Assignment 10

Q)

Code:

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

struct node \*head=NULL;

struct node \*tail=NULL;

void takeinput()

{

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

printf("\nEnter the data");

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

newnode->prev=NULL;

newnode->next=NULL;

if(head==NULL)

{

head=tail=newnode;

newnode->prev=head;

newnode->next=head;

}

else

{

tail->next=newnode;

newnode->prev=tail;

head->prev=newnode;

newnode->next=head;

tail=newnode;

}

}

int length()

{

int l=0;

struct node \*temp=head;

do{

l++;

temp=temp->next;

}while(temp!=head);

return l;

}

void insert()

{

int l=length();

int pos;

printf("\nEnter the position at which you want to insert the node (0th position to the %dth position of the linked list)",l);

scanf("%d",&pos);

if(pos>l)

{

printf("\nInvalid loaction");

return;

}

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

printf("\nEnter the data");

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

newnode->prev=NULL;

newnode->next=NULL;

if(pos==0)

{

head->prev->next=newnode;

newnode->prev=head->prev;

head->prev=newnode;

newnode->next=head;

head=newnode;

}

/\*else if(pos==l)

{

tail->next=newnode;

newnode->prev=tail;

head->prev=newnode;

newnode->next=head;

tail=newnode;

}\*/

else

{

int ctr=0;

struct node \*temp=head;

while(ctr<pos-1)

{

temp=temp->next;

ctr++;

}

temp->next->prev=newnode;

newnode->next=temp->next;

temp->next=newnode;

newnode->prev=temp;

}

}

void delete()

{

if(head==NULL)

{

printf("\n Pls enter atleast one element");

return;

}

int l=length();

int pos;

printf("\nEnter the position at which you want to delete the node (0th position to the %dth position of the linked list)",l-1);

scanf("%d",&pos);

if(pos>=l)

{

printf("\nInvalid loaction");

return;

}

struct node \*temp=head;

if(pos==0)

{

temp=head;

head->prev->next=head->next;

head->next->prev=head->prev;

head=head->next;

temp->next=NULL;

temp->prev=NULL;

free(temp);

}

/\*else if(pos==l-1)

{

tail->next=newnode;

newnode->prev=tail;

head->prev=newnode;

newnode->next=head;

tail=newnode;

}\*/

else

{

int ctr=0;

while(ctr<pos)

{

temp=temp->next;

ctr++;

}

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

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

temp->next=NULL;

temp->prev=NULL;

free(temp);

}

}

void display()

{

if(head==NULL)

{

printf("\n Pls enter atleast one element");

return;

}

struct node \*temp=head;

printf("\n");

do{

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

temp=temp->next;

}while(temp!=head);

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

}

int main()

{

int choice;

while(1)

{

printf("\nMain Menu");

printf("\n1.Creation of the List");

printf("\n2.Insert");

printf("\n3.Delete");

printf("\n4.Display");

printf("\n5.Exit");

printf("\nNote: You can use this option only before deletion of the last node of the circular doubly linked list.");

printf("\nEnter your choice");

scanf("%d",&choice);

switch (choice)

{

case 1:takeinput();

display();

break;

case 2:insert();

display();

break;

case 3:delete();

display();

break;

case 4:display();

break;

case 5:exit(0);

break;

default:printf("\n Invalid Entry");

}

}

}

Output:

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice1

Enter the data1

1 1

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice1

Enter the data2

1 2 1

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice1

Enter the data3

1 2 3 1

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice2

Enter the position at which you want to insert the node (0th position to the 3th position of the linked list)0

Enter the data4

4 1 2 3 4

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice2

Enter the position at which you want to insert the node (0th position to the 4th position of the linked list)3

Enter the data5

4 1 2 5 3 4

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice2

Enter the position at which you want to insert the node (0th position to the 5th position of the linked list)5

Enter the data6

4 1 2 5 3 6 4

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice3

Enter the position at which you want to delete the node (0th position to the 5th position of the linked list)0

1 2 5 3 6 1

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice3

Enter the position at which you want to delete the node (0th position to the 4th position of the linked list)2

1 2 3 6 1

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice3

Enter the position at which you want to delete the node (0th position to the 3th position of the linked list)3

1 2 3 1

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice4

1 2 3 1

Main Menu

1.Creation of the List

2.Insert

3.Delete

4.Display

5.Exit

Note: You can use this option only before deletion of the last node of the circular doubly linked list.

