**CMR TECHNICAL CAMPUS**

**Kandlakoya(V), Medchal Road, Hyderabad – 501 401**

***An UGC Autonomous Institute***

**Accredited by NBA and NAAC with A Grade**

**Approved by AICTE, New Delhi and Affiliated to JNTU, Hyderabad**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**(DATA SCIENCE)**

****

**DATA STRUCTURE USING C**

**LAB MANUAL**

**(R20)**

DATA STRUCTURES LAB

Course Objectives:

1. It covers various concepts of C programming language
2. It introduces searching and sorting algorithms
3. It provides an understanding of data structures such as stacks and queues.

Course Outcomes:

1. Ability to develop C programs for computing and real-life applications using basic elements like control statements, arrays, functions, pointers and strings, and data structures like stacks, queues and linked lists.

2. Ability to implement searching and sorting algorithms.

LIST OF EXPERIMENTS

1. Write a program that uses functions to perform the following operations on singly linked list.

i) Creation

ii) Insertion

iii) Deletion

iv) Traversal

2. Write a program that uses functions to perform the following operations on doubly linked list.

i) Creation

ii) Insertion

iii) Deletion

iv) Traversal

3. Write a program that uses functions to perform the following operations on circular linked list.

i) Creation

ii) Insertion

iii) Deletion

iv) Traversal

4. Write a program that implement stack (its operations) using

i) Arrays

ii) Pointers

5. Write a program that implement Queue (its operations) using

i) Arrays

ii) Pointers

6. Write a program that implements the following sorting methods to sort a given list of integers in ascending order

i) Bubble sort

ii) Selection sort

iii) Insertion sort

7. Write a program that uses both recursive and non recursive functions to perform the following searching operations for a Key value in a given list of integers:

i) Linear search

ii) Binary search

8. Write a program to implement the tree traversal methods.

9. Write a program to implement the graph traversal methods.

1. Write a program that uses functions to perform the following operations on singly linked

list.

i) Creation

ii) Insertion

iii) Deletion

iv) Traversal

**Program:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void beginsert ();

void lastinsert ();

void randominsert();

void begin\_delete();

void last\_delete();

void random\_delete();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 9)

{

printf("\n\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\n5.Delete from last\n6.Delete node after specified location\n7.Search for an element\n8.Show\n9.Exit\n");

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

scanf("%d",&choice);

switch(choice)

{

case 1:

beginsert();

break;

case 2:

lastinsert();

break;

case 3:

randominsert();

break;

case 4:

begin\_delete();

break;

case 5:

last\_delete();

break;

case 6:

random\_delete();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

exit(0);

break;

default:

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

}

}

}

void beginsert()

{

struct node \*ptr;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value\n");

scanf("%d",&item);

ptr->data = item;

ptr->next = head;

head = ptr;

printf("\nNode inserted");

}

}

void lastinsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value?\n");

scanf("%d",&item);

ptr->data = item;

if(head == NULL)

{

ptr -> next = NULL;

head = ptr;

printf("\nNode inserted");

}

else

{

temp = head;

while (temp -> next != NULL)

{

temp = temp -> next;

}

temp->next = ptr;

ptr->next = NULL;

printf("\nNode inserted");

}

}

}

void randominsert()

{

int i,loc,item;

struct node \*ptr, \*temp;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter element value");

scanf("%d",&item);

ptr->data = item;

printf("\nEnter the location after which you want to insert ");

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

temp=head;

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

{

temp = temp->next;

if(temp == NULL)

{

printf("\ncan't insert\n");

return;

}

}

ptr ->next = temp ->next;

temp ->next = ptr;

printf("\nNode inserted");

}

}

void begin\_delete()

{

struct node \*ptr;

if(head == NULL)

{

printf("\nList is empty\n");

}

else

{

ptr = head;

head = ptr->next;

free(ptr);

printf("\nNode deleted from the begining ...\n");

}

}

void last\_delete()

{

struct node \*ptr,\*ptr1;

if(head == NULL)

{

printf("\nlist is empty");

}

else if(head -> next == NULL)

{

head = NULL;

free(head);

printf("\nOnly node of the list deleted ...\n");

}

else

{

ptr = head;

while(ptr->next != NULL)

{

ptr1 = ptr;

ptr = ptr ->next;

