break;

case 3 : Top();

break;

case 4 : display();

break;

}

printf("Do you want to continue(0/1)\n");

scanf("%d",&c);

}

}

void push() {

int x;

printf("Enter an element to push\n");

scanf("%d",&x);

if(top==n-1) {

printf("\nOverflow\n");

} else {

top++;

s[top]=x;

}

}

void pop() {

if(top==-1) {

printf("\nUnderflow\n");

} else {

printf("Popped element is %d",s[top]);

top--;

}

}

void Top() {

if(top==-1) {

printf("\nUnderflow\n");

} else {

printf("Top element is %d",s[top]);

}

}

void display() {

if(top==-1) {

printf("\nUnderflow\n");

} else {

printf("Stack elements are\n");

for (int i=top; i>=0; i--) {

printf("%d\n",s[i]);

}

}

}

**OUTPUT :**

Text

Description automatically generated

**RESULT :**

**PROGRAM NO : 2**

**AIM :** Write a C program to implement Queue Operations

**CODE :**

#include<stdio.h>

#define n 5

int q[n],front=-1,rear=-1;

void insert();

void delete();

void peak();

void display();

void main() {

int o,c=1;

while(c==1) {

printf("Enter any of the below option number\n1.Insert\n2.Delete\n3.Peak\n4.Display\n");

scanf("%d",&o);

switch(o) {

case 1 : insert();

break;

case 2 : delete();

break;

case 3 : peak();

break;

case 4 : display();

break;

default : printf("Invalid entry");

}

printf("Do you want to continue(0/1)\n");

scanf("%d",&c);

}

}

void insert() {

int x;

printf("Enter an element to insert\n");

scanf("%d",&x);

if(rear==n-1) {

printf("\nOverflow\n");

//return;

} else if(front==-1 && rear==-1) {

front=rear=0;

} else {

rear++;

}

q[rear]=x;

}

void delete() {

if(front==-1 || front>rear) {

printf("\nUnderflow\n");

} else {

printf("Deleted element is %d",q[front]);

front++;

if(front>rear) {

front=rear=-1;

}

}

}

void peak() {

if(front==-1 || front>rear) {

printf("\nUnderflow\n");

} else {

printf("Peak element is %d",q[front]);

}

}

void display() {

if(front==-1 || front>rear) {

printf("\nUnderflow\n");

} else {

printf("Queue elements are\n");

for (int i=front;i<=rear;i++) {

printf("%d\n",q[i]);

}

}}

**OUTPUT :**

Text

Description automatically generated

**RESULT :**

**PROGRAM NO : 3**

**AIM :** Write a C program to implement Linked List Operations

**CODE :**

#include<stdio.h>

#include<stdlib.h>

struct node {

int data;

struct node \*next;};

struct node \*head,\*newnode,\*temp,\*prev;

void InsertBeg() {

int m;

printf("Enter value to insert");

scanf("%d",&m);

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

newnode->data=m;

newnode->next=head;

head=newnode;

}

void InsertBetween() {

int m,x;

printf("Enter value to insert");

scanf("%d",&m);

printf("Enter after which value to insert");

scanf("%d",&x);

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

newnode->data=m;

temp=head;

while(temp->next!=0) {

if(temp->data==x) {

break;}

temp=temp->next;}

newnode->next=temp->next;

temp->next=newnode;}

void InsertEnd() {

int m;

printf("Enter value to insert");

scanf("%d",&m);

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

newnode->data=m;

while(temp->next!=0) {

temp=temp->next;}

temp->next=newnode;

newnode->next=0;}

void DeleteBeg() { head=head->next;}

void DeleteBetween() {

int x;

printf("Enter the node data for position");

scanf("%d",&x);

temp=head;

while(temp->data!=x) {

temp=temp->next;}

temp->next=temp->next->next;}

void DeleteEnd() {

temp=head;

while(temp->next!=0) {

prev=temp;

temp=temp->next;}

prev->next=NULL;}

void display() {

temp=head;

while(temp!=0) {

printf("%d\n",temp->data);

temp=temp->next;}}

void addnode() {

int o=1,c=0,m;

printf("Enter value to insert");

scanf("%d",&m);

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

newnode->data=m;

newnode->next=0;

if(head==0) {

head=temp=newnode;