Enter your choice5

**...Program finished with exit code 0**

**Press ENTER to exit console.**

Q)

Code:

#include<stdio.h>

#include<stdlib.h>

struct node{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item){

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

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void traversetree(struct node \*root){

if (root != NULL){

traversetree(root->left);

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

traversetree(root->right);

}

}

struct node\* search(struct node\* root, int key){

if (root == NULL || root->key == key)

return root;

if (root->key < key)

return search(root->right, key);

return search(root->left, key);

}

struct node\* insert(struct node\* node, int key){

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

return node;

}

int main(){

struct node \*root = NULL;

root = insert(root, 23);

insert(root, 15);

insert(root, 12);

insert(root, 17);

insert(root, 32);

insert(root, 29);

insert(root, 45);

printf("The tree is :\n");

traversetree(root);

printf("\nSearching for 12 in this tree ");

if(search(root , 12))

printf("\nelement found");

else

printf("\nelement not found");

return 0;

}

Output:

The tree is :

12      15      17      23      29      32      45

Searching for 12 in this tree

element found

**...Program finished with exit code 0**

**Press ENTER to exit console.**

Q)

Code:

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*lchild;

struct node \*rchild;

}\*root;

void find(int item,struct node \*\*par,struct node \*\*loc)

{

struct node \*ptr,\*ptrsave;

if(root==NULL)

{

\*loc=NULL;

\*par=NULL;

return;

}

if(item==root->info)

{

\*loc=root;

\*par=NULL;

return;

}

if(item<root->info)

ptr=root->lchild;

else

ptr=root->rchild;

ptrsave=root;

while(ptr!=NULL)

{

if(item==ptr->info)

{ \*loc=ptr;

\*par=ptrsave;

return;

}

ptrsave=ptr;

if(item<ptr->info)

ptr=ptr->lchild;

else

ptr=ptr->rchild;

}

\*loc=NULL;

\*par=ptrsave;

}

void insert(int item)

{

struct node \*tmp,\*parent,\*location;

find(item,&parent,&location);

if(location!=NULL)

{

printf("Item already present");

return;

}

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

tmp->info=item;

tmp->lchild=NULL;

tmp->rchild=NULL;

if(parent==NULL)

root=tmp;

else

if(item<parent->info)

parent->lchild=tmp;

else

parent->rchild=tmp;

}

void case\_a(struct node \*par,struct node \*loc )

{

if(par==NULL)

root=NULL;

else

if(loc==par->lchild)

par->lchild=NULL;

else

par->rchild=NULL;

}

void case\_b(struct node \*par,struct node \*loc)

{

struct node \*child;

if(loc->lchild!=NULL)

child=loc->lchild;

else

child=loc->rchild;

if(par==NULL )

root=child;

else

if( loc==par->lchild)

par->lchild=child;

else

par->rchild=child;

}

void case\_c(struct node \*par,struct node \*loc)

{

struct node \*ptr,\*ptrsave,\*suc,\*parsuc;

ptrsave=loc;

ptr=loc->rchild;

while(ptr->lchild!=NULL)

{

ptrsave=ptr;

ptr=ptr->lchild;

}

suc=ptr;

parsuc=ptrsave;

if(suc->lchild==NULL && suc->rchild==NULL)

case\_a(parsuc,suc);

else

case\_b(parsuc,suc);

if(par==NULL)

root=suc;

else

if(loc==par->lchild)

par->lchild=suc;

else

par->rchild=suc;

suc->lchild=loc->lchild;

suc->rchild=loc->rchild;

}

int del(int item)

{

struct node \*parent,\*location;

if(root==NULL)

{

printf("Tree empty");

return 0;

}

find(item,&parent,&location);

if(location==NULL)

{

printf("Item not present in tree");

return 0;