}

ptr1->next = NULL;

free(ptr);

printf("\nDeleted Node from the last ...\n");

}

}

void random\_delete()

{

struct node \*ptr,\*ptr1;

int loc,i;

printf("\n Enter the location of the node after which you want to perform deletion \n");

scanf("%d",&loc);

ptr=head;

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

{

ptr1 = ptr;

ptr = ptr->next;

if(ptr == NULL)

{

printf("\nCan't delete");

return;

}

}

ptr1 ->next = ptr ->next;

free(ptr);

printf("\nDeleted node %d ",loc+1);

}

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("item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("Item not found\n");

}

}

}

void display()

{

struct node \*ptr;

ptr = head;

if(ptr == NULL)

{

printf("Nothing to print");

}

else

{

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

while (ptr!=NULL)

{

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

ptr = ptr -> next;

}

}

}

OUTPUT:

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

1

Enter value

12

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

2

Enter value?

56

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

1

Enter value

34

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

3

Enter element value32

Enter the location after which you want to insert 1

Node inserted

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

8

printing values . . . . .

34

12

32

56

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

4

Node deleted from the begining ...

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

8

printing values . . . . .

12

32

56

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

5

Deleted Node from the last ...

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

8

printing values . . . . .

12

32

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

7

Enter item which you want to search?

34

Item not found

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?

8

printing values . . . . .

12

32

\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*

Choose one option from the following list ...

===============================================

1.Insert in begining

2.Insert at last

3.Insert at any random location

4.Delete from Beginning

5.Delete from last

6.Delete node after specified location

7.Search for an element

8.Show

9.Exit

Enter your choice?9

2. Write a program that uses functions to perform the following operations on doubly linked

list.

i) Creation

ii) Insertion

iii) Deletion

iv) Traversal

PROGRAM:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

struct node {

int data;

struct node \*next;

struct node \*prev;

};

struct node \*head = NULL;

struct node \*last = NULL;

struct node \*current = NULL;

bool isEmpty() {

return head == NULL;

}

int length() {

int length = 0;

struct node \*current;

for(current = head; current != NULL; current = current->next){

length++;

}

return length;

}

void displayForward()

{

struct node \*ptr = head;

while(ptr != NULL)

{

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

ptr = ptr->next;

}

}

void displayBackward() {

struct node \*ptr = last;

while(ptr != NULL) {

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

ptr = ptr ->prev;

}

}

void insertFirst(int data)

{

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

link->data = data;

if(isEmpty()) {

last = link;

} else {

head->prev = link;

}

link->next = head;

head = link;

}

void insertLast(int data)

{

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

link->data = data;

link->next=NULL;

if(isEmpty()) {

last = link;

} else {

last->next = link;

link->prev = last;

}

last = link;

}

void deleteFirst() {

struct node \*tempLink = head;

if(head->next == NULL){

last = NULL;

} else {

head->next->prev = NULL;

}

head = head->next;

}

void deleteLast() {

struct node \*tempLink = last;

if(head->next == NULL) {

head = NULL;

} else {

last->prev->next = NULL;

}

last = last->prev;

}

struct node\* delete(int key) {

struct node\* current = head;

struct node\* previous = NULL;

if(head == NULL) {

return NULL;

}

while(current->data != key) {

if(current->next == NULL) {

return NULL;

} else {

previous = current;

current = current->next;

}

}

if(current == head) {

head = head->next;

} else {

current->prev->next = current->next;

}

if(current == last) {

last = current->prev;

} else {

current->next->prev = current->prev;

}

return current;

}

void insertAfter(int key,int data) {

struct node \*current = head;

if(head == NULL) {

printf("empty");

return;

}

while(current->data != key) {

if(current->next == NULL) {

return;

} else {

current = current->next;

}

}

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

newLink->data = data;

if(current == last) {

newLink->next = NULL;

last = newLink;

} else {

newLink->next = current->next;

current->next->prev = newLink;

}

newLink->prev = current;

current->next = newLink;

}

void main()

{

int choice =0,i,j;

while(1)

{

printf("\n0.insertafter\n1.insertFirst\n2.insertLast\n3.deleteLast\n4.Deletefirst\n5.delete\n6.display

Forward\n7.displayBackward\n8.exit");

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

scanf("%d",&choice);

switch(choice)