} else {

temp->next=newnode;

temp=newnode;}}

void main() {

int c=1,o,i,m;

head=0;

while(c==1) {

printf("Enter any of the below option number\n1.AddNode\n2.Insert-Begining\n3.Insert-End\n4.Insert-Between\n5.Delete-Begining\n6.Delete-End\n7.Delete-Between\n8.Dispaly\n");

scanf("%d",&o);

switch(o) {

case 1 : addnode();

break;

case 2 : InsertBeg();

break;

case 3 : InsertBetween();

break;

case 4 : InsertEnd();

break;

case 5 : DeleteEnd();

break;

case 6 : DeleteEnd();

break;

case 7 : DeleteEnd();

break;

case 8 : display();

break;

}

printf("Do you want to continue(0/1)\n");

scanf("%d",&c);

}

}

**OUTPUT :**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**RESULT :**

**PROGRAM NO : 4**

**AIM :** Write a C program to implement Doubly Linked List Operations .

**CODE :**

#include<stdio.h>

#include<stdlib.h>

struct node {

struct node \*prev;

int data;

struct node \*next;};

struct node \*head,\*newnode,\*temp,\*ptr;

void Beg() {

newnode->prev = head;

head=newnode;

newnode->next = head->prev;}

void Between() {

int x;

temp=head;

printf("Enter the node data for position");

scanf("%d",&x);

while(temp->data!=x) {

temp=temp->next;}

newnode->next=temp->next;

temp->next=newnode;

temp->next->prev=newnode;}

void End() {

temp=head;

while(temp->next!=NULL) {

temp=temp->next;}

temp->next=newnode;

newnode->prev=temp;}

void DelBeg() {head=head->next;}

void DelBetween() {

int x;

temp=head;

printf("Enter the node data for position");

scanf("%d",&x);

while(temp->data!=x) {

temp=temp->next;}

printf("asa");

ptr=temp;

temp->next->next->prev=ptr;

temp->next=ptr->next->next;}

void DelEnd() {

temp=head;

while(temp->next!=NULL) {

temp=temp->next;}

temp->next=newnode;

newnode->prev=temp;}

void display() {

temp=head;

int count=0;

while(temp!=0) {

printf("%d\n",temp->data);

count++;

temp=temp->next;}}

void addnode() {

int o=1,c=0;

head=0;

while(o==1) {

if(head==0) {

head=temp=newnode;

} else {

temp->next=newnode;

newnode->prev=temp;

temp=newnode;}

printf("Do you want to continue insertion (0/1) ");

scanf("%d",&o);}}

void CreateNode() {

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

printf("Enter the data ");

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

newnode->next=NULL;

newnode->prev=NULL;}

void main() {

int c=1,o,i,m;

while(c==1) {

printf("Enter any of the below option number\n1.AddNode\n2.Insert-Begining\n3.Insert-End\n4.Insert-Between\n5.Delete-Begining\n6.Delete-End\n7.Delete-Between\n8.Dispaly\n");

scanf("%d",&o);

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

switch(o) {

case 1 : CreateNode();

addnode();

break;

case 2 : CreateNode();

Beg(m);

break;

case 3 : CreateNode();

End(m);

break;

case 4 : CreateNode();

Between();

break;

case 5 : DelBeg();

break;

case 6 : DelEnd();

break;

case 7 : DelBetween();

break;

case 8 : display();

break;

}

printf("Do you want to continue(0/1)\n");

scanf("%d",&c);

}

}

**PROGRAM NO : 5**

**AIM :** Write a C program to implement BitString Operations .

**CODE :**

#include<stdio.h>

int x,y,z,a[10],b[10],b1[10],b2[10],b2c[10],u[10],d=0,o=1;

void Union() {

printf("\nA Union B : ");

for (int i=0;i<10;i++) {

printf("%d",b1[i] | b2[i]);

}

}

void Intersection() {

printf("\nA Intersection B : ");

for (int i=0;i<10;i++) {

printf("%d",b1[i] \* b2[i]);

}

}

void Diff() {

for (int i=0;i<10;i++) {

if(b2[i]==0) {

b2c[i]=1;

} else {

b2c[i]=0;

}

}

printf("\nA - B : ");

for (int i=0;i<10;i++) {

printf("%d",b1[i] \* b2c[i]);