}

if(location->lchild==NULL && location->rchild==NULL)

case\_a(parent,location);

if(location->lchild!=NULL && location->rchild==NULL)

case\_b(parent,location);

if(location->lchild==NULL && location->rchild!=NULL)

case\_b(parent,location);

if(location->lchild!=NULL && location->rchild!=NULL)

case\_c(parent,location);

free(location);

}/\*End of del()\*/

int preorder(struct node \*ptr)

{

if(root==NULL)

{

printf("Tree is empty");

return 0;

}

if(ptr!=NULL)

{

printf("%d ",ptr->info);

preorder(ptr->lchild);

preorder(ptr->rchild);

}

}

void inorder(struct node \*ptr)

{

if(root==NULL)

{

printf("Tree is empty");

return;

}

if(ptr!=NULL)

{

inorder(ptr->lchild);

printf("%d ",ptr->info);

inorder(ptr->rchild);

}

}

void postorder(struct node \*ptr)

{

if(root==NULL)

{

printf("Tree is empty");

return;

}

if(ptr!=NULL)

{

postorder(ptr->lchild);

postorder(ptr->rchild);

printf("%d ",ptr->info);

}

}

int main()

{

int choice,num;

root=NULL;

while(1)

{

printf("\n");

printf("1.Insert\n");

printf("2.Delete\n");

printf("3.Inorder Traversal\n");

printf("4.Preorder Traversal\n");

printf("5.Postorder Traversal\n");

printf("6.Display\n");

printf("7.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("Enter the number to be inserted : ");

scanf("%d",&num);

insert(num);

break;

case 2:printf("Enter the number to be deleted : ");

scanf("%d",&num);

del(num);

break;

case 3:inorder(root);

break;

case 4:preorder(root);

break;

case 5:postorder(root);

break;

case 6:exit(0);

break;

default:

printf("Wrong choice\n");

}

}

}

Output:

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 1

Enter the number to be inserted : 34

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 1

Enter the number to be inserted : 23

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 1

Enter the number to be inserted : 23

Item already present

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 1

Enter the number to be inserted : 65

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 1

Enter the number to be inserted : 87

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 1

Enter the number to be inserted : 34

Item already present

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 1

Enter the number to be inserted : 83

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 1

Enter the number to be inserted : 92

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 2

Enter the number to be deleted : 56

Item not present in tree

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 2

Enter the number to be deleted : 92

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 2

Enter the number to be deleted : 23

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 3

34  65  83  87

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 4

34  65  87  83

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 5

83  87  65  34

1.Insert

2.Delete

3.Inorder Traversal

4.Preorder Traversal

5.Postorder Traversal

6.Quit

Enter your choice : 6

**...Program finished with exit code 0**

**Press ENTER to exit console.**

#include <stdio.h>

#include <stdlib.h>

struct BSTNode

{

int data;

struct BSTNode \*lc;

struct BSTNode \*rc;

};

struct BSTNode \*Insert(struct BSTNode \*root)

{

struct BSTNode \*n=(struct BSTNode\*)malloc(sizeof(struct BSTNode));

printf("\nEnter the data to be inserted: ");

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

n->lc=NULL;

n->rc=NULL;

struct BSTNode \*temp=root;

struct BSTNode \*temp2=NULL;

while(temp!=NULL)

{

temp2=temp;

if(temp->data<n->data)

{

temp=temp->rc;

}

else

{

temp=temp->lc;

}

}

if(temp2==NULL)

{

root=n;

}

else if(temp2->data<n->data)

{

temp2->rc=n;

}

else

{

temp2->lc=n;

}

return root;

}

void Inorder\_Traversal(struct BSTNode \*root)

{

if(root==NULL)

{

return;

}

Inorder\_Traversal(root->lc);

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

Inorder\_Traversal(root->rc);

return;

}

void Preorder\_Traversal(struct BSTNode \*root)

{

if(root==NULL)

{

return;

}

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

Preorder\_Traversal(root->lc);

Preorder\_Traversal(root->rc);

return;

}

void Postorder\_Traversal(struct BSTNode \*root)

{

if(root==NULL)

{

return;