{

case 0:

printf("enter key and data");

scanf("%d%d",&i,&j);

insertAfter(i,j);

break;

case 1:

printf("enter data");

scanf("%d",&i);

insertFirst(i);

break;

case 2:

printf("enter data");

scanf("%d",&i);

insertLast(i);

break;

case 3:

deleteLast();

break;

case 4:

deleteFirst();

break;

case 5:

printf("enter element to be deleted");

scanf("%d",&i);

delete(i);

break;

case 6: displayForward();

break;

case 7:displayBackward();

break;

case 8:exit(0);

default:

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

}

}

}

OUTPUT:

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

1

enter data12

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

2

enter data23

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

0

enter key and data1

22

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

6

12

23

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

7

23

12

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

1

enter data45

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

1

enter data123

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

6

123

45

12

23

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

0

enter key and data12

234

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

7

23

234

12

45

123

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

3

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

6

123

45

12

234

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

1

enter data11

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

4

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

1

enter data34

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

5

enter element to be deleted34

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?

7

234

12

45

123

0.insertafter

1.insertFirst

2.insertLast

3.deleteLast

4.Delete first

5.delete

6.displayForward

7.displayBackward

8.exit

Enter your choice?8

3. Write a program that uses functions to perform the following operations on circular linked

list.

i) Creation

ii) Insertion

iii) Deletion

iv) Traversal

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*front=NULL,\*rear=NULL,\*temp;

void enque(int data);

void search(int key);

void deque();

void display();

void main()

{

int no,ch,e;

printf("\n 1 - insert\n 2 - delete \n 3 - Dipslay\n 4 - search\n 5 - Exit");

while (1)

{

printf("\n Enter choice : ");

scanf("%d",&ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d",&no);

enque(no);

break;

case 2:

deque();

break;

case 3:display();

break;

case 4:printf("enter search element");

scanf("%d",&no);

search(no);

break;

case 5:exit(0);

}

}

}

void enque(int d)

{

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

temp->data = d;

temp->next=NULL;

if (rear == NULL)

{

front = temp;

rear=temp;

return;

}

rear->next=temp;

rear=temp;

}

void display()

{

temp = front;

if (front == NULL)

{

printf("Queue is empty");

return;

}

while (temp != NULL)

{

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

temp = temp->next;

}

}

void deque()

{

if (front==NULL)

{

printf("\n Error : Trying to delete from empty queue");

return;

}

front=front->next;

}

void search(int key)

{

temp=front;

while(temp!=NULL)

{

if(temp->data==key)

{

printf("element found");

return;

}

temp=temp->next;

}

printf("element not found");

}

OUTPUT:

1 - insert

2 - delete

3 - Dipslay

4 - search

5 - Exit

Enter choice : 1

Enter data : 12

Enter choice : 1

Enter data : 23

Enter choice : 1

Enter data : 34

Enter choice : 3

12 23 34

Enter choice : 2

Enter choice : 3

23 34

Enter choice : 4

enter search element23

element found

Enter choice : 5

4. Write a program that implement stack (its operations) using

i) Arrays

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

void main()

{

int stk[10],top=-1,c,ch,i;

while(1)

{

printf("\nenter choice 1.push,2.pop,3.display,4.exit");

scanf("%d",&ch);

switch(ch)

{

case 1:printf("\nenter element");

scanf("%d",&c);

if(top==10)

{

printf("\nstack overloaded");

break;

}

stk[++top]=c;

break;

case 2:if(top==-1)

{

printf("\nstack underflow");

break;

}

printf("element poped is %d",stk[top--]);

break;

case 3:for(i=0;i<=top;i++)

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

break;

case 4:exit(1);

}

}

}

OUTPUT:

enter choice 1.push,2.pop,3.display,4.exit1

enter element12

enter choice 1.push,2.pop,3.display,4.exit1

enter element23

enter choice 1.push,2.pop,3.display,4.exit3

12 23

enter choice 1.push,2.pop,3.display,4.exit2

element poped is 23

enter choice 1.push,2.pop,3.display,4.exit3

12

enter choice 1.push,2.pop,3.display,4.exit4

4. Write a program that implement stack (its operations) using

ii) Pointers

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*top=NULL,\*temp;

void push(int data);

void pop();

void display();

void main()