}

}

void main() {

printf("Enter the number of elements in set U\n");

scanf("%d",&x);

printf("Enter the elements in set U\n");

for (int i=0;i<x;i++) {

scanf("%d",&u[i]);

}

printf("Enter the number of elements in set A\n");

scanf("%d",&y);

printf("Enter the elements in set A\n");

for (int i=0;i<y;i++) {

scanf("%d",&a[i]);

}

printf("Enter the number of elements in set B\n");

scanf("%d",&z);

printf("Enter the elements in set B\n");

for (int i=0;i<z;i++) {

scanf("%d",&b[i]);

}

for (int i=0;i<x;i++) {

for (int j=0;j<y;j++) {

if(a[j]==u[i]) {

b1[i]=1;

break;

} else {

b1[i]=0;

}

}

}

for (int i=0;i<x;i++) {

for (int j=0;j<z;j++) {

if(b[j]==u[i]) {

b2[i]=1;

break;

} else {

b2[i]=0;

}

}

}

printf("\nA : ");

for (int i=0;i<10;i++) {

printf("%d",b1[i]);

}

printf("\nB : ");

for (int i=0;i<10;i++) {

printf("%d",b2[i]);

}

while(o==1) {

printf("\nEnter any option \n1.Union\n2.Intersection\n3.Difference\n");

scanf("%d",&d);

switch(d) {

case 1 : Union();

printf("\n");

break;

case 2 : Intersection();

printf("\n");

break;

case 3 : Diff();

printf("\n");

break;

}

printf("Continue(0/1)");

scanf("%d",&o);

}

}

**OUTPUT :**

Text

Description automatically generated

**RESULT :**

**PROGRAM NO : 6**

**AIM :** Write a C program to implement Doubly Linked List Operations .

**CODE :**

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \*left, \*right;};

void inorder(struct node \*r) {

if(r!=NULL) {

inorder(r->left);

printf("%d ",r->data);

inorder(r->right);}}

void min(struct node \*r) {

struct node \*pre;

while(pre!= NULL && r->left!=NULL) {

pre=r;

r=r->left;

}

printf("\nMin element is : %d",r->data);}

void max(struct node \*r) {

struct node \*pre;

while(pre!=NULL && r->right!=NULL) {

pre=r;

r=r->right;}

printf("\nMax element is : %d",r->data);}

void insuc(struct node \*r,int x) {

struct node \*temp=r,\*l;

struct node \*n;

while(temp->right!=NULL || temp->left!=NULL) {

if(x==temp->data) {

n=temp;

break;

} else if(x> temp->data) {

temp=temp->right;

} else {

l=temp;

temp=temp->left;}}

if(temp->right!=NULL) {

temp=temp->right;

while(temp->left!=NULL) {

temp=temp->left; }

printf("\nInorder succesor of %d is %d",x,temp->data);

} else {

printf("\nInorder succesor of %d is %d",x,l->data);}}

void search(struct node \*r,int x) {

int f=0;

struct node \*pre=r;

while(pre->right!=NULL || pre->left!=NULL) {

if(x==r->data) {

pre=r;

f=1;

break;

} else if(x> r->data) {

pre=r;

r=r->right;

} else {

pre=r;

r=r->left;}}

if(f==1) {

printf("Found");

} else {

printf("Not Found");

}}

void main() {

int n, i, item,x;

struct node \*new, \*temp, \*root;

printf("Enter the number of elements\n");

scanf("%d", &n);

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

new = (struct node \*) malloc(sizeof(struct node));

new->right = NULL;

new->left = NULL;

printf("Enter the data \n");

scanf("%d", &item);

new->data = item;

if (i == 0) {

root = new;

} else {

temp = root;

while (1) {

if (item > temp->data) {

if (temp->right != NULL) {

temp = temp->right;

} else {

temp->right = new;

break;

}

} else {

if (temp->left != NULL) {

temp = temp->left;

} else {

temp->left = new;

break;

}}

}

}}

inorder(root);

printf("\nEnter the data to search \n");

scanf("%d",&x);

search(root,x);

min(root);

max(root);

int v=0;

printf("Enter an element to get inorder succesor\n");

scanf("%d",&v);

insuc(root,v);}

**OUTPUT:**

Text

Description automatically generated

**RESULT :**

**PROGRAM NO : 7**

**AIM :** Write a C program to implement Red Black Tree Operations .

**CODE :**

#include <stdio.h>

#include <stdlib.h>

enum nodeColor {

RED,

BLACK

};

struct rbNode {

int data, color;

struct rbNode \*link[2];

