Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400050

Division : C3

Batch : C31

Roll Number : 2003148

Certify that Mr Priyansh Salian of Computers Department, Semester III with Roll No. 2003148 has completed a course of the necessary experiments in the subject Data Structures Lab (CSL301) under my supervision in the Thadomal Shahani Engineering College
Laboratory in the year 2021 - 2022

Teacher In-Charge

Prof. Anagha Durugkar

Head of the Department

Date

_____ Principal

List of Experiments
S.E. (Comp.) Sem III
Subject: Data Structures Lab (CSL301)

Title of Experiments

Expt No.	Experiment	Date
1	Implement Stack ADT using array.	20/8/21
2	Convert an Infix expression to Postfix expression using stack ADT.	27/8/21
3	Evaluate Postfix Expression using Stack ADT.	3/9/21
4	Implement Linear Queue ADT using array.	10/9/21
5	Implement Singly Linked List ADT.	24/9/21
6	Implement Circular Queue ADT using array.	1/10/21
7	Implement Circular Linked List ADT.	29/10/21
8	Implement Stack / Linear Queue ADT using Linked	15/10/21
	List.	
9	Implement Binary Search Tree ADT using Linked	19/11/21
	List.	
10	Implement Graph Traversal techniques:) Depth	26/11/21
	First Search b) Breadth First Search	
	Written Assignments/Mini Project	
1	Assignment 1:	24/9/21
2	Assignment 2:	19/11/21
	EXTRA:	
1	AVL Tree	

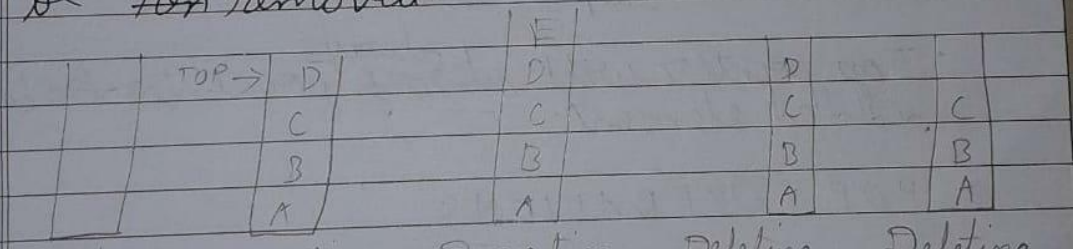
Experiment 1

AIM - To perform and analyse stack operations

THEORY

Stack is a linear data structure in which addition of a new element or deletion of an existing element always takes place at the same end which is known as 'TOP' to the stack.

As the elements in the stack can be added or removed from the top, the stack works on the principle of last in first out as the last element is the first to be removed.



Stack operations

We can perform two main operations on a stack

(1) PUSH

(2) POP

1. PUSH OPERATION

- Means adding a new element in the stack
- For pushing or adding the new element on the stack, first we have to the check the condition of overflow i.e. stack is full or not, then only we can push the element.

- Code

```
if (top == max - 1)
{
    printf ("Stack Overflow");
}
else
{
    top = top + 1;
    arr[top] = element;
}
```

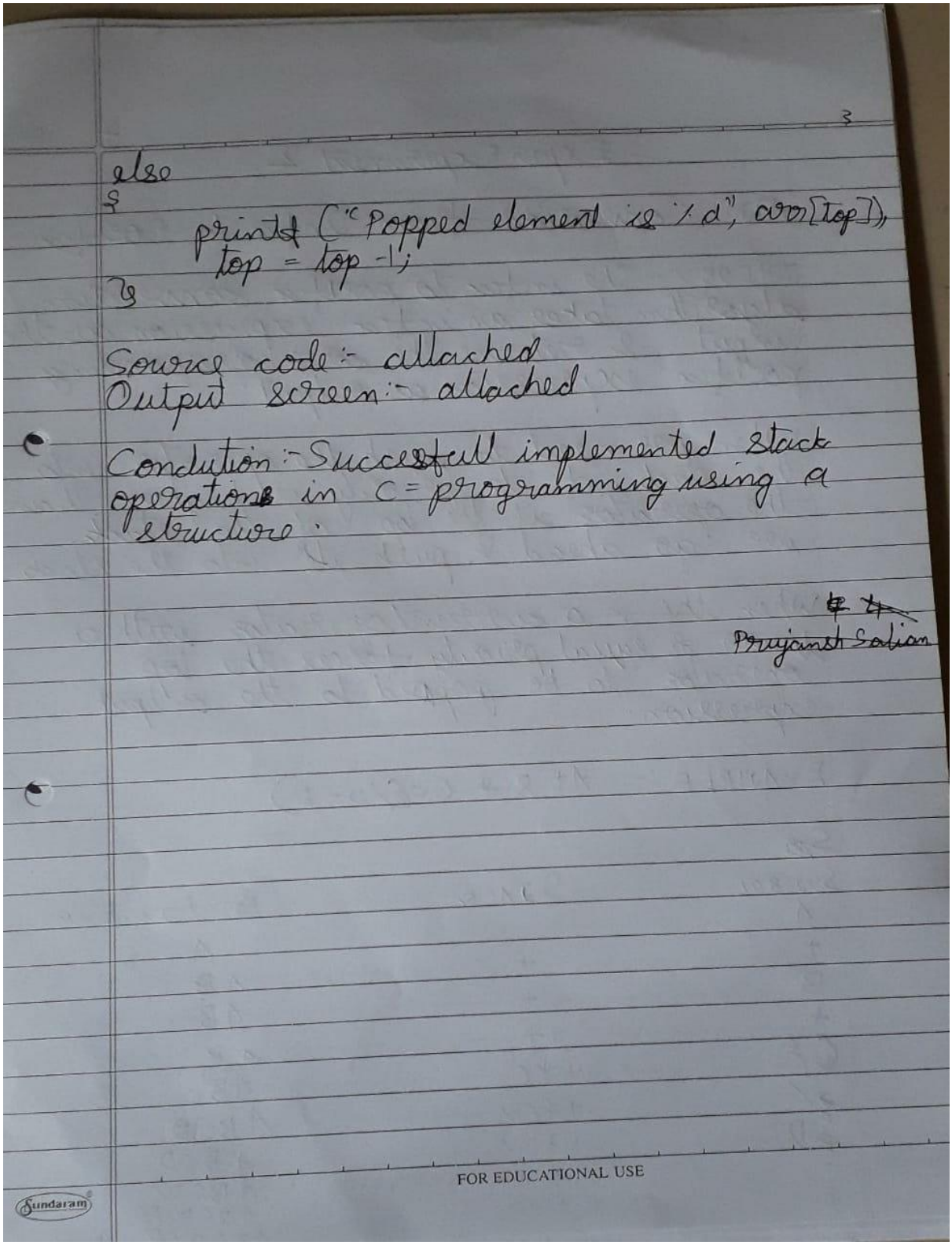
- Top pointer will always point to the last added element.

2. POP OPERATIONS

- Means removing an element from the stack
- For pop operation we should check the condition of or the underflow

Code:

```
if (top == -1)
{
    printf ("Underflow
}
```



SOURCE CODE:

```
#include<conio.h>
#include<stdio.h>
#define max 10
struct stack
```

```
{
int a[max];
int top;
};
void init(struct stack* s)
{
s->top=-1;
}
int empty(struct stack* s)
{
return s->top==-1;
}
int full(struct stack* s)
{
return s->top==max-1;
}
void push(struct stack* s,int data)
{
if(!full(s))
{
s->a[++s->top]=data;
}
else{
printf("Stack is full \n");
}
}
```

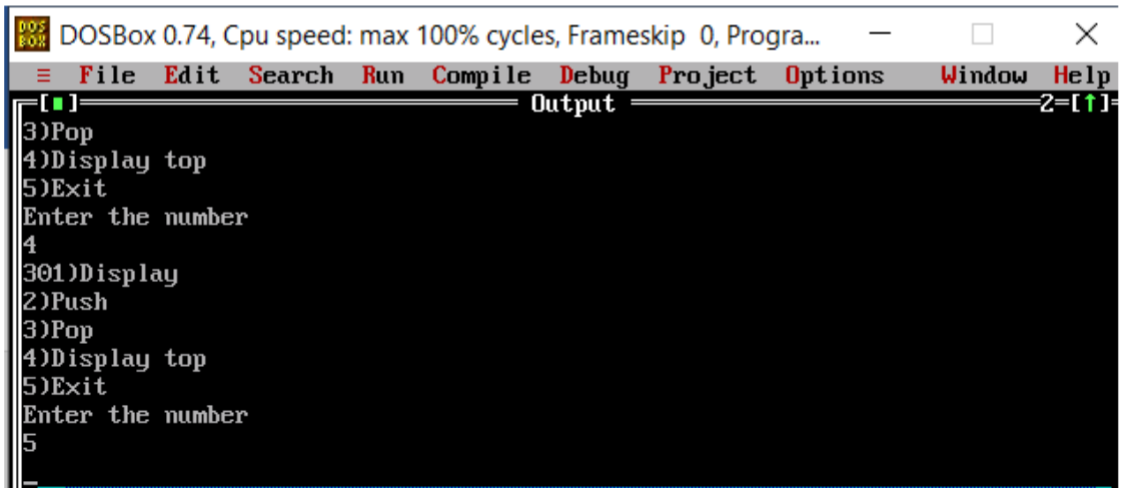
```
int pop(struct stack* s)
{
if(!empty(s))
{
int data=s->a[s->top];
--s->top;
printf("Element %d popped\n", data);
return data;
}
else
printf("Stack is empty \n");
}
int displayTop(struct stack* s)
{
if(!empty(s))
{
return s->a[s->top];
}
}
void display(struct stack* s)
{int i;
for(i=s->top;i>=0;i--)
{
printf("%d \n\n",s->a[i]);
}
}
int main()
{
struct stack s;
int c=1,k,n;
clrscr();
```

```
init(&s);
while(c!=0)
{
printf("1)Display \n2)Push \n3)Pop \n4)Display top \n5)Exit \nEnter the number \n");
scanf("%d",&k);
switch(k)
{
case 1:
display(&s);
break;
case 2:
printf("Enter the number to be pushed\n");
scanf("%d",&n);
push(&s,n);
break;
case 3:
pop(&s);
break;
case 4:
printf("%d",displayTop(&s));
break;
case 5:
c=0;
break;
default:
printf("Invalid input\n");
}
}
getch();
return 0;
}
```

OUTPUT SCREENS:

```
1)Display
2)Push
3)Pop
4)Display top
5)Exit
Enter the number
2
Enter the number to be pushed
10
1)Display
2)Push
3)Pop
4)Display top
5)Exit
Enter the number
2
Enter the number to be pushed
20
1)Display
2)Push
3)Pop
4)Display top
5)Exit
Enter the number
_
```

```
Enter the number
1
40

30

20

10

1)Display
2)Push
3)Pop
4)Display top
5)Exit
Enter the number
3
Element 40 popped
1)Display
2)Push
3)Pop
4)Display top
5)Exit
Enter the number
_
```



Experiment 2

AIM - Conversion from Infix to Postfix

THEORY - The infix to postfix conversion algorithm takes an infix expression as the input & returns the corresponding postfix expression as output.