{

int no,ch,e;

printf("\n 1 - Push\n 2 - Pop \n 3 - Dipslay\n 4 - Exit");

while (1)

{

printf("\n Enter choice : ");

scanf("%d",&ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

push(no);

break;

case 2:

pop();

break;

case 3:display();

break;

case 4:exit(0);

}

}

}

void push(int d)

{

temp =(struct node \*)malloc(1\*sizeof(struct node));

temp->data = d;

if (top == NULL)

{

temp->next = NULL;

}

else

{

temp->next = top;

}

top = temp;

}

void display()

{

temp = top;

if (top == NULL)

{

printf("Stack is empty");

return;

}

while (temp != NULL)

{

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

temp = temp->next;

}

}

void pop()

{

if (top==NULL)

{

printf("\n Error : Trying to pop from empty stack");

return;

}

top=top->next;

}

OUTPUT:

1 - Push

2 - Pop

3 - Dipslay

4 - Exit

Enter choice : 1

Enter data : 12

Enter choice : 1

Enter data : 23

Enter choice : 1

Enter data : 34

Enter choice : 3

34 23 12

Enter choice : 2

Enter choice : 3

23 12

Enter choice : 4

5. Write a program that implement Queue (its operations) using

i) Arrays

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

#define maxsize 5

void insert();

void delete();

void display();

int front = -1, rear = -1;

int queue[maxsize];

void main ()

{

int choice;

while(choice != 4)

{

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

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

printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");

printf("\nEnter your choice ?");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("\nEnter valid choice??\n");

}

}

}

void insert()

{

int item;

printf("\nEnter the element\n");

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

if(rear == maxsize-1)

{

printf("\nOVERFLOW\n");

return;

}

if(front == -1 && rear == -1)

{

front = 0;

rear = 0;

}

else

{

rear = rear+1;

}

queue[rear] = item;

printf("\nValue inserted ");

}

void delete()

{

int item;

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

{

printf("\nUNDERFLOW\n");

return;

}

else

{

item = queue[front];

if(front == rear)

{

front = -1;

rear = -1 ;

}

else

{

front = front + 1;

}

printf("\nvalue deleted ");

}

}

void display()

{

int i;

if(rear == -1)

{

printf("\nEmpty queue\n");

}

else

{ printf("\nprinting values .....\n");

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

{

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

}

}

}

OUTPUT:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=================================================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

12

Value inserted

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=================================================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

23

Value inserted

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=================================================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

34

Value inserted

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=================================================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

12

23

34

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=================================================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

value deleted

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=================================================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

value deleted

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=================================================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2# #3

printing values .....

34

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=================================================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?4

5. Write a program that implement Queue (its operations) using

ii) Pointers

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*front=NULL,\*rear=NULL,\*temp;

void enque(int data);

void search(int key);

void deque();

void display();

void main()

{

int no,ch,e;

printf("\n 1 - insert\n 2 - delete \n 3 - Dipslay\n 4 - search\n 5 - Exit");

while (1)

{

printf("\n Enter choice : ");

scanf("%d",&ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d",&no);

enque(no);

break;

case 2:

deque();

break;

case 3:display();

break;

case 4:printf("enter search element");

scanf("%d",&no);

search(no);

break;

case 5:exit(0);

}

}

}

void enque(int d)

{

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

temp->data = d;

temp->next=NULL;

if (rear == NULL)

{

front = temp;

rear=temp;

return;

}

rear->next=temp;

rear=temp;

}

void display()

{

temp = front;

if (front == NULL)

{

printf("Queue is empty");

return;

}

while (temp != NULL)

{

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

temp = temp->next;

}

}

void deque()

{

if (front==NULL)

{

printf("\n Error : Trying to delete from empty queue");

return;

}

front=front->next;

}

void search(int key)

{

temp=front;

while(temp!=NULL)

{

if(temp->data==key)

{

printf("element found");

return;

}

temp=temp->next;

}

printf("element not found");

}

OUTPUT:

1 - insert

2 - delete

3 - Dipslay

4 - search

5 - Exit

Enter choice : 1

Enter data : 12

Enter choice : 1

Enter data : 23

Enter choice : 1

Enter data : 34

Enter choice : 3

12 23 34

Enter choice : 2

Enter choice : 3

23 34

Enter choice : 4

enter search element23

element found

Enter choice : 5

6. Write a program that implements the following sorting methods to sort a given list of

integers in ascending order

i) Bubble sort

PROGRAM:

#include <stdio.h>

#define MAXSIZE 10

void main()

{

int array[MAXSIZE];

int i, j, num, temp;

printf("Enter the value of num \n");

scanf("%d", &num);

printf("Enter the elements one by one \n");

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

{

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

}

printf("Input array is \n");

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

{

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

}

/\* Bubble sorting begins \*/

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

{

for (j = 0; j < (num - i - 1); j++)

{

if (array[j] > array[j + 1])

{

temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

printf("Sorted array is...\n");

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

{

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

}

}

OUTPUT:

Enter the value of num

5

Enter the elements one by one

23

12

34

14

15

Input array is

23

12

34

14

15

Sorted array is...

12

14

15

23

34

6. Write a program that implements the following sorting methods to sort a given list of

integers in ascending order

ii) Selection sort

PROGRAM:

#include<stdio.h>

int main(){

int i, j, count, temp, number[25];

printf("How many numbers u are going to enter?: ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

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

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

// Logic of selection sort algorithm

for(i=0;i<count;i++){

for(j=i+1;j<count;j++){

if(number[i]>number[j]){

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

}

printf("Sorted elements: ");

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

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

return 0;

}

OUTPUT:

How many numbers u are going to enter?: 5

Enter 5 elements: 23

12

34

45

24

Sorted elements: 12 23 24 34 45

6. Write a program that implements the following sorting methods to sort a given list of

integers in ascending order

iii) Insertion sort

PROGRAM:

#include<stdio.h>

int main()

{

int i, j, count, temp, number[25];

printf("How many numbers u are going to enter?: ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

// This loop would store the input numbers in array

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

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

for(i=1;i<count;i++){

temp=number[i];

j=i-1;

while((temp<number[j])&&(j>=0)){

number[j+1]=number[j];

j=j-1;

}

number[j+1]=temp;

}

printf("Order of Sorted elements: ");

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

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

return 0;

}

OUTPUT:

How many numbers u are going to enter?: 5

Enter 5 elements: 23

12

45

32

11

Order of Sorted elements: 11 12 23 32 45

7. Write a program that use both recursive and non recursive functions to perform the

following searching operations for a Key value in a given list of integers:

i) Linear search(NON RECURSIVE)

PROGRAM:

#include<stdio.h>

int main()

{

int a[20],i,x,n;

printf("How many elements?");

scanf("%d",&n);

printf("Enter array elements");

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

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

printf("nEnter element to search:");

scanf("%d",&x);

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

if(a[i]==x)

break;

if(i<n)

printf("Element found at index %d",i);

else

printf("Element not found");

return 0;

}

OUTPUT:

How many elements?5

Enter array elements21

23

22

34

45

nEnter element to search:34

Element found at index 3

7. Write a program that use both recursive and non recursive functions to perform the

following searching operations for a Key value in a given list of integers:

i) Linear search(RECURSIVE)

PROGRAM:

#include <stdio.h>

int RecursiveLS(int arr[], int value, int index, int n)

{

int pos = 0;

if(index >= n)

{

return 0;

}

else if (arr[index] == value)

{

pos = index + 1;

return pos;

}

else

{

return RecursiveLS(arr, value, index+1, n);

}

return pos;

}

int main()

{

int n, value, pos, m = 0, arr[100];

printf("Enter the total elements in the array ");

scanf("%d", &n);

printf("Enter the array elements\n");

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

{

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

}

printf("Enter the element to search ");

scanf("%d", &value);

pos = RecursiveLS(arr, value, 0, n);

if (pos != 0)

{

printf("Element found at pos %d ", pos);

}

else

{

printf("Element not found");

}

return 0;

}

OUTPUT:

Enter the total elements in the array 5

Enter the array elements

12

23

32

43

31

Enter the element to search 43

Element found at pos 4

7. Write a program that use both recursive and non recursive functions to perform the

following searching operations for a Key value in a given list of integers:

ii) Binary search(RECURSIVE)

PROGRAM:

#include <stdio.h>

void binary\_search(int [], int, int, int);

void bubble\_sort(int [], int);

int main()

