**LAB CYCLE 1**

**SUBMITTED TO : SUBMITTED BY:**

**FOUSIA MISS NANDANA ANIL**

**ROLL NO : MCA107**

**S1 MCA**

**1.MERGE TWO SORTED ARRAYS**

#include <stdio.h>

int n;

int a[11] ;

int b[10];

void merging(int low, int mid, int high) {

int l1, l2, i;

for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {

if(a[l1] <= a[l2])

b[i] = a[l1++];

else

b[i] = a[l2++];

}

while(l1 <= mid)

b[i++] = a[l1++];

while(l2 <= high)

b[i++] = a[l2++];

for(i = low; i <= high; i++)

a[i] = b[i];

}

void sort(int low, int high) {

int mid;

if(low < high) {

mid = (low + high) / 2;

sort(low, mid);

sort(mid+1, high);

merging(low, mid, high);

} else {

return;

}

}

int main() {

int i;

printf("number of elements=");

scanf("%d",&n);

printf("enter the array\n");

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

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

printf("List before sorting\n");

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

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

sort(0, n-1 );

printf("\nList after sorting\n");

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

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

printf("\n");

}

**2.CIRCULAR QUEUE**

#include <stdio.h>

#define SIZE 5

int items[SIZE];

int front = -1, rear = -1;

int isFull() {

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

  return 0;

}

int isEmpty() {

  if (front == -1) return 1;

  return 0;

}

void 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);

  }

}

int deQueue() {

  int element;

  if (isEmpty()) {

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

    return (-1);

  } else {

    element = items[front];

    if (front == rear) {

      front = -1;

      rear = -1;

    }

    else {

      front = (front + 1) % SIZE;

    }

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

    return (element);

  }

}

void 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() {

  deQueue();

  enQueue(1);

  enQueue(2);

  enQueue(3);

  enQueue(4);

  enQueue(5);

  enQueue(6);

  display();

  deQueue();

  display();

  enQueue(7);

  display();

  enQueue(8);

  return 0;

}

**3.SINGLY LINKED STACK**

#include<stdio.h>

#include<stdlib.h>

struct NODE

{

int data;

struct NODE \*link;

}\*header,\*top,\*ptr;

int c=0;

void main()

{

void push();

void pop();

void status();

void display();

int ch;

do

{

printf("\n CHOICES \n");

printf("\n 1.PUSH \n 2.POP \n 3.STATUS \n 4.EXIT \n");

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

scanf("%d",&ch);

switch(ch)

{

case 1 : push();

display();

break;

case 2 : pop();

display();

break;

case 3 : status();

display();

break;

case 4 : exit(0);

break;

default : printf("wrong choice\n");

break;

}

}

while(ch!=4);

}

void push()

{

struct NODE \*newnode;

newnode = malloc(sizeof(struct NODE));

printf("enter data : ");

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

newnode->link=top;

top=newnode;

header=top;

c=c+1;

}

void pop()

{

if(top == NULL)

{

printf("stack underflow\n");

}

else

{

ptr = top->link;

header ->link=ptr;

printf("\n popped element : %d",top->data);

free(top);

top=ptr;

}

c=c+1;

}

void status()

{

}

void display()

{

if(top ==NULL)

{

printf("stack is empty");

}

else

{

ptr=top;

printf("\n stack is :\n");

while(ptr!=NULL)

{

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

ptr=ptr->link;

}

}

}

**4.DOUBLY LINKED LIST**

#include<stdio.h>

#include<stdlib.h>

struct NODE

{

int data;

struct NODE \* Rlink;

struct NODE \* Llink;

}\*header,\*newnode,\*firstnode,\*ptr,\*ptr1,\*ptr2,\*insertfnode,\*insertenode,\*insertnodeany;

void main()

{

int ch;

void create();

void traverse();

void insertfront();

void insertend();

void insertany();

void deletefront();

void deleteend();

void deleteany();

create();

traverse();

do

{

printf("\n \n\t\t CHOICES \n");

printf("\n\t1.INSERTION AT FRONT\n\t2.INSERTION AT ANY POSITION\n\t3.INSERTION AT END\n\t4.DELETION AT FRONT\n\t5.DELETION AT ANY POSITION\n\t6.DELETION AT END\n\t7.EXIT\n");

printf("Enter your choice : ");

scanf("%d",&ch);

switch(ch)

{

case 1:insertfront();

traverse();

break;

case 2:insertany();

traverse();

break;

case 3:insertend();

traverse();

break;

case 4:deletefront();

traverse();

break;

case 5:deleteany();

traverse();

break;

case 6:deleteend();

traverse();

break;

case 7:exit(0);

break;

default:printf("Invalid choice");

break;

}

}

while(ch!=7);

}

void create()

{

if ( header == NULL)

{

ptr=header;

firstnode=malloc(sizeof(struct NODE));

printf("\n\t Enter data : ");

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

firstnode->Rlink=NULL;

header=firstnode;

firstnode->Llink=header;

ptr=firstnode;

}

while(1)

{

newnode=malloc(sizeof(struct NODE));

printf("\n\t Enter data : ");

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

if ( newnode -> data == 0)

{

break;

}

newnode->Rlink=NULL;

ptr->Rlink=newnode;

newnode->Llink=ptr;

ptr=newnode;

}

}

void traverse()