When we need to push the operator into the stack, if its priority is higher than the operator at the top of the stack, we go ahead & push its into the stack.

When a current operator with a lower or equal priority forces the top operator to be popped to the output expression.

EXAMPLE :- A + B * (C (E / D - E)

SYMBOL	STACK	POSTFIX EXP
A		A
+	+	A
B	+	AB
*	+ *	AB
(	+ * (	ABC
C	+ * (	ABC
/	+ * (/	ABC
D	+ * (/	ABCD
-	+ * (-	ABCD/
E	+ * (-	ABCD/E
)	+ *	ABCD/E-*+

Source Code - Attached

Output screen - Attached

Conclusion - Successfully implement infix to postfix conversion in C-programming language.

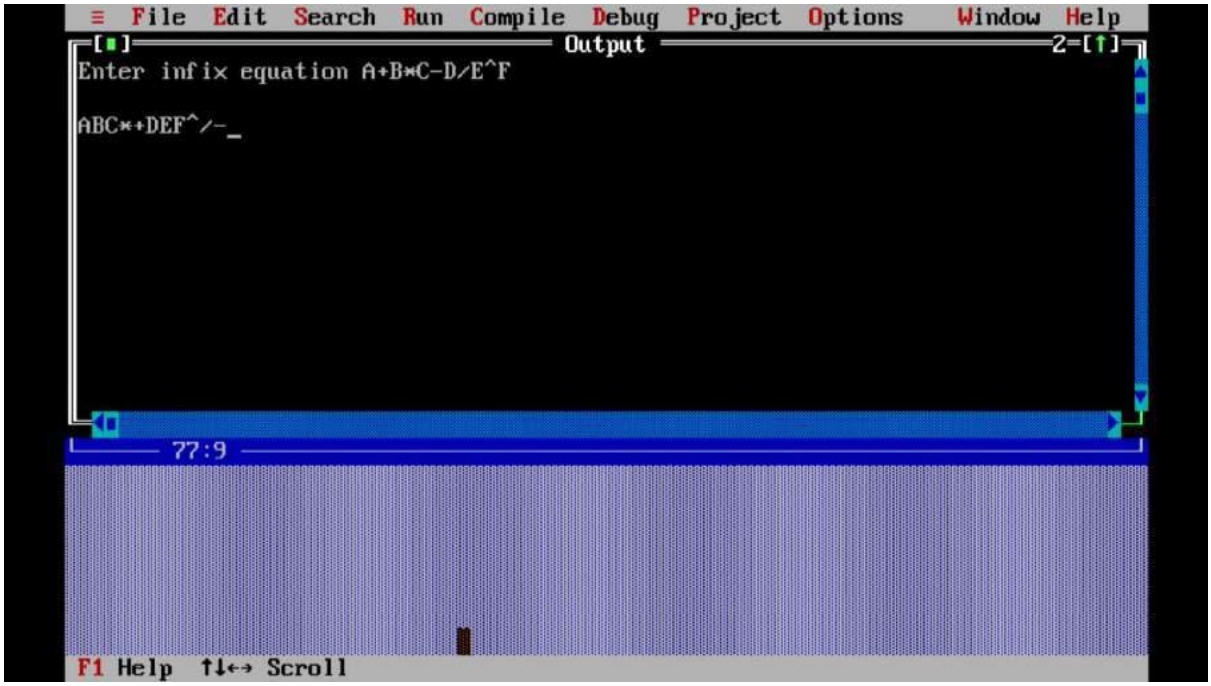
SOURCE CODE

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int i,j=-1,k=0;
char infix[50];
char postfix[50];
char stack[50];
int checkChar(char a)
{
if(infix[i]=='^' || infix[i]==')' || infix[i]=='(' || infix[i]=='%' || infix[i]=='*' || infix[i]=='+' || infix[i]=='/'
|| infix[i]=='-')
return 1; else

return 0;
}
void push(){
stack[++j]=infix[i];
}
void pop(){
postfix[k++]=stack[j--];
}
int priority(char a)
{
if(a=='^' || a=='$')
return 3;
else if(a=='/' || a=='*')
return 2;
else if(a=='+' || a=='-')
return 1;
else
```

```
return 0;
}
int main()
{
char ans[50];
clrscr();
printf("Enter infix equation ");
scanf("%s",&infix);
for(i=0;infix[i]!='\0';i++)
{
if(checkChar(infix[i]))
{
if(infix[i]==')')
{
while(stack[j]!='(')
pop();
j--;
}
else
{
if(infix[i]=='(' || j==-1 || priority(infix[i])>priority(stack[j]))
push();
else
{
while(priority(infix[i])<=priority(stack[j]))
{
pop();
if(j==-1)
break;
}
push();
```

```
}
}
}
else
{
postfix[k++]=infix[i];
}
}
stack[++j]='\0';
postfix[k]='\0';
printf("\n%s",postfix);
for(i=j-1;i>=0;i--)
{ if(stack[i]=='(')
continue;
printf("%c",stack[i]);
}
getch();
return 0;
}
```

OUTPUT SCREENS:



Experiment - 3

AIM - To execute POSTFIX expression using Stack ADT

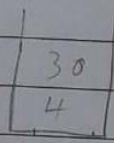
THEORY -

To evaluate a postfix expression, execute the following steps until the end of the expression:

1. If you recognize an operand, push it on the stack.
2. If you recognize an operator pop the top two elements of the stack & perform the operation.
3. Store the value of the operation as the top of the stack.

Ex:- AB C * + DE F / - + , A=4, B=5, C=6, D=2, E=7, F=9

∴ 4 5 6 * + 2 7 9 / - +

Postfix Element	Stack
4	
5	
6	
*	

+

2

7

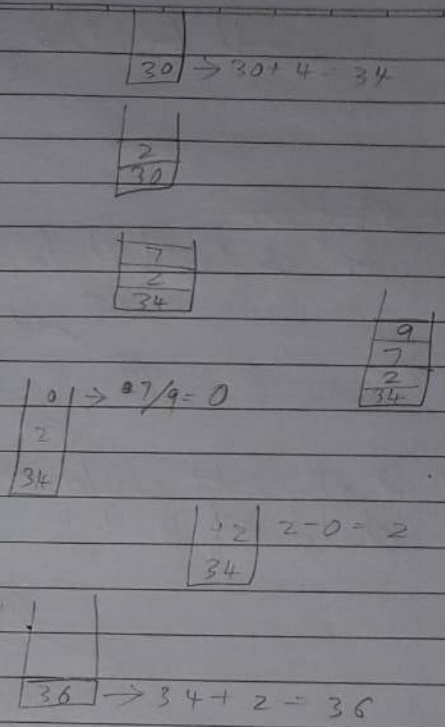
9

11

/

−

+



∴ Answer ~~for of~~ the postfix expression is 36

11 X

Source Code :- Attached

Output ~~Code~~ :- screen :- Attached

Conclusion :- Successfully implemented POSTFIX evaluation using TURBO C

SOURCE CODE:

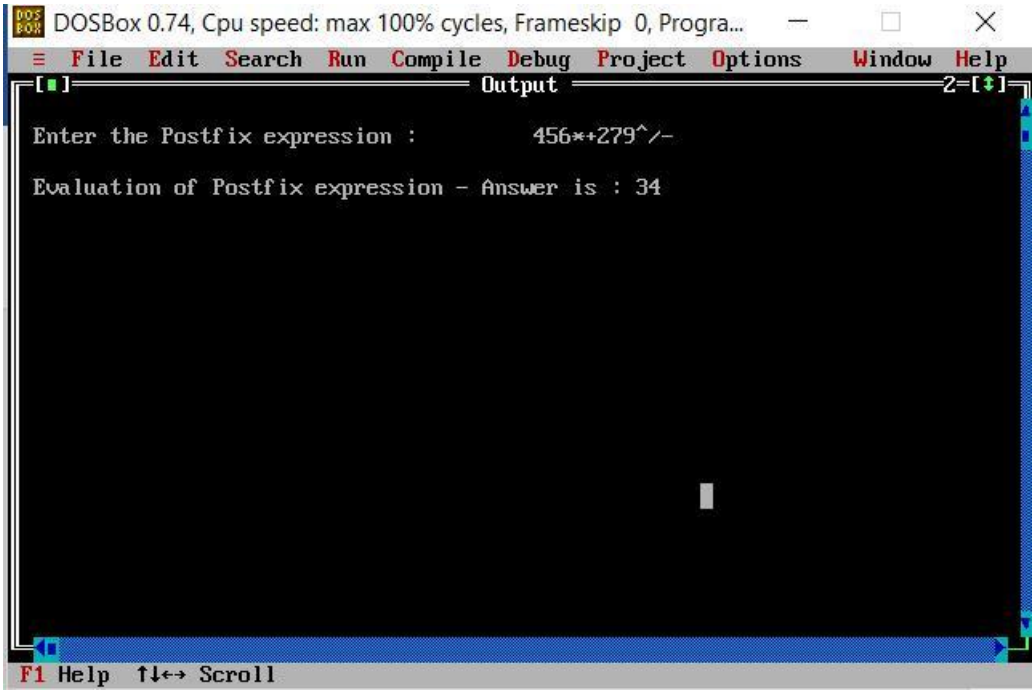
```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define MAX 80
int stack[MAX];
int top=-1;
int gettype(char t)
 {
  switch(t)
  {
  case '+':
  case '-':
  case '*':
  case '/':
  case '%':
  case '^':
  case '$': return 2;
  default: return 1;
  }
}
void push( int temp)
{
 if(top==MAX-1)
 {
   printf("\n Stack full");
 }
 else
 stack[++top]=temp;
}
```