{

int key, size, i;

int list[25];

printf("Enter size of a list: ");

scanf("%d", &size);

printf("Enter elements\n");

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

{

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

}

bubble\_sort(list, size);

printf("\n");

printf("Enter key to search\n");

scanf("%d", &key);

binary\_search(list, 0, size, key);

}

void bubble\_sort(int list[], int size)

{

int temp, i, j;

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

{

for (j = i; j < size; j++)

{

if (list[i] > list[j])

{

temp = list[i];

list[i] = list[j];

list[j] = temp;

}

}

}

}

void binary\_search(int list[], int lo, int hi, int key)

{

int mid;

if (lo > hi)

{

printf("Key not found\n");

return;

}

mid = (lo + hi) / 2;

if (list[mid] == key)

{

printf("Key found\n");

}

else if (list[mid] > key)

{

binary\_search(list, lo, mid - 1, key);

}

else if (list[mid] < key)

{

binary\_search(list, mid + 1, hi, key);

}

}

OUTPUT:

Enter size of a list: 5

Enter elements

12

23

21

32

43

Enter key to search

21

Key found

7. Write a program that use both recursive and non recursive functions to perform the

following searching operations for a Key value in a given list of integers:

ii) Binary search(NON RECURSIVE)

PROGRAM:

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

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

scanf("%d", &n);

printf("Enter %d integers\n", n);

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

scanf("%d", &array[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if (first > last)

printf("Not found! %d isn't present in the list.\n", search);

return 0;

}

OUTPUT:

Enter number of elements

5

Enter 5 integers

21

23

24

25

22

Enter value to find

25

25 found at location 4.

8. Write a program to implement the tree traversal methods.

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

struct btnode

{

int value;

struct btnode \*l;

struct btnode \*r;

}\*root = NULL, \*temp = NULL, \*t2, \*t1;

void delete1();

void insert();

void delete();

void inorder(struct btnode \*t);

void search(struct btnode \*t);

void preorder(struct btnode \*t);

void postorder(struct btnode \*t);

void search1(struct btnode \*t,int data);

int smallest(struct btnode \*t);

int largest(struct btnode \*t);

int flag = 1;

void main()

{

int ch;

printf("\nOPERATIONS ---");

printf("\n1 - Insert an element into tree\n");

printf("2 - Delete an element from the tree\n");

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

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

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

printf("6 - Exit\n");

while(1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

inorder(root);

break;

case 4:

preorder(root);

break;

case 5:

postorder(root);

break;

case 6:

exit(0);

default :

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

void insert()

{

int data;

printf("Enter data of node to be inserted : ");

scanf("%d", &data);

temp = (struct btnode \*)malloc(1\*sizeof(struct btnode));

temp->value = data;

temp->l = temp->r = NULL;

if (root == NULL)

root = temp;

else

search(root);

}

void search(struct btnode \*t)

{

if ((temp->value > t->value) && (t->r != NULL))

search(t->r);

else if ((temp->value > t->value) && (t->r == NULL))

t->r = temp;

else if ((temp->value < t->value) && (t->l != NULL))

search(t->l);

else if ((temp->value < t->value) && (t->l == NULL))

t->l = temp;

}

void inorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

if (t->l != NULL)

inorder(t->l);

printf("%d -> ", t->value);

if (t->r != NULL)

inorder(t->r);

}

void delete()

{

int data;

if (root == NULL)

{

printf("No elements in a tree to delete");

return;

}

printf("Enter the data to be deleted : ");

scanf("%d", &data);

t1 = root;

t2 = root;

search1(root, data);

}

void preorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

printf("%d -> ", t->value);

if (t->l != NULL)

preorder(t->l);

if (t->r != NULL)

preorder(t->r);

}

void postorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display ");

return;

}

if (t->l != NULL)

postorder(t->l);

if (t->r != NULL)

postorder(t->r);

printf("%d -> ", t->value);

}

void search1(struct btnode \*t, int data)

{

if ((data>t->value))

{

t1 = t;

search1(t->r, data);

}

else if ((data < t->value))

{

t1 = t;

search1(t->l, data);

}

else if ((data==t->value))

{

delete1(t);

}

}

void delete1(struct btnode \*t)

{

int k;

if ((t->l == NULL) && (t->r == NULL))