{

ptr=header;

printf("\n\t THE LINKEDLIST IS \n");

printf("DATA\t ADDRESS\t LLINK\t RLINK");

do

{

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

printf("%p\t",&ptr->data);

printf("%p\t",ptr->Llink);

printf("%p\t",ptr->Rlink);

ptr=ptr->Rlink;

}

while(ptr != NULL);

}

void insertfront()

{

int data;

insertfnode=malloc(sizeof(struct NODE));

if ( insertfnode == NULL )

{

printf("\nmemory underflow");

}

else

{

printf("\n\t INSERTION AT FRONT\n");

printf("Enter data : ");

scanf("%d",&data);

ptr=header;

insertfnode->data=data;

insertfnode->Llink=NULL;

insertfnode->Rlink=ptr;

ptr->Llink=insertfnode;

header=insertfnode;

}}

void insertend()

{

int m;

insertenode=malloc(sizeof(struct NODE));

if ( insertenode == NULL )

{

printf("Memory underflow\n");

}

else

{

printf("\n\t INSERTION AT END\n");

printf("Enter data : ");

scanf("%d",&m);

ptr=header;

while ( ptr ->Rlink != NULL )

ptr=ptr->Rlink;

ptr->Rlink=insertenode;

insertenode->data=m;

insertenode->Llink=ptr;

insertenode->Rlink=NULL;

}

}

void insertany()

{

int y,key;

insertnodeany=malloc(sizeof(struct NODE));

if ( insertnodeany == NULL )

{

printf("\n Memory underflow");

}

else

{

printf("\n\t INSERTION AT ANY POSITION\n");

ptr=header;

printf("Enter data and key : ");

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

while (( ptr -> data != key ) && ( ptr -> Rlink != NULL ))

ptr=ptr->Rlink;

if (ptr == NULL)

printf("\nkey not found");

else

{

ptr1=ptr->Rlink;

ptr->Rlink=insertnodeany;

insertnodeany->data=y;

insertnodeany->Llink=ptr;

insertnodeany->Rlink=ptr1;

ptr1->Llink=insertnodeany;

}

}

}

void deletefront()

{

ptr = header;

if ( ptr == NULL )

{

printf("List is empty ");

}

else

{

printf("\n\t DELETION AT FRONT \n");

ptr1=ptr->Rlink;

ptr1->Llink=NULL;

header=ptr1;

free(ptr);

}

}

void deleteend()

{

ptr=header;

if ( ptr == NULL )

printf("\n list is empty ");

else

{

printf("\n\t DELETION AT END \n");

while ( ptr -> Rlink != NULL )

ptr=ptr->Rlink;

ptr1=ptr->Llink;

ptr1->Rlink=NULL;

free(ptr);

}

}

void deleteany()

{

int key;

ptr=header;

if ( ptr == NULL )

{

printf("\n list is empty");

}

else

printf("\n\t DELETION AT ANY POSTION \n ");

printf("\n Enter data : ");

scanf("%d",&key);

while (( ptr -> data != key ) && ( ptr ->Rlink != NULL ))

ptr=ptr->Rlink;

if ( ptr == NULL )

printf("\n key not found ");

else

{

ptr1=ptr->Llink;

ptr2=ptr->Rlink;

ptr1->Rlink=ptr2;

ptr2->Llink=ptr1;

free(ptr);

}}

**5.BINARY SEARCH TREE**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct node{

int data;

struct node \*left;

struct node \*right;

};

struct node \*root= NULL;

struct node\* createNode(int data){

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

newNode->data= data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

void insert(int data) {

struct node \*newNode = createNode(data);

if(root == NULL){

root = newNode;

return;

}

else {

struct node \*current = root, \*parent = NULL;

while(true) {

parent = current;

if(data < current->data) {

current = current->left;

if(current == NULL) {

parent->left = newNode;

return;

}

}

else {

current = current->right;

if(current == NULL) {

parent->right = newNode;

return;

}

}

}

}

}

struct node\* minNode(struct node \*root) {

if (root->left != NULL)

return minNode(root->left);

else

return root;

}

struct node\* deleteNode(struct node \*node, int value) {

if(node == NULL){

return NULL;

}

else {

if(value < node->data)

node->left = deleteNode(node->left, value);

else if(value > node->data)

node->right = deleteNode(node->right, value);

else {

if(node->left == NULL && node->right == NULL)

node = NULL;

else if(node->left == NULL) {

node = node->right;

}

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

node = node->left;

}

else {

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

node->data = temp->data;

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

}

}

return node;

}

}

void inorderTraversal(struct node \*node) {

if(root == NULL){

printf("Tree is empty\n");

return;

}

else {

if(node->left!= NULL)

inorderTraversal(node->left);

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

if(node->right!= NULL)

inorderTraversal(node->right);

}

}

int main()

{

insert(50);

insert(30);

insert(70);

insert(60);

insert(10);

insert(90);

printf("Binary search tree after insertion: \n");

inorderTraversal(root);

struct node \*deletedNode = NULL;

deletedNode = deleteNode(root, 90);

printf("\nBinary search tree after deleting node 90: \n");

inorderTraversal(root);

deletedNode = deleteNode(root, 30);

printf("\nBinary search tree after deleting node 30: \n");

inorderTraversal(root);

deletedNode = deleteNode(root, 50);

printf("\nBinary search tree after deleting node 50: \n");

inorderTraversal(root);

return 0;

}

**LINK TO GITHUB REPOSITORY:**

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