```
void eval(char op,int num1, int num2)
{
  int result;
  switch(op)
  {
  case '+' :result=num1 + num2;
                break;
  case '-' :result=num1 - num2;
                break;
  case '*' :result=num1 * num2;
                break;
  case '/' :result=num1 / num2;
                break;
  case '%' :result=num1 % num2;
                break;
  case '^':
  case '$': result=pow(num1,num2) ;
    }
   push(result);
}

int pop()
{
  int x;
  if(top == -1)
    printf("stack-Underflow");
  else
    x=stack[top--];

 return x;
```

```
 }
void main()
{
int ans,i,n1,n2,res,ch;
char exp[MAX];
clrscr();
printf("\n Enter the Postfix expression : \t");
scanf("%s",exp);
for(i=0;exp[i]!='\0';i++)
 {
   ans=gettype(exp[i]);
   switch(ans)
   {
   case 1:
   {
    ch=exp[i]-'0';
    push(ch);
    break;
   }
   case 2:
   {
   n2=pop();
   n1=pop();
   eval(exp[i],n1,n2);
   break;
   }
   default:
   {
   printf("Invalid choice");
   break;
   }
```

```
  }
}
printf("\n Evaluation of Postfix expression - Answer is : %d",pop());
getch();
}
```

OUTPUT:



Experiment 4

AIM - To implement linear queue ADT using array.

Theory - Part of memory which operates on first in first out or last in last out principle.

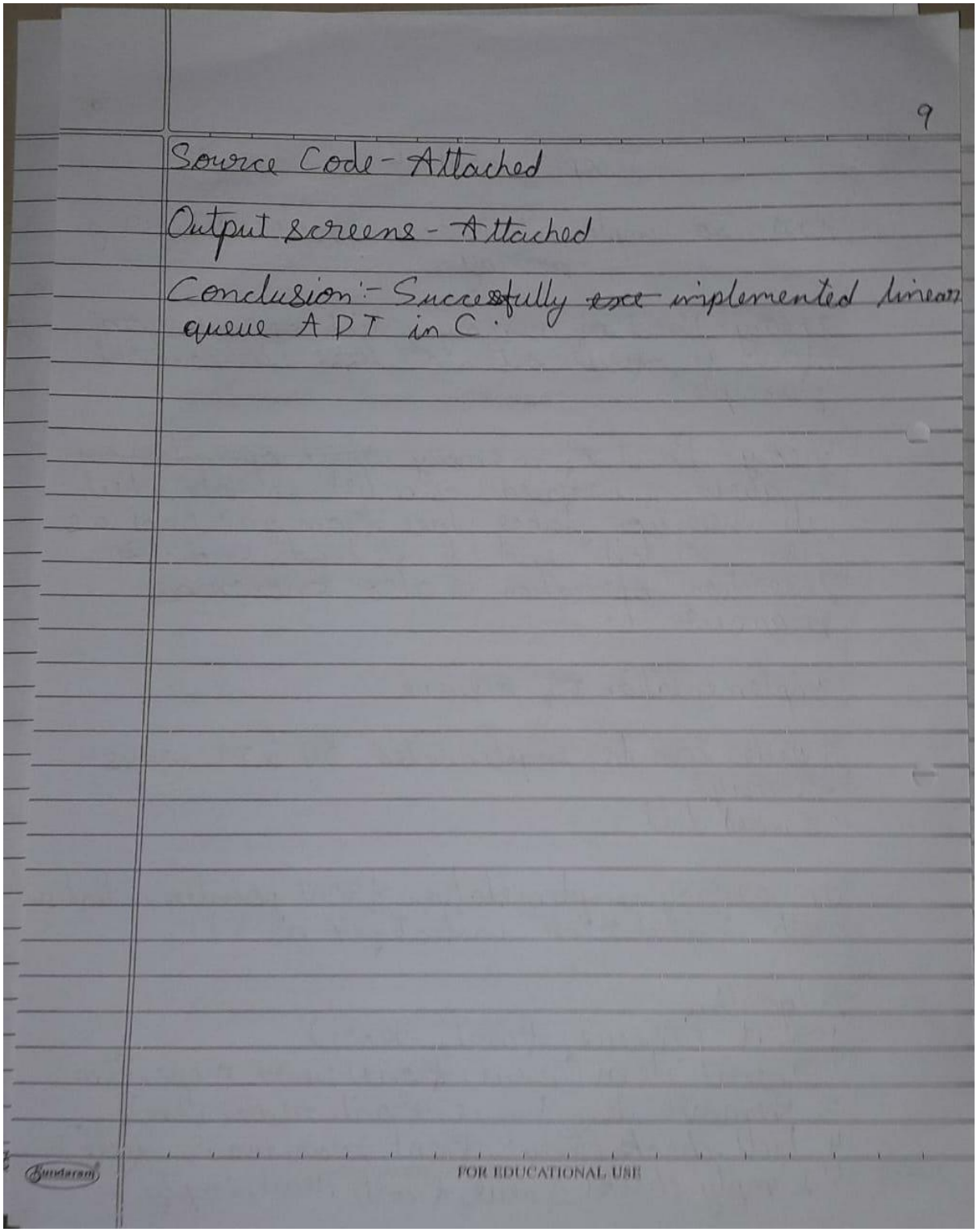
Theory - Part of memory which operates on
A queue is defined as a list in which all insertion takes place from one end as "rear" or "tail" end & or "head" end. The insertion operation is also known as "enqueue".

Implementation of Queue

Queue can be implemented by two ways:-
- Arrays
- Linked list

In array implemetation front pointer initialise with 0 and rear initialized as -1.

Algorithm
1. Init (Queue, front, rear)
2. Insert item (Queue, front, rear, max, item)
3. Remove item (Queue, front, rear, item)
4. Full check (Queue, front, rear, max, full)
5. Empty check (Queue, front, rear, empty)



SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void enQueue(int);
void deQueue();
```

```
void display();
int queue[MAX], front=0, rear=-1;
void main()
{
int value, choice;
clrscr();
while(choice!=4)
{
printf("\nMENU\n1.Insert\n2.Delete\n3.Show\n4.Exit");
printf("\nEnter choice:");
scanf("%d",&choice);
switch(choice)
{
case 1:
{
printf("Enter value to insert: ");
scanf("%d",&value);
enQueue(value);
break;
}
case 2:
{
deQueue();
break;
}
```

```
case 3:
{
display();
break;
}
case 4:
printf("Exiting");
break;
default:
printf("Entered wrong choice");
}
}
}
void enQueue(int value)
{
if(rear==MAX-1)
printf("\nQueue is full");
else
{
if(front==-1)
front=0;
rear++;
queue[rear]=value;
printf("\nInsertion successful");
}
}
void deQueue()
{
if(front==rear)
printf("\nQueue is Empty");
else
```

```
{
printf("\nDeleted: %d",queue[front]);
if(front==rear)
front=rear=-1;
}
}
void display()
{
if(rear==-1)
printf("\nQueue is empty");
else
{
int i;
printf("\nQueue elements are: ");
for(i=front; i<=rear; i++)
{
printf("\n %d\t",queue[i]);
}
}
}
```

OUTPUT SCREENS:

```
MENU
1.Insert
2.Delete
3.Show
4.Exit
Enter choice:1
Enter value to insert: 10

Insertion successful
MENU
1.Insert
2.Delete
3.Show
4.Exit
Enter choice:1
Enter value to insert: 20

Insertion successful
MENU
1.Insert
2.Delete
3.Show
4.Exit
Enter choice:1
Enter value to insert: 30
```

```
2.Delete
3.Show
4.Exit
Enter choice:2

Deleted: 10
MENU
1.Insert
2.Delete
3.Show
4.Exit
Enter choice:3

Queue elements are:
 10
 20
 30
 40
 50
MENU
1.Insert
2.Delete
3.Show
4.Exit
Enter choice:4
```

Experiment - 5

AIM - To implement singly linked list ADT

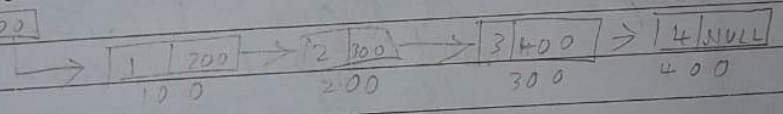
Theory :-

- A linked list is a collection of element called nodes
- Each node has two parts
- First part is called as information field & second contains address of the next node.
- The address of the last node of linked list will have null value.
- In a linked list memory is assigned with use of structure which is it self referential structure.

```
struct node
{
    int data;
    struct node* link;
};
struct node* start = NULL;
```

Here, number of structure struct node* link points to the structure itself.

Ex -



Source Code - Attached
Output Screen - Attached
Conclusion - Successfully implement single linked list ADT list

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
struct node
{
int data;
struct node *next;
};
struct node *start = NULL;
struct node *create_ll(struct node *start)
{
struct node *new_node, *ptr;
int num;
printf("\n Enter -1 to end");
printf("\n Enter the data : ");
scanf("%d", &num);
while(num!=-1)
{
new_node = (struct node*)malloc(sizeof(struct node));
new_node -> data=num;
if(start==NULL)
{
new_node -> next = NULL;
start = new_node;
```

```
}
else
{
ptr=start;
while(ptr->next!=NULL)
ptr=ptr->next;
ptr->next = new_node;
new_node->next=NULL;
}
printf("\n Enter the data : ");
scanf("%d", &num);
}
return start;
}
struct node *display(struct node *start)
{
struct node *ptr;
ptr = start;
while(ptr != NULL)
{
printf("\t %d", ptr -> data);
ptr = ptr -> next;
}
return start;
}
```

```
struct node *insert_beg(struct node *start)
{
struct node *new_node;
int num;
printf("\n Enter the data : ");
scanf("%d", &num);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
new_node -> next = start;
start = new_node;
return start;
}
struct node *insert_end(struct node *start)
{
struct node *ptr, *new_node;
int num;
printf("\n Enter the data : ");
scanf("%d", &num);
new_node = (struct node *)malloc(sizeof(struct node));
new_node -> data = num;
new_node -> next = NULL;
ptr = start;
while(ptr -> next != NULL)
ptr = ptr -> next;
ptr -> next = new_node;
```

```
return start;
}
struct node *delete_beg(struct node *start)
{
struct node *ptr;
ptr = start;
start = start -> next;
free(ptr);
return start;
}
struct node *delete_end(struct node *start)
{
struct node *ptr, *preptr;
ptr = start;
while(ptr -> next != NULL)
{
preptr = ptr;
ptr = ptr -> next;
}
preptr -> next = NULL;
free(ptr);
return start;
}
```

```
int main(int argc, char *argv[]) {

int option;

do

{

printf(“\n\n *****MAIN MENU *****”);

printf(“\n 1: Create a list”);

printf(“\n 2: Display the list”);

printf(“\n 3: Add a node at the beginning”);

printf(“\n 4: Add a node at the end”);

printf(“\n 5: Delete a node from the beginning”);

printf(“\n 6: Delete a node from the end”);

printf(“\n 7: EXIT”);

printf(“\n\n Enter your option : “);

scanf(“%d”, &option);

switch(option)