{

if (t1->l == t)

{

t1->l = NULL;

}

else

{

t1->r = NULL;

}

t = NULL;

free(t);

return;

}

else if ((t->r == NULL))

{

if (t1 == t)

{

root = t->l;

t1 = root;

}

else if (t1->l == t)

{

t1->l = t->l;

}

else

{

t1->r = t->l;

}

t = NULL;

free(t);

return;

}

else if (t->l == NULL)

{

if (t1 == t)

{

root = t->r;

t1 = root;

}

else if (t1->r == t)

t1->r = t->r;

else

t1->l = t->r;

t == NULL;

free(t);

return;

}

else if ((t->l != NULL) && (t->r != NULL))

{

t2 = root;

if (t->r != NULL)

{

k = smallest(t->r);

flag = 1;

}

else

{

k =largest(t->l);

flag = 2;

}

search1(root, k);

t->value = k;

}

}

int smallest(struct btnode \*t)

{

t2 = t;

if (t->l != NULL)

{

t2 = t;

return(smallest(t->l));

}

else

return (t->value);

}

int largest(struct btnode \*t)

{

if (t->r != NULL)

{

t2 = t;

return(largest(t->r));

}

else

return(t->value);

}

OUTPUT:

OPERATIONS ---

1 - Insert an element into tree

2 - Delete an element from the tree

3 - Inorder Traversal

4 - Preorder Traversal

5 - Postorder Traversal

6 - Exit

Enter your choice : 11

Wrong choice, Please enter correct choice

Enter your choice : 1

Enter data of node to be inserted : 12

Enter your choice : 1

Enter data of node to be inserted : 23

Enter your choice : 1

Enter data of node to be inserted : 34

Enter your choice : 1

Enter data of node to be inserted : 1

Enter your choice : 3

1 -> 12 -> 23 -> 34 ->

Enter your choice : 4

12 -> 1 -> 23 -> 34 ->

Enter your choice : 5

1 -> 34 -> 23 -> 12 ->

Enter your choice : 2

Enter the data to be deleted : 23

Enter your choice : 3

1 -> 12 -> 34 ->

Enter your choice : 6

9. Write a program to implement the graph traversal methods.

1. //Graph Traversals DFS

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 5

struct Vertex {

char label;

bool visited;

};

//stack variables

int stack[MAX];

int top = -1;

//graph variables

//array of vertices

struct Vertex\* lstVertices[MAX];

//adjacency matrix

int adjMatrix[MAX][MAX];

//vertex count

int vertexCount = 0;

//stack functions

void push(int item) {

stack[++top] = item;

}

int pop() {

return stack[top--];

}

int peek() {

return stack[top];

}

bool isStackEmpty() {

return top == -1;

}

//graph functions

//add vertex to the vertex list

void addVertex(char label) {

struct Vertex\* vertex = (struct Vertex\*) malloc(sizeof(struct Vertex));

vertex->label = label;

vertex->visited = false;

lstVertices[vertexCount++] = vertex;

}

//add edge to edge array

void addEdge(int start,int end) {

adjMatrix[start][end] = 1;

adjMatrix[end][start] = 1;

}

//display the vertex

void displayVertex(int vertexIndex) {

printf("%c ",lstVertices[vertexIndex]->label);

}

//get the adjacent unvisited vertex

int getAdjUnvisitedVertex(int vertexIndex) {

int i;

for(i = 0; i < vertexCount; i++) {

if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false)

{

return i;

}

}

return -1;

}

void depthFirstSearch()

{

int i;

//mark first node as visited

lstVertices[0]->visited = true;

//display the vertex

displayVertex(0);

//push vertex index in stack

push(0);

while(!isStackEmpty())

{

//get the unvisited vertex of vertex which is at top of the stack

int unvisitedVertex = getAdjUnvisitedVertex(peek());

//no adjacent vertex found

if(unvisitedVertex == -1)

{

pop();

}

else

{

lstVertices[unvisitedVertex]->visited = true;

displayVertex(unvisitedVertex);

push(unvisitedVertex);

}

}

//stack is empty, search is complete, reset the visited flag

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

{

lstVertices[i]->visited = false;

}

}

int main()

{

int i, j;

for(i = 0; i < MAX; i++) // set adjacency

{

for(j = 0; j < MAX; j++) // matrix to 0

adjMatrix[i][j] = 0;

