**LAB CYCLE 1**

**Submitted by**

**SIJI JOSE**

**MCA 135**

**GIT LINK DS**

https://github.com/siji1999/Data-Structures.git

<https://github.com/siji1999/Data-Structures.git>

**PROGRAM 1 : MERGE TWO SORTED ARRAY**

#include<stdio.h>

void main()

{

int n, m, i, j, k, c[40], a[20], b[20];

printf ("Enter limit for A:");

scanf ("%d", &n);

printf ("\nEnter limit for B:");

scanf ("%d", &m);

printf ("Enter elements for A in sorted order:-\n");

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

{

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

}

printf ("Enter elements for B in sorted order:-\n");

for (j = 0; j < m; j++)

{

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

}

i = j = k = 0;

while (i < n && j < m)

{

if (a[i] < b[j])

{

c[k++] = a[i++];

}

else if (a[i] > b[j])

{

c[k++] = b[j++];

}

else

{

c[k++] = b[j++];

i++;

j++;

}}

if (i < n)

{

for (int t = 0; t < n; t++)

{

c[k++] = a[i++];

}}

if (j < m)

{

for (int t = 0; t < m; t++)

{

c[k++] = b[j++];

}}

printf ("\n");

for (k = 0; k < (m + n); k++)

{

printf ("\t \n %d ", c[k]);

}

printf("\n");

}

**PROGRAM 2 : CIRCULAR QUEUE**

#include <stdio.h>

#define SIZE 5

int items[SIZE];int front = -1, rear = -1;

// Check if the queue is fullint isFull() {

if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;

return 0;

}

// Check if the queue is emptyint isEmpty() {

if (front == -1) return 1;

return 0;

}

// Adding an elementvoid enQueue(int element) {

if (isFull())

printf("\n Queue is full!! \n");

else {

if (front == -1) front = 0;

rear = (rear + 1) % SIZE;

items[rear] = element;

printf("\n Inserted -> %d", element);

}}

// Removing an elementint

deQueue() {

int element;

if (isEmpty()) {

printf("\n Queue is empty !! \n");

return (-1);

} else {

element = items[front];

if (front == rear) {

front = -1;

rear = -1;

}

// Q has only one element, so we reset the

// queue after dequeing it. ?

else {

front = (front + 1) % SIZE;

}

printf("\n Deleted element -> %d \n", element);

return (element);

}

}

// Display the queuevoid display() {

int i;

if (isEmpty())

printf(" \n Empty Queue\n");

else {

printf("\n Front -> %d ", front);

printf("\n Items -> ");

for (i = front; i != rear; i = (i + 1) % SIZE) {

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

}

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

printf("\n Rear -> %d \n", rear);

}}

int main() {

// Fails because front = -1

deQueue();

enQueue(1);

enQueue(2);

enQueue(3);

enQueue(4);

enQueue(5);

// Fails to enqueue because front == 0 && rear == SIZE - 1

enQueue(6);

display();

deQueue();

display();

enQueue(7);

display();

// Fails to enqueue because front == rear + 1

enQueue(8);

return 0;

}

**PROGRAM 3 : SINGLY LINKED STACK**

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

struct node

{

int val;

struct node \*next;

};

struct node \*head;

void main ()

{

int choice=0;

while(choice != 4)

{

printf("\n\nChose one from the below options...\n");

printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

printf("\n Enter your choice = ");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("Exiting....");

break;

}

default:

{

printf("Please Enter valid choice ");

} };

}}

void push ()

{

int val;

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

if(ptr == NULL)

{

printf("not able to push the element");

}

else

{

printf("Enter the value = ");

scanf("%d",&val);

if(head==NULL)

{

ptr->val = val;

ptr -> next = NULL;

head=ptr;

}

else

{

ptr->val = val;

ptr->next = head;

head=ptr;

}

printf("Item pushed");

}}

void pop()

{

int item;

struct node \*ptr;

if (head == NULL)

{

printf("Underflow");

}

else

{

item = head->val;

ptr = head;

head = head->next;

free(ptr);

printf("Item popped");

} }

void display()

{

int i;

struct node \*ptr;

ptr=head;

if(ptr == NULL)

{

printf("Stack is empty\n");

}

else

{

printf("Printing Stack elements \n");

while(ptr!=NULL)

{

printf("%d\n",ptr->val);

ptr = ptr->next;

}}}

**PROGRAM 4 : DOUBLY LINKED LIST**

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 9)

{

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

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n 5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

printf("\nEnter your choice? = ");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

insertion\_beginning();

break;

case 2:

insertion\_last();

break;

case 3:

insertion\_specified();

break;

case 4:

deletion\_beginning();

break;

case 5:

deletion\_last();

break;

case 6:

deletion\_specified();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

exit(0);

break;

default:

printf("Please enter valid choice..");

} } }

void insertion\_beginning()

{

struct node \*ptr;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter Item value = ");

scanf("%d",&item);

if(head==NULL)

{

ptr->next = NULL;

ptr->prev=NULL;

ptr->data=item;

head=ptr;

}

else

{

ptr->data=item;

ptr->prev=NULL;

ptr->next = head;

head->prev=ptr;

head=ptr;

}

printf("\nNode inserted\n");

} }

void insertion\_last()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value = ");

scanf("%d",&item);

ptr->data=item;

if(head == NULL)