{

case 1: start = create_ll(start);

printf(“\n LINKED LIST CREATED”);

break;

case 2: start = display(start);

break;

case 3: start = insert_beg(start);

break;

case 4: start = insert_end(start);

break;
```

```
case 5: start =
delete_beg(start); break;
case 6: start =
delete_end(start); break;
}
}while(optio
n !=7);
getch();
return 0;
}
```

```
*****MAIN MENU *****
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Delete a node from the beginning
6: Delete a node from the end
7: EXIT

Enter your option :3

Enter the data :0


*****MAIN MENU *****
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Delete a node from the beginning
6: Delete a node from the end
7: EXIT

Enter your option :4

Enter the data :4


*****MAIN MENU *****
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Delete a node from the beginning
6: Delete a node from the end
7: EXIT

Enter your option :
```

```
Enter your option :4

Enter the data :5


*****MAIN MENU *****
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Delete a node from the beginning
6: Delete a node from the end
7: EXIT

Enter your option :3

Enter the data :6


*****MAIN MENU *****
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Delete a node from the beginning
6: Delete a node from the end
7: EXIT

Enter your option :2
        6       4       5

*****MAIN MENU *****
1: Create a list
2: Display the list
3: Add a node at the beginning
4: Add a node at the end
5: Delete a node from the beginning
6: Delete a node from the end
7: EXIT
```

Experiment - 6

AIM - To implement circular queue using ADT using arrays.

Theory -

Circular arrays are very common ways of implementing array based queues.

- It is a regular array.
- We simply view it as being circular
- In a circular implementation the queue is considered to be full whenever the front of the queue immediately precedes the rear of the queue in the counter-clockwise direction.

i. Algorithm of enqueue

Step 1:- if (rear + 1) % max = front
then print "overflow"
goto Step 4

Step 2:- if front = -1 & rear = -1
set front = rear = 0.
else if rear = max - 1 & front != 0
set rear = 0
else
set rear = (rear + 1) % max.
or (end of if)

Step 3:- Set queue[rear] = val

Step 4:- Exit

Source code - Attached

Output screen:- Attached

Conclusion:- Successfully implement circular queue ADT using array in C

SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>
#define MAX 10
int queue[MAX];
int front=-1,rear=-1;
void insert(void);
int delete_element(void);
int peek(void);
void display(void);
int main()
{
	int option, val;
	clrscr();
	do
	{
		printf("\n ***** MAIN MENU *****");
		printf("\n 1. Insert an element\n 2. Delete an element\n 3. Peek\n 4. Display the
	queue\n 5. EXIT");
		printf("\n Enter your option : ");
		scanf("%d", &option);
		switch(option)
		{
			case 1: insert();
				break;
			case 2: val = delete_element();
				if(val!=-1)
				printf("\n The number deleted is : %d", val);
				break;
			case 3: val = peek();
				if(val!=-1)
```

```
                                        printf("\n The first value in queue is : %d", val);
                                        break;
                        case 4: display();
                                        break;
                }
        }while(option!=5);
        getch();
        return 0;
}
void insert()
{
        int num;
        printf("\n Enter the number to be inserted in the queue : ");
        scanf("%d", &num);
        if(front==0 && rear==MAX-1)
                printf("\n OVERFLOW");
        else if(front==-1 && rear==-1)
        {
                front=rear=0;
                queue[rear]=num;
        }
        else if(rear==MAX-1 && front!=0)
        {
                rear=0;
                queue[rear]=num;
        }
        else
        {
                rear++;
                queue[rear]=num;
        }
```

```
}
int delete_element()
{
	int val;
	if(front==-1 && rear==-1)
	{
		printf("\n UNDERFLOW");
		return -1;
	}
	val = queue[front];
	if(front==rear)
		 front=rear=-1;
	else
	{
		if(front==MAX-1)
		front=0;
	else
		front++;
	}
	return val;
}

int peek()
{
	if(front==-1 && rear==-1)
	{
		printf("\n QUEUE IS EMPTY");
		return -1;
	}
	else
	{
```

```
                return queue[front];
        }
}

void display()
{
        int i;
        printf("\n");
        if (front==-1 && rear==-1)
                printf ("\n QUEUE IS EMPTY");
        else
        {
                if(front<rear)
                        {
                                for(i=front;i<=rear;i++)
                                printf("\t %d", queue[i]);
                        }
                else
                        {
                                for(i=front;i<MAX;i++)
                                        printf("\t %d", queue[i]);
                                for(i=0;i<=rear;i++)
                                        printf("\t %d", queue[i]);
                        }
        }
}
```

OUTPUT SCREENS:

```
C:\TURBOC3\BIN>TC C:\TURBOC3\Projects\DSEXP6.c

***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option : 1

Enter the number to be inserted in the queue : 10

***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option : 1

Enter the number to be inserted in the queue : 20
```

```
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option : 2

The number deleted is : 10
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option : 3

The first value in queue is : 20
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option : _
```

```
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option : 4

        20      30      40      50
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option : 5
```

Experiment 7

ASSN - 9 To implement doubly linked list

Theory -

Doubly linked list is a variation of linked list in which navigation is possible in both ways, either forward or backwards, easily as compared to Singly linked list.

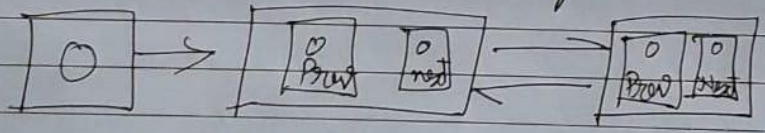
- Link - each link of a linked list can store a data

Next - Each link contains a link to the next node

Prev - Each link is attached to previous node

Linked list - Contains the connection link to the first link caused first & last link called list

Double linked list representation



Basic Operation

1. Insertion :- Adds element at the beginning of the list.
2. Deletion - Deletes the element from beginning of the list
3. Insert to last - Adds an element at the end of the list.
4. Delete last - Deletes an element at end of the list.
5. Delete forward - Displays the complete list forwards.
6. Display background :- Displays the complete list backwards.

Source Code :- Attached in

Output = At screens - Attached

Conclusion - Successfully implemented doubly linked list

Program

```
/*
 * C Program to Implement a Doubly Linked List & provide Insertion, Deletion & Display Operations
 */
#include  <stdio.h>
#include <stdlib.h>

struct node
{
   struct node *prev;
   int n;
   struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;

void insert1();
void insert2();
void insert3();
void traversebeg();
void traverseend(int);
void sort();
void search();
void update();
void delete();

int count = 0;

void main()
{
   int ch;
```

```
h = NULL;
temp = temp1 = NULL;

printf("\n 1 - Insert at beginning");
printf("\n 2 - Insert at end");
printf("\n 3 - Insert at position i");
printf("\n 4 - Delete at i");
printf("\n 5 - Display from beginning");
printf("\n 6 - Display from end");
printf("\n 7 - Search for element");
printf("\n 8 - Sort the list");
printf("\n 9 - Update an element");
printf("\n 10 - Exit");

while (1)
{
   printf("\n Enter choice : ");
   scanf("%d", &ch);
   switch (ch)
   {
   case 1:
      insert1();
      break;
   case 2:
      insert2();
      break;
   case 3:
      insert3();
```

```
    break;
case 4:
    delete();
    break;
case 5:
    traversebeg();
    break;
case 6:
    temp2 = h;
    if (temp2 == NULL)
        printf("\n Error : List empty to display ");
    else
    {
        printf("\n Reverse order of linked list is : ");
        traverseend(temp2->n);
    }
    break;
case 7:
    search();
    break;
case 8:
    sort();
    break;
case 9:
    update();
    break;
 case 10:
    exit(0);
default:
```

```
            printf("\n Wrong choice menu");
        }
    }
}

/* TO create an empty node */
void create()
{
    int data;

    temp =(struct node *)malloc(1*sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\n Enter value to node : ");
    scanf("%d", &data);
    temp->n = data;
    count++;
}

/* TO insert at beginning */
void insert1()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
```

```
    {
        create();
        temp->next = h;
        h->prev = temp;
        h = temp;
    }
}

/* To insert at end */
void insert2()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp1->next = temp;
        temp->prev = temp1;
        temp1 = temp;
    }
}

/* To insert at any position */
void insert3()
{
```

```
int pos, i = 2;

printf("\n Enter position to be inserted : ");
scanf("%d", &pos);
temp2 = h;

if ((pos < 1) || (pos >= count + 1))
{
   printf("\n Position out of range to insert");
   return;
}
if ((h == NULL) && (pos != 1))
{
   printf("\n Empty list cannot insert other than 1st
   position"); return;
}
if ((h == NULL) && (pos == 1))
{
   create();
   h = temp;
   temp1 = h;
   return;
}
else
{
   while (i < pos)
   {
      temp2 = temp2->next;
      i++;
```

```
        }
        create();
        temp->prev = temp2;
        temp->next = temp2->next;
        temp2->next->prev = temp;
        temp2->next = temp;
    }
}

/* To delete an element */
void delete()
{
    int i = 1, pos;

    printf("\n Enter position to be deleted : ");
    scanf("%d", &pos);
    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Error : Position out of range to delete");
        return;
    }
    if (h == NULL)
    {
        printf("\n Error : Empty list no elements to delete");
        return;
    }
    else
```

```
{
   while (i < pos)
   {
      temp2 = temp2->next;
      i++;
   }
   if (i == 1)
   {
      if (temp2->next == NULL)
      {
         printf("Node deleted from list");
         free(temp2);
         temp2 = h = NULL;
         return;
      }
   }
   if (temp2->next == NULL)
   {
      temp2->prev->next = NULL;
      free(temp2);
      printf("Node deleted from list");
      return;
   }
   temp2->next->prev = temp2->prev;
   if (i != 1)
      temp2->prev->next = temp2->next; /* Might not need this statement if i == 1 check */
   if (i == 1)
      h = temp2->next;
   printf("\n Node deleted");
```

```
        free(temp2);
    }
    count--;
}

/* Traverse from beginning */
void traversebeg()
{
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("List empty to display \n");
        return;
    }
    printf("\n Linked list elements from begining : ");

    while (temp2->next != NULL)
    {
        printf(" %d ", temp2->n);
        temp2 = temp2->next;
    }
    printf(" %d ", temp2->n);
}

/* To traverse from end recursively */
void traverseend(int i)
{
    if (temp2 != NULL)
```

```
    {
        i = temp2->n;
        temp2 = temp2->next;
        traverseend(i);
        printf(" %d ", i);
    }
}

/* To search for an element in the list */
void search()
{
    int data, count = 0;
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("\n Error : List empty to search for data");
        return;
    }
    printf("\n Enter value to search : ");
    scanf("%d", &data);
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {
            printf("\n Data found in %d position",count + 1);
            return;
        }
        else
```

```
            temp2 = temp2->next;
            count++;
    }
    printf("\n Error : %d not found in list", data);
}

/* To update a node value in the list */
void update()
{
    int data, data1;

    printf("\n Enter node data to be updated : ");
    scanf("%d", &data);
    printf("\n Enter new data : ");
    scanf("%d", &data1);
    temp2 = h;
    if (temp2 == NULL)
    {
        printf("\n Error : List empty no node to update");
        return;
    }
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {

            temp2->n = data1;
            traversebeg();
            return;
```

```
        }
        else
            temp2 = temp2->next;
    }

    printf("\n Error : %d not found in list to update", data);
}

/* To sort the linked list */
void sort()
{
    int i, j, x;

    temp2 = h;
    temp4 = h;

    if (temp2 == NULL)
    {
        printf("\n List empty to sort");
        return;
    }

    for (temp2 = h; temp2 != NULL; temp2 = temp2->next)
    {
        for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)
        {
            if (temp2->n > temp4->n)
            {
                x = temp2->n;
```

```
            temp2->n = temp4->n;
            temp4->n = x;
          }
        }
    }
    traversebeg();
}
```