}

;

struct rbNode \*root = NULL;

// Create a red-black tree

struct rbNode \*createNode(int data) {

struct rbNode \*newnode;

newnode = (struct rbNode \*)malloc(sizeof(struct rbNode));

newnode->data = data;

newnode->color = RED;

newnode->link[0] = newnode->link[1] = NULL;

return newnode;

}

// Insert an node

void insertion(int data) {

struct rbNode \*stack[98], \*ptr, \*newnode, \*xPtr, \*yPtr;

int dir[98], ht = 0, index;

ptr = root;

if (!root) {

root = createNode(data);

return;

}

stack[ht] = root;

dir[ht++] = 0;

while (ptr != NULL) {

if (ptr->data == data) {

printf("Duplicates Not Allowed!!\n");

return;

}

index = (data - ptr->data) > 0 ? 1 : 0;

stack[ht] = ptr;

ptr = ptr->link[index];

dir[ht++] = index;

}

stack[ht - 1]->link[index] = newnode = createNode(data);

while ((ht >= 3) && (stack[ht - 1]->color == RED)) {

if (dir[ht - 2] == 0) {

yPtr = stack[ht - 2]->link[1];

if (yPtr != NULL && yPtr->color == RED) {

stack[ht - 2]->color = RED;

stack[ht - 1]->color = yPtr->color = BLACK;

ht = ht - 2;

} else {

if (dir[ht - 1] == 0) {

yPtr = stack[ht - 1];

} else {

xPtr = stack[ht - 1];

yPtr = xPtr->link[1];

xPtr->link[1] = yPtr->link[0];

yPtr->link[0] = xPtr;

stack[ht - 2]->link[0] = yPtr;

}

xPtr = stack[ht - 2];

xPtr->color = RED;

yPtr->color = BLACK;

xPtr->link[0] = yPtr->link[1];

yPtr->link[1] = xPtr;

if (xPtr == root) {

root = yPtr;

} else {

stack[ht - 3]->link[dir[ht - 3]] = yPtr;

}

break;

}

} else {

yPtr = stack[ht - 2]->link[0];

if ((yPtr != NULL) && (yPtr->color == RED)) {

stack[ht - 2]->color = RED;

stack[ht - 1]->color = yPtr->color = BLACK;

ht = ht - 2;

} else {

if (dir[ht - 1] == 1) {

yPtr = stack[ht - 1];

} else {

xPtr = stack[ht - 1];

yPtr = xPtr->link[0];

xPtr->link[0] = yPtr->link[1];

yPtr->link[1] = xPtr;

stack[ht - 2]->link[1] = yPtr;

}

xPtr = stack[ht - 2];

yPtr->color = BLACK;

xPtr->color = RED;

xPtr->link[1] = yPtr->link[0];

yPtr->link[0] = xPtr;

if (xPtr == root) {

root = yPtr;

} else {

stack[ht - 3]->link[dir[ht - 3]] = yPtr;

}

break;

}

}

}

root->color = BLACK;

}

// Delete a node

void deletion(int data) {

struct rbNode \*stack[98], \*ptr, \*xPtr, \*yPtr;

struct rbNode \*pPtr, \*qPtr, \*rPtr;

int dir[98], ht = 0, diff, i;

enum nodeColor color;

if (!root) {

printf("Tree not available\n");

return;

}

ptr = root;

while (ptr != NULL) {

if ((data - ptr->data) == 0)

break;

diff = (data - ptr->data) > 0 ? 1 : 0;

stack[ht] = ptr;

dir[ht++] = diff;

ptr = ptr->link[diff];

}

if (ptr->link[1] == NULL) {

if ((ptr == root) && (ptr->link[0] == NULL)) {

free(ptr);

root = NULL;

} else if (ptr == root) {

root = ptr->link[0];

free(ptr);

} else {

stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];

}

} else {

xPtr = ptr->link[1];

if (xPtr->link[0] == NULL) {

xPtr->link[0] = ptr->link[0];

color = xPtr->color;

xPtr->color = ptr->color;

ptr->color = color;

if (ptr == root) {

root = xPtr;