}

Postorder\_Traversal(root->lc);

Postorder\_Traversal(root->rc);

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

return;

}

struct BSTNode \*search(struct BSTNode \*root,int item)

{

if(root==NULL)

{

printf("\nThe BST is empty.");

return root;

}

else

{

struct BSTNode \*temp=root;

while(temp!=NULL)

{

if(temp->data==item)

{

return temp;

}

else if(temp->data<item)

{

temp=temp->rc;

}

else

{

temp=temp->lc;

}

}

if(temp==NULL)

{

return temp;

}

}

}

struct BSTNode \*Delete(struct BSTNode \*root,int item)

{

if(root==NULL)

{

return root;

}

if(item<root->data)

{

root->lc=Delete(root->lc,item);

}

else if(item>root->data)

{

/ root->rc=Delete(root->rc,item);

}

else

{

if(root->lc==NULL && root->rc==NULL)

{

root=NULL;

return root;

}

else if(root->lc!=NULL && root->rc==NULL)

{

struct BSTNode \*r=root;

root=root->lc;

r->lc=NULL;

free(r);

return root;

}

else if(root->lc==NULL && root->rc!=NULL)

{

struct BSTNode \*r=root;

root=root->rc;

r->rc=NULL;

free(r);

return root;

}

else

{

struct BSTNode \*r=root;

struct BSTNode \*t=root->rc;

root=t;

while(t->lc!=NULL)

{

t=t->lc;

}

t->lc=r->lc;

r->lc=NULL;

r->rc=NULL;

free(r);

return root;

}

}

}

int main()

{

struct BSTNode \*root=NULL;

printf("\nEnter the elements of the binary search tree.");

root=Insert(root);

root=Insert(root);

root=Insert(root);

root=Insert(root);

root=Insert(root);

root=Insert(root);

root=Insert(root);

root=Insert(root);

root=Insert(root);

printf("\nInorder traversal of the Binary Search Tree is: ");

Inorder\_Traversal(root);

printf("\nPreorder traversal of the Binary Search Tree is: ");

Preorder\_Traversal(root);

printf("\nPostorder traversal of the Binary Search Tree is: ");

Postorder\_Traversal(root);

printf("\nEnter the item to be searched:");

int item;

scanf("%d",&item);

struct BSTNode \*i=search(root,item);

if(!i)

{

printf("\nThe item is not found in BST.");

}

else

{

printf("\nThe item is found in BST.");

}

int item1,item2,item3,item4;

printf("\nEnter the item to be deleted: ");

scanf("%d",&item1);

root=Delete(root,item1);

printf("\nInorder traversal of the Binary Search Tree is: ");

Inorder\_Traversal(root);

printf("\nPreorder traversal of the Binary Search Tree is: ");

Preorder\_Traversal(root);

printf("\nPostorder traversal of the Binary Search Tree is: ");

Postorder\_Traversal(root);

printf("\nEnter the item to be deleted: ");

scanf("%d",&item2);

root=Delete(root,item2);

printf("\nInorder traversal of the Binary Search Tree is: ");

Inorder\_Traversal(root);

printf("\nPreorder traversal of the Binary Search Tree is: ");

Preorder\_Traversal(root);

printf("\nPostorder traversal of the Binary Search Tree is: ");

Postorder\_Traversal(root);

printf("\nEnter the item to be deleted: ");

scanf("%d",&item3);

root=Delete(root,item3);

printf("\nInorder traversal of the Binary Search Tree is: ");

Inorder\_Traversal(root);

printf("\nPreorder traversal of the Binary Search Tree is: ");

Preorder\_Traversal(root);

printf("\nPostorder traversal of the Binary Search Tree is: ");

Postorder\_Traversal(root);

printf("\nEnter the item to be deleted: ");

scanf("%d",&item4);

root=Delete(root,item4);

printf("\nInorder traversal of the Binary Search Tree is: ");

Inorder\_Traversal(root);

printf("\nPreorder traversal of the Binary Search Tree is: ");

Preorder\_Traversal(root);

printf("\nPostorder traversal of the Binary Search Tree is: ");

Postorder\_Traversal(root);

return 0;

}