Output

input

```
7 - Search for element
8 - Sort the list
9 - Update an element
10 - Exit
Enter choice : 1

Enter value to node : 4

Enter choice : 1

Enter value to node : 3

Enter choice : 2

Enter value to node : 7

Enter choice : 3

Enter position to be inserted : 3

Enter value to node : 9

Enter choice : 5

Linked list elements from begining :  3  4  9  7
Enter choice : 6

Reverse order of linked list is :  7  9  4  3
Enter choice :
```

input

```
7 - Search for element
8 - Sort the list
9 - Update an element
10 - Exit
Enter choice : 1

Enter value to node : 2

Enter choice : 1

Enter value to node : 3

Enter choice : 2

Enter value to node : 6

Enter choice : 2

Enter value to node : 4

Enter choice : 7

Enter value to search : 3

Data found in 1 position
Enter choice : 8

Linked list elements from begining :  2  3  4  6
Enter choice :
```


Enter choice : 1
Enter value to node : 3
Enter choice : 2
Enter value to node : 6
Enter choice : 2
Enter value to node : 4
Enter choice : 7
Enter value to search : 3
Data found in 1 position
Enter choice : 8
Linked list elements from beginning : 2 3 4 6
Enter choice : 9
Enter node data to be updated : 2
Enter new data : 0
Linked list elements from beginning : 0 3 4 6
Enter choice :

Ex- 8

AIM - To implement stack & circular queue ADT using linked list

THEORY -

Stack using linked list

A stack data structure can be implemented by using a linked list data structure. The stack implemented ~~no number of~~ using linked list can work ~~for~~ an unlimited number ~~of~~ values. That means stack implemented using linked list works ~~for~~ the variable size ~~of~~ data. So there is no need to pick the size at the beginning ~~of~~ the implementation. The stack implemented using linked list can ~~be~~ organise as many data values as we want.

Stack operations

1. Include all header files
2. Node ADT structure, node top pointer to NULL
3. Implement main.

Algorithms

1. Push (value)

new node

if (empty/(top == NULL))

{

set new node -> next = NULL

}

else new node -> next top

top = new node.

2. POP (value)

```
if (empty/(top == NULL))
{
    display ("stack is empty");
}
else{
    node -> temp = top;
    set top = top -> next
    temp
}
```

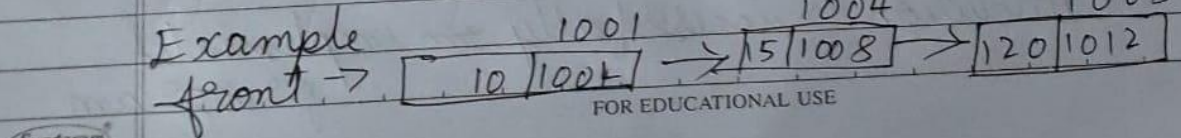
3. Display ()

```
if (empty)
    printf ("stack is empty");
else
    top temp = top
    printf (temp -> data)
    until
    temp -> next != NULL
```

Linear ~~Que~~ Queues using linked list

The queue which is implemented using a linked list can work for an unlimited number of variables. The queue using linked list can organise as many data as we want.

Example



Operations

1. Include all header files
2. Define node structure with 2 data & next
3. Define two node pointers front & rear and & set both NULL.

Algorithms

1. enQueue(value)

Create new node
if (empty)
    set front = new node
    rear = new Node
else
    rear -> next = new Node
    rear = new node

2. Queue(value)
if(empty)
    printf ("queue is empty")
else
temp -> front, front = front -> next

Source Code - Attached in document
Output Screens Attached in document

Conclusion :- Successfully implemented program.

Program

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
   int info;
   struct node *ptr;
}*front,*rear,*temp,*front1;

int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();

int count = 0;

void main()
{
   int no, ch, e;

   printf("\n 1 - Enque");
   printf("\n 2 - Deque");
   printf("\n 3 - Front element");
```

```
printf("\n 4 - Empty");
printf("\n 5 - Exit");
printf("\n 6 - Display");
printf("\n 7 - Queue size");
create();
while (1)
{
   printf("\n Enter choice : ");
   scanf("%d", &ch);
   switch (ch)
   {
   case 1:
      printf("Enter data : ");
      scanf("%d", &no);
      enq(no);
      break;
   case 2:
      deq();
      break;
   case 3:
      e = frontelement();
      if (e != 0)
         printf("Front element : %d", e);
      else
         printf("\n No front element in Queue as queue is
      empty"); break;
   case 4:
```

```
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            queuesize();
            break;
        default:
            printf("Wrong choice, Please enter correct choice ");
            break;
        }
    }
}

/* Create an empty queue */
void create()
{
    front = rear = NULL;
}

/* Returns queue size */
void queuesize()
{
    printf("\n Queue size : %d", count);
```

```
}

/* Enqueing the queue */
void enq(int data)
{
   if (rear == NULL)
   {
      rear = (struct node *)malloc(1*sizeof(struct node));
      rear->ptr = NULL;
      rear->info = data;
      front = rear;
   }
   else
   {
      temp=(struct node *)malloc(1*sizeof(struct node));
      rear->ptr = temp;
      temp->info = data;
      temp->ptr = NULL;

      rear = temp;
   }
   count++;
}

/* Displaying the queue elements */
void display()
{
```

```
    front1 = front;

    if ((front1 == NULL) && (rear == NULL))
    {
        printf("Queue is empty");
        return;
    }
    while (front1 != rear)
    {
        printf("%d ", front1->info);
        front1 = front1->ptr;
    }
    if (front1 == rear)
        printf("%d", front1->info);
}

/* Dequeing the queue */
void deq()
{
    front1 = front;

    if (front1 == NULL)
    {
        printf("\n Error: Trying to display elements from empty
        queue"); return;
    }
    else
```

```
        if (front1->ptr != NULL)
        {
            front1 = front1->ptr;
            printf("\n Dequed value : %d", front->info);
            free(front);
            front = front1;
        }
        else
        {
            printf("\n Dequed value : %d", front->info);
            free(front);
            front = NULL;
            rear = NULL;
        }
        count--;
}

/* Returns the front element of queue */
int frontelement()
{
    if ((front != NULL) && (rear != NULL))
        return(front->info);
    else
        return 0;
}

/* Display if queue is empty or not */
```

```
void empty()
{
    if ((front == NULL) && (rear == NULL))
       printf("\n Queue empty");
   else
      printf("Queue not empty");
}
```

Output

Experiment -9

AIM- To implement binary search tree using linked list.

Theory-

Binary search tree is a node based binary tree data structure which as the following properties:-

i. The left subtree of a node contains only nodes with keys lesser than the nodes key

ii. The right subtree of a node contains only nodes with keys greater than the nodes key.

iii. The left and right subtree must also be a binary search tree.

Algorithm

1. Create a root node & initialize the left & right child as NULL.
2. Take input for inserting new node.
3. If (x > root->data), insert the node as right child.
4. Else if (x <= root->data), insert the node as the left child.
5. If root == NULL, insert x as new node.

For deleting a node
1. Take the input of x to be deleted
2. If x > root -> data go down inside right tree
3. If x < root -> data go down inside left tree
4. If the root node has no child, simple tree that root.
5. If one child delete it

Source Code - Attached in document

Output Screen - Attached in document

Conclusion - Succesfully implemented program.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{ int data; struct
node *left; struct
node *right; };

struct node *new_node(int d)
{ struct node *new;
new=(struct node*)malloc(sizeof(struct node));
new->data=d;
new->left=NULL;
new->right=NULL;
return(new);
}
struct node *create(struct node *root,int k)
{ if(root==NULL)
{ return(new_node(k));
}
else if(root->data>k)
{ root->left=create(root->left,k);
}
else
{ root->right=create(root->right,k);
}
return(root);
}
int preorder(struct node *root)
{ if(root!=NULL)
{ printf("%d\t",root->data);
preorder(root->left);
preorder(root->right);
}
}
int inorder(struct node
*root) { if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->data);
inorder(root->right);
}
}
int postorder(struct node *root)
{ if(root!=NULL)
{
postorder(root->left);
postorder(root->right);
printf("%d\t",root->data);
}
}
struct node *minvalue(struct node
*root) { struct node *curr=root;
while(curr!=NULL && curr->left!=NULL)
{ curr=curr->left;
}
return (curr);
```

```
}
struct node *delete(struct node *root,int k)
{ if(k<root->data)
{ root->left=delete(root->left,k);
}
else if(k>root->data)
{ root->right=delete(root->right,k);
}
else
{ 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;
}
struct node *temp=minvalue(root->right);
root->data=temp->data; root-
>right=delete(root->right,temp->data);
}
return (root);
}
void main()
{ struct node *root=NULL;
int ele,a,b,m;
printf("Enter the first element\n");
scanf("%d",&ele);
root=create(root,ele);
printf("1.CREATION\n");
printf("2.INORDER TRAVERSAL\n");
printf("3.PREORDER TRAVERSAL\n");
printf("4.POSTORDER TRAVERSAL\n");
printf("5.DELETION\n");
int flag=0;
while(flag!=1)
{
printf("\nEnter your choice\n");
scanf("%d",&a);
switch(a)
{ case 1:
printf("Enter the element\n");
scanf("%d",&b);
create(root,b);
break;
case 2:
inorder(root);
break;
case 3:
preorder(root);
break;
case 4:
postorder(root);
break;
```

```
case 5:
printf("Enter the data to be deleted\n");
scanf("%d",&m);
root=delete(root,m);
break;
default: flag=1;
break;
}
}
}
```

```
Enter the first element
2
1.CREATION
2.INORDER TRAVERSAL
3.PREORDER TRAVERSAL
4.POSTORDER TRAVERSAL
5.DELETION

Enter your choice
1
Enter the element
1

Enter your choice
1
Enter the element
5

Enter your choice
1
Enter the element
4

Enter your choice
1
Enter the element
8

Enter your choice
1
Enter the element
9

Enter your choice
2
1       2       4       5       8       9
Enter your choice
3
2       1       5       4       8       9
Enter your choice
4
1       4       9       8       5       2
Enter your choice
5
Enter the data to be deleted
1
```

Experiment 10

AIM- To implement graph traversed techniques
1. Depth First Search
2. Breadth First Search

Theory -
Graph Traversals

The graph is a non-linear data structure. This consists of nodes & edges.