} else {

stack[ht - 1]->link[dir[ht - 1]] = xPtr;

}

dir[ht] = 1;

stack[ht++] = xPtr;

} else {

i = ht++;

while (1) {

dir[ht] = 0;

stack[ht++] = xPtr;

yPtr = xPtr->link[0];

if (!yPtr->link[0])

break;

xPtr = yPtr;

}

dir[i] = 1;

stack[i] = yPtr;

if (i > 0)

stack[i - 1]->link[dir[i - 1]] = yPtr;

yPtr->link[0] = ptr->link[0];

xPtr->link[0] = yPtr->link[1];

yPtr->link[1] = ptr->link[1];

if (ptr == root) {

root = yPtr;

}

color = yPtr->color;

yPtr->color = ptr->color;

ptr->color = color;

}

}

if (ht < 1)

return;

if (ptr->color == BLACK) {

while (1) {

pPtr = stack[ht - 1]->link[dir[ht - 1]];

if (pPtr && pPtr->color == RED) {

pPtr->color = BLACK;

break;

}

if (ht < 2)

break;

if (dir[ht - 2] == 0) {

rPtr = stack[ht - 1]->link[1];

if (!rPtr)

break;

if (rPtr->color == RED) {

stack[ht - 1]->color = RED;

rPtr->color = BLACK;

stack[ht - 1]->link[1] = rPtr->link[0];

rPtr->link[0] = stack[ht - 1];

if (stack[ht - 1] == root) {

root = rPtr;

} else {

stack[ht - 2]->link[dir[ht - 2]] = rPtr;

}

dir[ht] = 0;

stack[ht] = stack[ht - 1];

stack[ht - 1] = rPtr;

ht++;

rPtr = stack[ht - 1]->link[1];

}

if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&

(!rPtr->link[1] || rPtr->link[1]->color == BLACK)) {

rPtr->color = RED;

} else {

if (!rPtr->link[1] || rPtr->link[1]->color == BLACK) {

qPtr = rPtr->link[0];

rPtr->color = RED;

qPtr->color = BLACK;

rPtr->link[0] = qPtr->link[1];

qPtr->link[1] = rPtr;

rPtr = stack[ht - 1]->link[1] = qPtr;

}

rPtr->color = stack[ht - 1]->color;

stack[ht - 1]->color = BLACK;

rPtr->link[1]->color = BLACK;

stack[ht - 1]->link[1] = rPtr->link[0];

rPtr->link[0] = stack[ht - 1];

if (stack[ht - 1] == root) {

root = rPtr;

} else {

stack[ht - 2]->link[dir[ht - 2]] = rPtr;

}

break;

}

} else {

rPtr = stack[ht - 1]->link[0];

if (!rPtr)

break;

if (rPtr->color == RED) {

stack[ht - 1]->color = RED;

rPtr->color = BLACK;

stack[ht - 1]->link[0] = rPtr->link[1];

rPtr->link[1] = stack[ht - 1];

if (stack[ht - 1] == root) {

root = rPtr;

} else {

stack[ht - 2]->link[dir[ht - 2]] = rPtr;

}

dir[ht] = 1;

stack[ht] = stack[ht - 1];

stack[ht - 1] = rPtr;

ht++;

rPtr = stack[ht - 1]->link[0];

}

if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&

(!rPtr->link[1] || rPtr->link[1]->color == BLACK)) {

rPtr->color = RED;

} else {

if (!rPtr->link[0] || rPtr->link[0]->color == BLACK) {

qPtr = rPtr->link[1];

rPtr->color = RED;

qPtr->color = BLACK;

rPtr->link[1] = qPtr->link[0];

qPtr->link[0] = rPtr;

rPtr = stack[ht - 1]->link[0] = qPtr;

}

rPtr->color = stack[ht - 1]->color;

stack[ht - 1]->color = BLACK;

rPtr->link[0]->color = BLACK;

stack[ht - 1]->link[0] = rPtr->link[1];

rPtr->link[1] = stack[ht - 1];

if (stack[ht - 1] == root) {

root = rPtr;

} else {

stack[ht - 2]->link[dir[ht - 2]] = rPtr;

}

break;

}

}

ht--;

}

}

}

// Print the inorder traversal of the tree

void inorderTraversal(struct rbNode \*node) {

if (node) {

inorderTraversal(node->link[0]);

printf("%d ", node->data);

inorderTraversal(node->link[1]);