}

addVertex('A'); // 0

addVertex('B'); // 1

addVertex('C'); // 2

addVertex('D'); // 3

addVertex('E'); // 4

addEdge(0, 1); // A - B

addEdge(0, 2); // A - C

addEdge(0, 3); // A - D

addEdge(1, 3); // B - D

addEdge(1, 4); // B - E

addEdge(2, 3); // C - D

addEdge(3, 4); // D - E

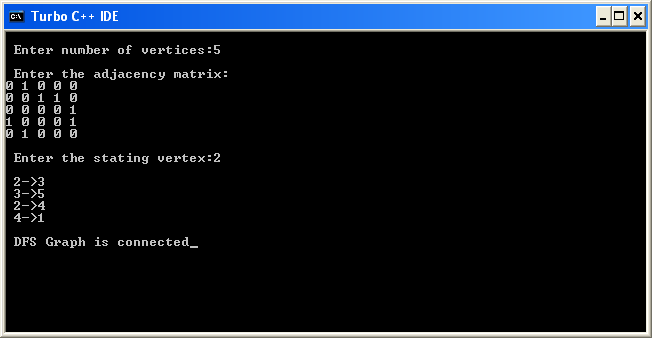
printf("Depth First Search: ");

depthFirstSearch();

return 0;

}

**OUTPUT:**

****

1. //Graph Traversals BFS

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 5

struct Vertex

{

char label;

bool visited;

};

//queue variables

int queue[MAX];

int rear = -1;

int front = 0;

int queueItemCount = 0;

//graph variables

//array of vertices

struct Vertex\* lstVertices[MAX];

//adjacency matrix

int adjMatrix[MAX][MAX];

//vertex count

int vertexCount = 0;

//queue functions

void insert(int data)

{

queue[++rear] = data;

queueItemCount++;

}

int removeData()

{

queueItemCount--;

return queue[front++];

}

bool isQueueEmpty() {

return queueItemCount == 0;

}

//graph functions

//add vertex to the vertex list

void addVertex(char label)

{

struct Vertex\* vertex = (struct Vertex\*) malloc(sizeof(struct Vertex));

vertex->label = label;

vertex->visited = false;

lstVertices[vertexCount++] = vertex;

}

//add edge to edge array

void addEdge(int start,int end)

{

adjMatrix[start][end] = 1;

adjMatrix[end][start] = 1;

}

//display the vertex

void displayVertex(int vertexIndex)

{

printf("%c ",lstVertices[vertexIndex]->label);

}

//get the adjacent unvisited vertex

int getAdjUnvisitedVertex(int vertexIndex)

{

int i;

for(i = 0; i<vertexCount; i++) {

if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false)

return i;

}

return -1;

}

void breadthFirstSearch()

{

int i;

//mark first node as visited

lstVertices[0]->visited = true;

//display the vertex

displayVertex(0);

//insert vertex index in queue

insert(0);

int unvisitedVertex;

while(!isQueueEmpty()) {

//get the unvisited vertex of vertex which is at front of the queue

int tempVertex = removeData();

//no adjacent vertex found

while((unvisitedVertex = getAdjUnvisitedVertex(tempVertex)) != -1) {

lstVertices[unvisitedVertex]->visited = true;

displayVertex(unvisitedVertex);

insert(unvisitedVertex);

}

}

//queue is empty, search is complete, reset the visited flag

for(i = 0;i<vertexCount;i++) {

lstVertices[i]->visited = false;

}

}

int main()

{

int i, j;

for(i = 0; i<MAX; i++) // set adjacency

{

for(j = 0; j<MAX; j++) // matrix to 0

adjMatrix[i][j] = 0;

}

addVertex('A'); // 0

addVertex('B'); // 1

addVertex('C'); // 2

addVertex('D'); // 3

addVertex('E'); // 4

addEdge(0, 1); // A - B

addEdge(0, 2); // A - C

//addEdge(0, 3); // A - D

addEdge(1, 3); // B - D

addEdge(1, 4); // B - E

addEdge(2, 3); // C - D

addEdge(3, 4); // D - E

printf("\nBreadth First Search: ");

breadthFirstSearch();

return 0;

}

**OUTPUT:**