It has 2 traversals

1. Depth First Search

In this algo one starting vertex is given & when an adjacent vertex is given found it moves to that vertex & do it again.

Algo
dfs(vertex vertice, start)
Input: The list of all vertices & the start node

Output:- Traverse all nodes in the graph

Begin:- initially move to the state to unvisited in all nodes.
push start into stack
while stack is not empty, do
pop element from stack & set it u.

display node u
if u is not visited then
mark us as visited

for all nodes 'i' connected to u do
if the vertex is unvisited then push
it to stack.

BFS
It is used to visit all the nodes of given graph. One node is selected & all the adjacent nodes visited one by one. After completing all the adjacent vertices.

Algo
Input:- The list of vertices & the start vertex.
Output:- Traverse all the node of graph is connected

Source Code:- as visited

Output screen - Attached in document

Conclution - Successfully implemented program

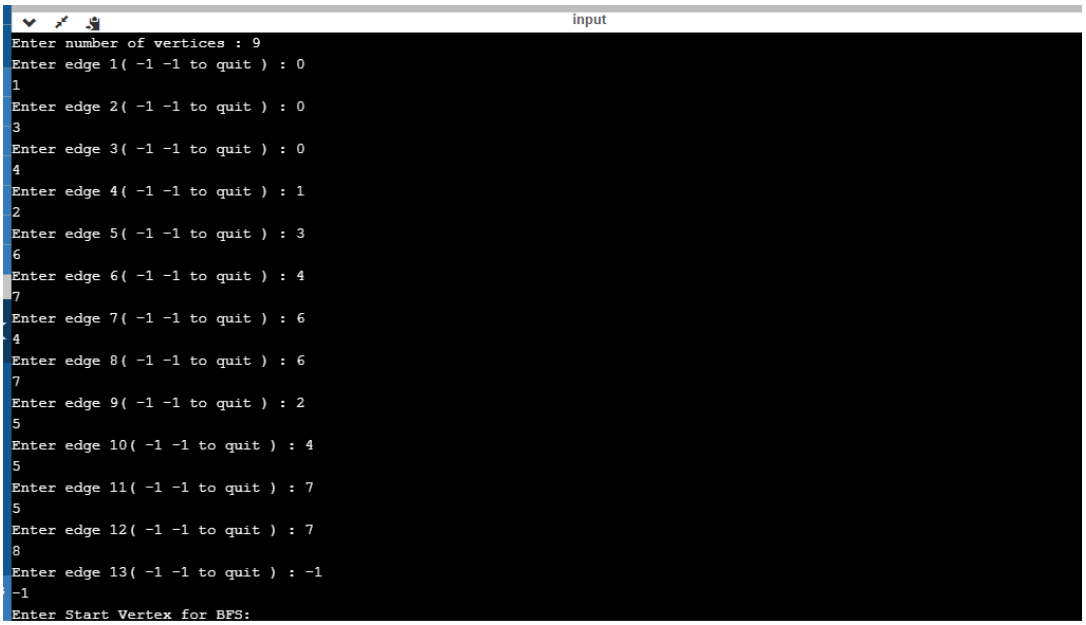
Program

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
        create_graph();
        BF_Traversal();
        return 0;
}

void BF_Traversal()
{
        int v;

        for(v=0; v<n; v++)
                state[v] = initial;

        printf("Enter Start Vertex for BFS: \n");
        scanf("%d", &v);
        BFS(v);
}

void BFS(int v)
{
        int i;
```

```
        insert_queue(v);
        state[v] = waiting;

        while(!isEmpty_queue())
        {
                v = delete_queue( );
                printf("%d ",v);
                state[v] = visited;

                for(i=0; i<n; i++)
                {
                        if(adj[v][i] == 1 && state[i] == initial)
                        {
                                insert_queue(i);
                                state[i] = waiting;
                        }
                }
        }
        printf("\n");
}

void insert_queue(int vertex)
{
        if(rear == MAX-1)
                printf("Queue Overflow\n");
        else
        {
                if(front == -1)
                        front = 0;
                rear = rear+1;
                queue[rear] = vertex ;
        }
}

int isEmpty_queue()
{
        if(front == -1 || front > rear)
                return 1;
        else
                return 0;
}

int delete_queue()
{
        int delete_item;
        if(front == -1 || front > rear)
```

```
        {
                printf("Queue Underflow\n");
                exit(1);
        }

        delete_item = queue[front];
        front = front+1;
        return delete_item;
}

void create_graph()
{
        int count,max_edge,origin,destin;

        printf("Enter number of vertices : ");
        scanf("%d",&n);
        max_edge = n*(n-1);

        for(count=1; count<=max_edge; count++)
        {
                printf("Enter edge %d( -1 -1 to quit ) : ",count);
                scanf("%d %d",&origin,&destin);

                if((origin == -1) && (destin == -1))
                        break;

                if(origin>=n || destin>=n || origin<0 || destin<0)
                {
                        printf("Invalid edge!\n");
                        count--;
                }
                else
                {
                        adj[origin][destin] = 1;
                }
        }
}
```

Output


input
Enter number of vertices : 9
Enter edge 1(-1 -1 to quit) : 0
1
Enter edge 2(-1 -1 to quit) : 0
3
Enter edge 3(-1 -1 to quit) : 0
4
Enter edge 4(-1 -1 to quit) : 1
2
Enter edge 5(-1 -1 to quit) : 3
6
Enter edge 6(-1 -1 to quit) : 4
7
Enter edge 7(-1 -1 to quit) : 6
4
Enter edge 8(-1 -1 to quit) : 6
7
Enter edge 9(-1 -1 to quit) : 2
5
Enter edge 10(-1 -1 to quit) : 4
5
Enter edge 11(-1 -1 to quit) : 7
5
Enter edge 12(-1 -1 to quit) : 7
8
Enter edge 13(-1 -1 to quit) : -1
-1
Enter Start Vertex for BFS:


Enter Start Vertex for BFS:
0
0 1 3 4 2 6 5 7 8
...Program finished with exit code 0
Press ENTER to exit console.

Program

```
#include<stdio.h>
#include<stdlib.h>

typedef struct node
{
   struct node *next;
   int vertex;
}node;

node *G[20];
//heads of linked list
int visited[20];
int n;
void read_graph();
//create adjacency list
void insert(int,int);
//insert an edge (vi,vj) in te adjacency list
void DFS(int);

void main()
{
   int i;
   read_graph();
   //initialised visited to 0

         for(i=0;i<n;i++)
       visited[i]=0;

   DFS(0);
}

void DFS(int i)
{
   node *p;

         printf("\n%d",i);
   p=G[i];
   visited[i]=1;
   while(p!=NULL)
   {
      i=p->vertex;

           if(!visited[i])
          DFS(i);
```

```
        p=p->next;
    }
}

void read_graph()
{
    int i,vi,vj,no_of_edges;
    printf("Enter number of vertices:");

          scanf("%d",&n);

    //initialise G[] with a null

          for(i=0;i<n;i++)
    {
        G[i]=NULL;
        //read edges and insert them in G[]

                  printf("Enter number of edges:");
          scanf("%d",&no_of_edges);

          for(i=0;i<no_of_edges;i++)
        {
          printf("Enter an edge(u,v):");
                          scanf("%d%d",&vi,&vj);
                          insert(vi,vj);
        }
    }
}

void insert(int vi,int vj)
{
    node *p,*q;

          //acquire memory for the new node
          q=(node*)malloc(sizeof(node));
    q->vertex=vj;
    q->next=NULL;

    //insert the node in the linked list number vi
    if(G[vi]==NULL)
        G[vi]=q;
    else
    {
        //go to end of the linked list
        p=G[vi];
```