}

return;

}

int main() {

int ch, data;

while (1) {

printf("1. Insertion\t2. Deletion\n");

printf("3. Traverse\t4. Exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter the element to insert:");

scanf("%d", &data);

insertion(data);

break;

case 2:

printf("Enter the element to delete:");

scanf("%d", &data);

deletion(data);

break;

case 3:

inorderTraversal(root);

printf("\n");

break;

case 4:

exit(0);

default:

printf("Not available\n");

break;

}

printf("\n");

}

return 0;

}

**PROGRAM NO : 8**

**AIM :** Write a C program to implement B-Tree Operations .

**CODE :**

#include <stdio.h>

#include <stdlib.h>

#define MAX 3

#define MIN 2

struct BTreeNode {

int val[MAX + 1], count;

struct BTreeNode \*link[MAX + 1];

}

;

struct BTreeNode \*root;

struct BTreeNode \*createNode(int val, struct BTreeNode \*child) {

struct BTreeNode \*newNode;

newNode = (struct BTreeNode \*)malloc(sizeof(struct BTreeNode));

newNode->val[1] = val;

newNode->count = 1;

newNode->link[0] = root;

newNode->link[1] = child;

return newNode;

}

void insertNode(int val, int pos, struct BTreeNode \*node,

struct BTreeNode \*child) {

int j = node->count;

while (j > pos) {

node->val[j + 1] = node->val[j];

node->link[j + 1] = node->link[j];

j--;

}

node->val[j + 1] = val;

node->link[j + 1] = child;

node->count++;

}

void splitNode(int val, int \*pval, int pos, struct BTreeNode \*node,

struct BTreeNode \*child, struct BTreeNode \*\*newNode) {

int median, j;

if (pos > MIN)

median = MIN + 1; else

median = MIN;

\*newNode = (struct BTreeNode \*)malloc(sizeof(struct BTreeNode));

j = median + 1;

while (j <= MAX) {

(\*newNode)->val[j - median] = node->val[j];

(\*newNode)->link[j - median] = node->link[j];

j++;

}

node->count = median;

(\*newNode)->count = MAX - median;

if (pos <= MIN) {

insertNode(val, pos, node, child);

} else {

insertNode(val, pos - median, \*newNode, child);

}

\*pval = node->val[node->count];

(\*newNode)->link[0] = node->link[node->count];

node->count--;

}

int setValue(int val, int \*pval,

struct BTreeNode \*node, struct BTreeNode \*\*child) {

int pos;

if (!node) {

\*pval = val;

\*child = NULL;

return 1;

}

if (val < node->val[1]) {

pos = 0;

} else {

for (pos = node->count;

(val < node->val[pos] && pos > 1); pos--)

;

if (val == node->val[pos]) {

printf("Duplicates are not permitted\n");

return 0;

}

}

if (setValue(val, pval, node->link[pos], child)) {

if (node->count < MAX) {

insertNode(\*pval, pos, node, \*child);

} else {

splitNode(\*pval, pval, pos, node, \*child, child);

return 1;

}

}

return 0;

}

void insert(int val) {

int flag, i;

struct BTreeNode \*child;

flag = setValue(val, &i, root, &child);

if (flag)

root = createNode(i, child);

}

void search(int val, int \*pos, struct BTreeNode \*myNode) {

if (!myNode) {

return;

}

if (val < myNode->val[1]) {

\*pos = 0;

} else {

for (\*pos = myNode->count;

(val < myNode->val[\*pos] && \*pos > 1); (\*pos)--)

;

if (val == myNode->val[\*pos]) {

printf("%d is found", val);

return;

}

}

search(val, pos, myNode->link[\*pos]);

return;

}

void traversal(struct BTreeNode \*myNode) {

int i;

if (myNode) {

for (i = 0; i < myNode->count; i++) {

traversal(myNode->link[i]);

printf("%d ", myNode->val[i + 1]);

}

traversal(myNode->link[i]);

}

}

int main() {

int val, ch;

insert(8);

insert(9);

insert(10);

insert(11);

insert(15);

insert(16);

insert(17);

insert(18);

insert(20);

insert(23);

traversal(root);

printf("\n");

search(11, &ch, root);

}

**OUTPUT :**

**RESULT :**