{

ptr->next = NULL;

ptr->prev = NULL;

head = ptr;

}

else

{

temp = head;

while(temp->next!=NULL)

{

temp = temp->next;

}

temp->next = ptr;

ptr ->prev=temp;

ptr->next = NULL;

} }

printf("\nnode inserted\n");

}

void insertion\_specified()

{

struct node \*ptr,\*temp;

int item,loc,i;

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

if(ptr == NULL)

{

printf("\n OVERFLOW");

}

else

{

temp=head;

printf("Enter the location = ");

scanf("%d",&loc);

for(i=0;i<loc;i++)

{

temp = temp->next;

if(temp == NULL)

{

printf("\n There are less than %d elements", loc);

return;

} }

printf("Enter value = ");

scanf("%d",&item);

ptr->data = item;

ptr->next = temp->next;

ptr -> prev = temp;

temp->next = ptr;

temp->next->prev=ptr;

printf("\nnode inserted\n");

} }

void deletion\_beginning()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

head = head -> next;

head -> prev = NULL;

free(ptr);

printf("\nnode deleted\n");

} }

void deletion\_last()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

if(ptr->next != NULL)

{

ptr = ptr -> next;

}

ptr -> prev -> next = NULL;

free(ptr);

printf("\nnode deleted\n");

} }

void deletion\_specified()

{

struct node \*ptr, \*temp;

int val;

printf("\n Enter the data after which the node is to be deleted : ");

scanf("%d", &val);

ptr = head;

while(ptr -> data != val)

ptr = ptr -> next;

if(ptr -> next == NULL)

{

printf("\nCan't delete\n");

}

else if(ptr -> next -> next == NULL)

{

ptr ->next = NULL;

}

else

{

temp = ptr -> next;

ptr -> next = temp -> next;

temp -> next -> prev = ptr;

free(temp);

printf("\nnode deleted\n");

} }

void display()

{

struct node \*ptr;

printf("\n printing values...\n");

ptr = head;

while(ptr != NULL)

{

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

ptr=ptr->next;

} }

void search()

{

struct node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

printf("\nitem found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("\nItem not found\n");

} } }

**PROGRAM 5 : BINARY SEARCH TREE**

#include<stdio.h>

#include<stdlib.h>

struct node{

struct node \*left;

struct node \*right;

int data;

};

struct node \*root;

struct node\* newNode(int value){

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

newnode->data = value;

newnode->left=NULL;

newnode->right=NULL;

return newnode;

}

struct node\* insert(struct node\* root,int value) {

if(root == NULL){

return newNode(value);

}

else if(value == root->data){

printf("Same data can't be stored");

}

else if(value>root->data){

root->right = insert(root->right,value);

}

else if(value<root->data){

root->left = insert(root->left,value);

}

return root;

}

// Inorder traversal

void inorderTraversal(struct node\* root) {

if (root == NULL) return;

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

// Preorder traversal

void preorderTraversal(struct node\* root) {

if (root == NULL) return;

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

preorderTraversal(root->left);

preorderTraversal(root->right);

}

// Postorder traversal

void postorderTraversal(struct node\* root) {

if (root == NULL) return;

postorderTraversal(root->left);

postorderTraversal(root->right);

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

}

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

if (root == NULL)

printf("\nNot FOUND!\n");

else if (root->data == key)

printf("\nFOUND!\n");

else{

if (root->data < key)

return search(root->right, key);

return search(root->left, key);

}

}

struct node\* minValueNode(struct node\* node){

struct node\* current = node;

/\* loop down to find the leftmost leaf \*/

while (current && current->left != NULL)

current = current->left;

return current;

}

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

if (root == NULL)

return root;

if (key < root->data)

root->left = deleteNode(root->left, key);

else if (key > root->data)

root->right = deleteNode(root->right, key);

else {

// node with only one child or no child

if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

// node with two children:

// Get the inorder successor

// (smallest in the right subtree)

struct node\* temp = minValueNode(root->right);

// Copy the inorder

// successor's content to this node

root->data = temp->data;

// Delete the inorder successor

root->right = deleteNode(root->right, temp->data);

}

return root;

}

int main(){

int opt;

int value,searchv,key;

do{

printf("\n1)Create Root Node \n2)Insert Node\n3)Search\n");

printf("4)inorderTraversal \n5)preorderTraversal \n6)postorderTraversal \n7)Delete \n8)Quiet \n");

printf("Choose Option :: ");

scanf("%d",&opt);

switch(opt){

case 1:

printf("\nEnter a number : ");

scanf("%d",&value);

root = newNode(value);

break;

case 2:

printf("\nEnter a number : ");

scanf("%d",&value);

root = insert(root,value);

break;

case 3:

printf("\nEnter a number : ");

scanf("%d",&searchv);

search(root,searchv);

break;

case 4:

printf("\n..................................\n");

inorderTraversal(root);

printf("\n..................................\n");

break;

case 5:

printf("\n..................................\n");

preorderTraversal(root);

printf("\n..................................\n");

break;

case 6:

printf("\n..................................\n");

postorderTraversal(root);

printf("\n..................................\n");

break;

case 7:

printf("\nEnter a number to be deleted : ");

scanf("%d",&key);

deleteNode(root,key);

break;

defualt:

printf("Invalid option!");

}

}while(opt!=8);

return 0;

}