```
                while(p->next!=NULL)
            p=p->next;
          p->next=q;
    }
}
```

Output

Experiment 11

AIM- To perform application of Binary Search Technique

Theory-
Binary search is a searching algorithm that works efficiently with a sorted list. It reduces time required.

Example:-

0	1	2	3	4	5	6
10	20	30	40	50	60	70

lets say we need to find 60
low = 0, high = 6

mid = (low + high) / 2 = 3

is x[mid] == number ?
is x[mid] < number ?
is x[mid] > number ?

Algo
Step 1: low = 0, high = n-1 ; (n = length of array)

Step 2: while (low <= high)
if (x[mid] < key)
low = mid + 1;
else if (x[mid] > key)
high = mid;
else
return x[mid];

Step 3:- Print x[mid] and mid

Source Code:- Attached

Output Code:- Attached

Conclution:- Successfully implemented binary search using C.

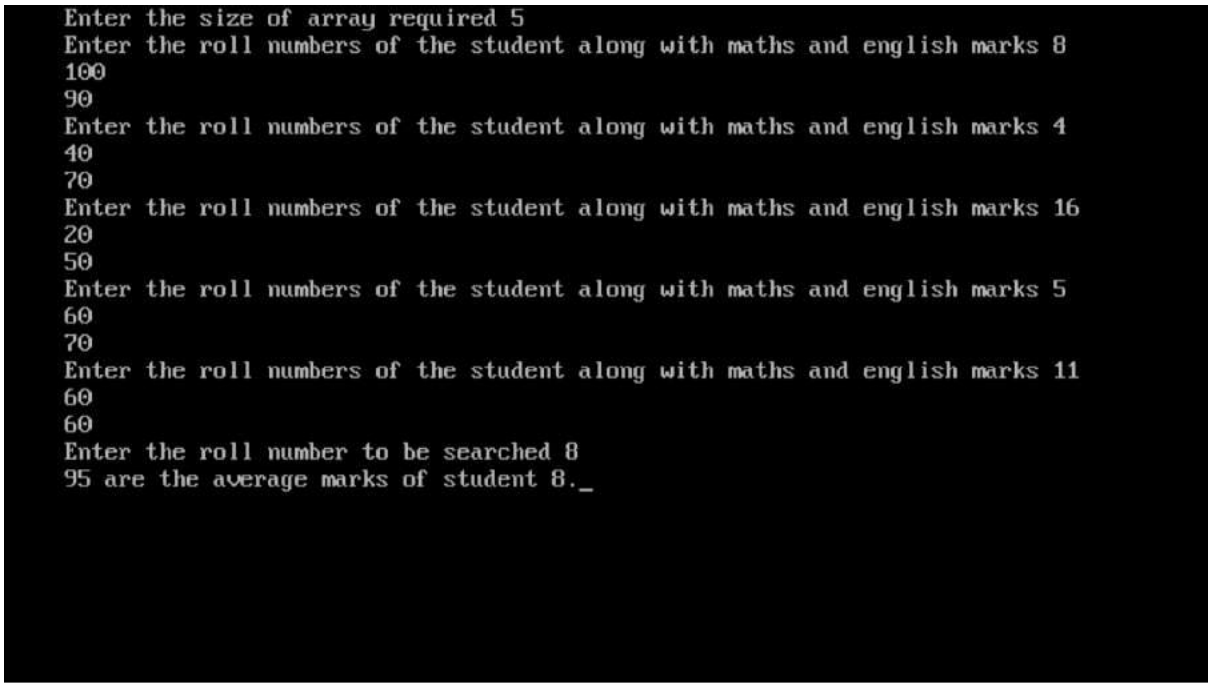
SOURCE CODE

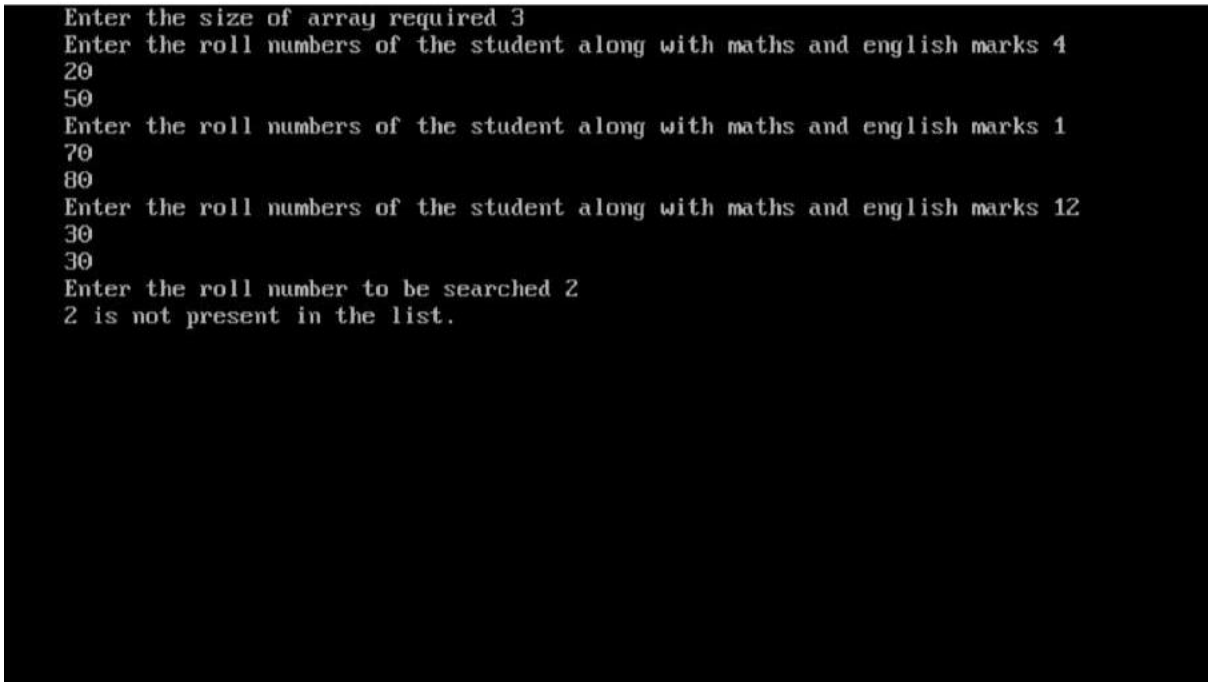
```
#include<stdio.h>
#include<conio.h>
#define MAX 100
void bubble(int arr[],int m[],int n)
{
int i, tmp, j, temp;
for(i=0; i<n; i++)
{
for(j=i+1; j<n; j++)
{
if(arr[i]>arr[j])
{
tmp=arr[i];
temp=m[i];
arr[i]=arr[j];
m[i]=m[j];
arr[j]=tmp;
m[j]=temp;
}
}
}
}
main()
{
int a[MAX], marks[MAX], temp[MAX];
int num, low=0, high, mid, i, size, pos=-1;
clrscr();
printf("Enter the size of array required ");
scanf("%d",&size);
for(i=0; i<size; i++)
```

```
{
printf("Enter the roll numbers of the student along with maths and english marks ");
scanf("%d%d%d",&a[i], &temp[i], &marks[i]);
marks[i]=(marks[i]+temp[i])/2;
}
bubble(a, marks, size);
printf("Enter the roll number to be searched ");
scanf("%d",&num);
high=size;
while(low<=high)
{
mid=(low+high)/2;
if(a[mid]==num)
{
pos=mid;
break;
}
else if(a[mid]>num)
{
high=mid-1;
}
else if(a[mid]<num)
{
low=mid+1;
}
}
if(pos==-1)
{printf("%d is not present in the list.", num);}
else
{printf("%d are the average marks of student %d.", marks[pos], num);}
```

```
getch();
}
```

OUTPUT SCREENS:





Extra Assignment:
AVL Tree Code:

```
#include<stdio.h>
#include<stdlib.h>

struct Node
{     int key;     struct Node
*left;     struct Node *right;
int height; };
int max (int a, int b){
   return (a>b)?a:b;
}

int getHeight(struct Node *n){     if(n==NULL)
       return 0;
   return n->height;
}

struct Node *createNode(int key){
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));     node-
>key=key;     node->left=NULL;     node->right=NULL;
   node->height=1;

   return node;
}


int getBalanceFactor(struct Node *n){     if(n==NULL){
       return 0;
   }
   return getHeight(n->left)- getHeight(n->right);
}

struct Node *RightRotate(struct Node * y){
   struct Node* x= y->left;     struct Node*
T2= x->right;     x->right=y;
   y->left=T2;

   y->height=max(getHeight(y->right), getHeight(y->left))+1;     x-
>height=max(getHeight(x->right), getHeight(x->left))+1;

   return x;
}
struct Node *LeftRotate(struct Node * x){
   struct Node* y= x->right;     struct Node*
T2= y->left;
   y->left=x;
   x->right=T2;
```

```
   y->height=max(getHeight(y->right), getHeight(y->left))+1;     x-
>height=max(getHeight(x->right), getHeight(x->left))+1;

   return y;
}
struct Node *insert(struct Node * node, int key){

   if(node== NULL){
      return createNode(key);
   }
   if(key<node->key){
     node->left= insert(node->left, key);
   }
   else if(key> node->key){
      node->right = insert(node->right, key);
   }
   node->height = 1 + max(getHeight(node->left), getHeight(node->right));     int bf =
getBalanceFactor(node);
   // Left Left Case
   if(bf>1 && key< node->left->key){
     return RightRotate(node);
   }
   // Right Right Case     if(bf<-1 && key> node-
>right->key){        return LeftRotate(node);
   }
   // Left Right Case     if(bf>1 && key > node->left-
>key){             node->left = LeftRotate(node->left);
          return RightRotate(node);
       }
   // Right Left Case
   if(bf<-1 && key < node->right->key){             node->right =
RightRotate(node->right);             return LeftRotate(node);
       }
   return node;
 }
void preOrder(struct Node *root)
{
   if(root != NULL)
   {
      printf("%d ", root->key);         preOrder(root->left);
      preOrder(root->right);
   }
}
int main(void){
   struct Node * root = NULL;

   root = insert(root, 1);     root =
insert(root, 2);

   root = insert(root, 4);
preOrder(root);     printf("\n");
root = insert(root, 5);
```

```
preOrder(root);     printf("\n");
root = insert(root, 6);
preOrder(root);     printf("\n");
root = insert(root, 3);
preOrder(root);     return 0; }
```

Output:

```
1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:1

Enter no. of elements:3

Enter tree data:4
3
5

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:2

Enter a data:10

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:4
```

```
Enter a data:10

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:4

Preorder sequence:
4(Bf=-1)3(Bf=0)5(Bf=-1)10(Bf=0)

Inorder sequence:
3(Bf=0)4(Bf=-1)5(Bf=-1)10(Bf=0)

1)Create:
2)Insert:
3)Delete:
4)Print:
5)Quit:

Enter Your Choice:
```

Prujansh Salian

Data Structures

Theory Assignment-1

1. Differentiate between linear and non-linear data structure

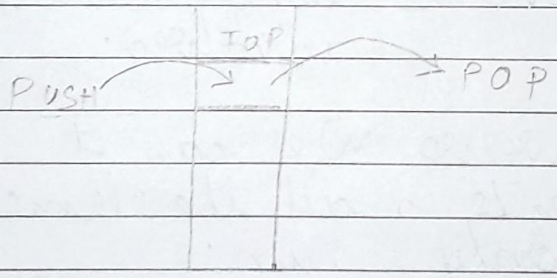
Ans:-

Linear data structure	Non-linear data structure
• Data structures where data elements are arranged sequentially or linearly	Data structures where data elements are not arranged sequentially or linearly.
• In linear data structure single level is involved	In non-linear data structure single level is not involved.
• We can traverse all the elements in single run only	We can't traverse all the elements in single run.
• Easy to implement	Not easy to implement.
• Inefficient memory allocation	Efficient memory allocation
• Ex- Array, stack, queue & linked list, etc.	Ex- trees and graphs

2. Explain ADT of stack

Ans: Stack is a linear data structure in which the inserting & deletion operations are performed at only one end. In a stack, adding & removing of elements are performed at a single position which is known as "top".

In stack, the insertion & deletion operations are performed based on LIFO (Last in First Out) principle. This insertion operation is performed using a function called "push" & deletion operation is performed using a function called "pop".

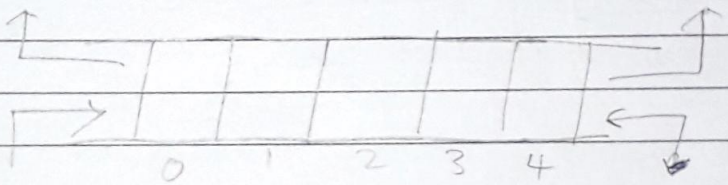


The common operations on a stack are
1. PUSH (To insert an element in the stack)
2. POP (To delete an element from stack)
3. PEEK (To display the element on top)
4. DISPLAY (To display element of the stack

Stack can be implemented using Array or linked list.

4. Write a short ~~not~~ note on double ended queue.

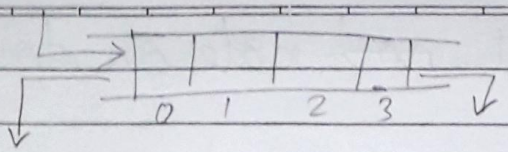
Ans: The double ended queue is also known as dequeue. In a queue, the insertion takes place from one end while the deletion takes place from another end. Dequeue is a linear data structure in which the insertion & deletion operations are performed from both ends.



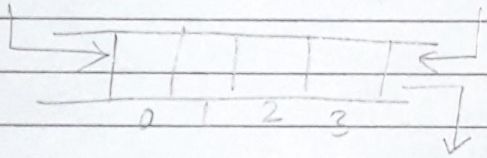
Dequeue can be used both as stack and queue as it allows the insertion & deletion operation on both sides.

There are two types ~~of queues~~ queues, Input restricted queue & output restricted queue.

1. Input restricted queue: ~~The input~~ This means that some restrictions are applied to the insertion. In input restricted queue, the insertions is applied to one end ~~but~~ while the deletion is appled from both the ends.



2. Output restricted queue - The output restricted queue means that some restrictions are ~~applied~~ applied to the deletion operation. In an output-restricted queue, the deletion can be applied only from one end, whereas the insertion is possible from both the ends.

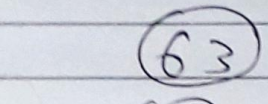


Priyansh Salian

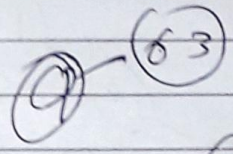
Assignment 2

1. 63, 9, 19, 18, 108, 99, 81, 45, 12, 106

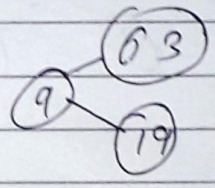
Step 1 :- Insert 63



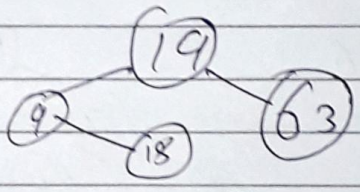
Step 2 Insert 9



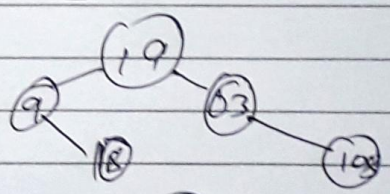
Step 3 Insert 19



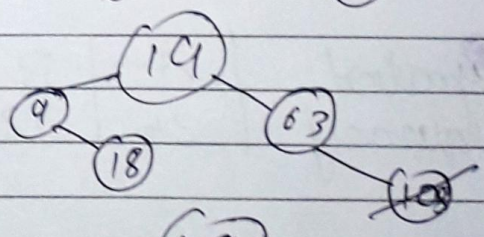
Step 4 Insert 18



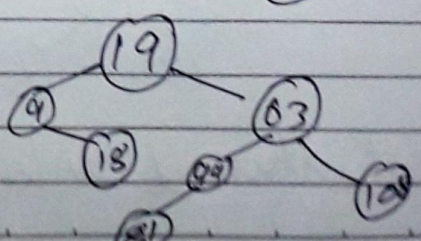
Step 5 Insert 108



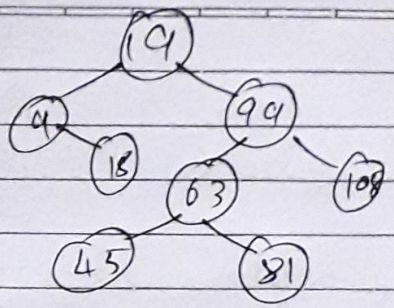
Step 6 Insert 99



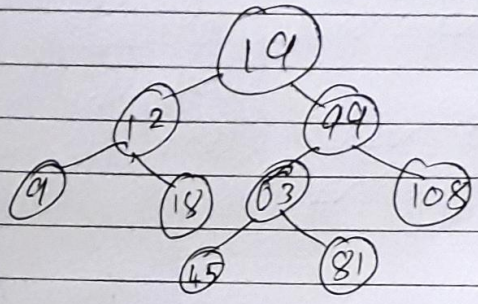
Step 7 Insert 81



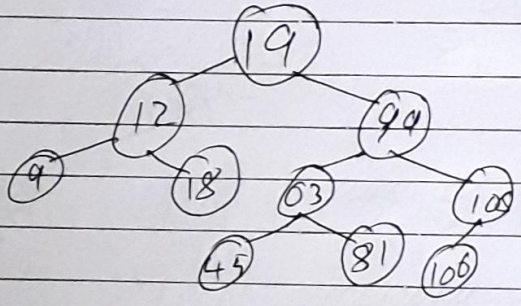
Step 8: Insert 45



Step 9 Insert 12



~~Insert 1~~
Step 10 Insert 100

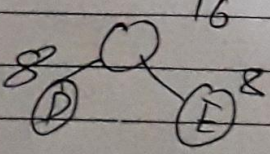


2.

Ans:

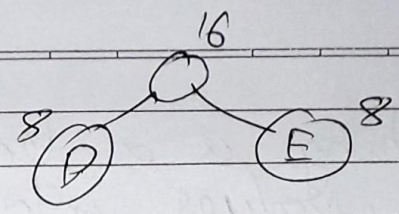
Symbol	A	B	C	D	E
Frequency	24	12	10	8	8

Step 1 Select D & E & and form a tree



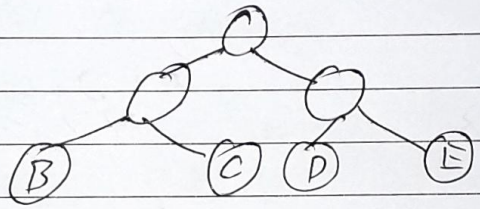
Insert this into the priority queue

A B C



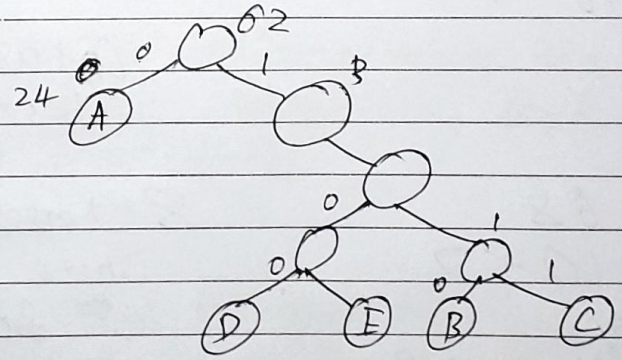
Step 2: Delete B & C and insert back into priority queue

Step 3: Delete (B-C) & (D-E) from the tree



Insert them in priority queue

Step 4: Delete (A) & [D-E and B-C]



Symbol	Frequency	Code
A	24	0
B	12	110
C	10	111
D	8	100
E	8	101

3.

Ans: Hashing is a technique of mapping keys, values into hash table by using a hash function. It is done for faster to access to elements.

Size = 10

63, 82, 94, 77, 53, 87, 23, 35, 10, 44

1. Insert 63
   = 63 % 10 = 3

2. Insert 82
   = 82 % 10 = 2

3. Insert 94
   94 % 10 = 4

4. Insert 77
   77 % 10 = 7

5. Insert 53
   53 % 10 = 3
   Collision here at 3
   (53+1) % 10
   = 4
   Collision again
   (53+2) % 10
   = 5

6. Insert 87
   = 87 % 10 = 7
   Collision
   (87+1) % 10 = 8

7. Insert 23
   23 % 10 = 3
   collision
   24 % 10 = 4
   Collision, 25 % 10 = 5
   collision, 26 % 10 = 6

8. Insert 35
   There is no collision at 8 so
   35 % 10 = 9

9. Insert 10
   10 % 10 = 0

10. Insert 44
    44 % 10 = 4

5. i Topological sorting for directed acyclic graph is a linear ordering vertices such that every edge u, v - vertex comes before v in ordering

ii. It is not possible if graph is not DAG

iii. There can be more than one topological sorting for a graph

iv. This first vertex in topological sorting is always a vertex within indegree as 0

v. This is used in data sterlization

vi. This is used in instruction sheduling

vii. This is used in determining the order of compilation tasks compiling task to perform in makefiles.

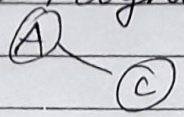
4.

Ans Post Order - D E F B G L J K H L A
In Order - D B F E G A G C L J H K

i. From the post order +ve last element is the node element since its L R V treatment.

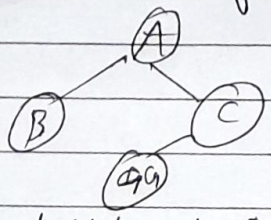
(A)

ii. C will be the right child of A

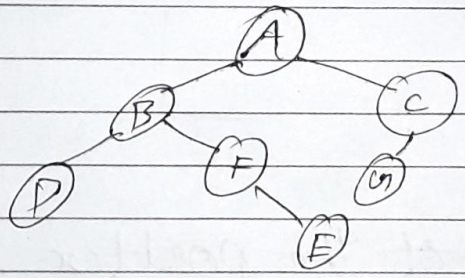


~~iii. After~~

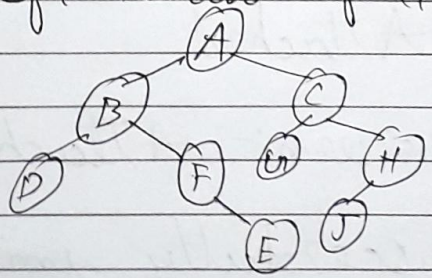
iii. B is left child of C & B is left child of A



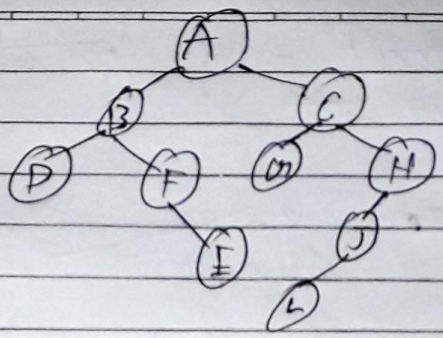
iv. D is left child of B and F & E are right child of B & F respectively



v. J is left child of H



vi. L is left child of J



11. K is right child of H