

**Practical:1**

**AIM:** 1A) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).

**PROGRAM CODE:**

```
#include <stdio.h>

#include <string.h>

// Define a structure to represent a day
struct Day {
    char name[20];
    int date;
    char activity[100];
};

int main() {
    // Declare an array of 7 elements to represent the calendar
    struct Day calendar[7];

    // Initialize the calendar with sample data
    strcpy(calendar[0].name, "Monday");
    calendar[0].date = 1;
    strcpy(calendar[0].activity, "Work from 9 AM to 5 PM");
    strcpy(calendar[1].name, "Tuesday");

    calendar[1].date = 2;
    strcpy(calendar[1].activity, "Meeting at 10 AM");
    strcpy(calendar[2].name, "Wednesday");
    calendar[2].date = 3;
    strcpy(calendar[2].activity, "Gym at 6 PM");
    strcpy(calendar[3].name, "Thursday");
```

```
calendar[3].date = 4;
strcpy(calendar[3].activity, "Dinner with friends at 7 PM");
strcpy(calendar[4].name, "Friday");
calendar[4].date = 5;
strcpy(calendar[4].activity, "Movie night at 8 PM");
strcpy(calendar[5].name, "Saturday");
calendar[5].date = 6;
strcpy(calendar[5].activity, "Weekend getaway");
strcpy(calendar[6].name, "Sunday");
calendar[6].date = 7;
strcpy(calendar[6].activity, "Relax and recharge");

// Print the calendar  printf("Calendar for the week:\n");
for (int i = 0; i < 7; i++) {
    printf("%s (Date: %d): %s\n", calendar[i].name, calendar[i].date, calendar[i].activity);
}

    return 0;
}
```

**OUTPUT:**

```
Calendar for the week:
Monday (Date: 1): Work from 9 AM to 5 PM
Tuesday (Date: 2): Meeting at 10 AM
Wednesday (Date: 3): Gym at 6 PM
Thursday (Date: 4): Dinner with friends at 7 PM
Friday (Date: 5): Movie night at 8 PM
Saturday (Date: 6): Weekend getaway
Sunday (Date: 7): Relax and recharge
```

**AIM:** 1B) Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.

**PROGRAM CODE:**

```
#include <stdio.h>

#include <string.h>

// Define a structure to represent a day
struct Day {
    char name[20];
    int date;
    char activity[100];
};

// Function to create the calendar
void create(struct Day calendar[7]) {
    for (int i = 0; i < 7; i++) {
        printf("Enter details for %s:\n", calendar[i].name);
        printf("Date: ");
        scanf("%d", &calendar[i].date);
        printf("Activity: ");
        scanf(" %[^\n]", calendar[i].activity);
    }
}

// Function to read data from the keyboard
void read(struct Day calendar[7]) {
    FILE *file = fopen("calendar.txt", "r");
    if (file == NULL) {
        printf("Error opening the file.\n");
        return;
    }
}
```

```
}  
for (int i = 0; i < 7; i++) {  
    fscanf(file, "%d", &calendar[i].date);  
    fscanf(file, " %[^\\n]", calendar[i].activity);  
}  
fclose(file);  
}  
  
// Function to display the calendar  
void display(struct Day calendar[7]) {  
    printf("Calendar for the week:\\n");  
    for (int i = 0; i < 7; i++) {  
        printf("%s (Date: %d): %s\\n", calendar[i].name, calendar[i].date, calendar[i].activity);  
    }  
}  
  
int main() {  
    struct Day calendar[7];  
    // Initialize the names of the days  
    strcpy(calendar[0].name, "Monday");  
    strcpy(calendar[1].name, "Tuesday");  
    strcpy(calendar[2].name, "Wednesday");  
    strcpy(calendar[3].name, "Thursday");  
    strcpy(calendar[4].name, "Friday");  
    strcpy(calendar[5].name, "Saturday");  
    strcpy(calendar[6].name, "Sunday");  
  
    int choice;  
    printf("1. Create Calendar\\n");  
    printf("2. Read Calendar from File\\n");
```

```
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:    create(calendar);
           break;
case 2:    read(calendar);
           break;
default:
           printf("Invalid choice.\n");
           return 1;
}
display(calendar);
return 0;
}
```

**OUTPUT:**

```
1. Create Calendar
2. Read Calendar from File
Enter your choice: 1
Enter details for Monday:
Date: 10
Activity: Work from home
Enter details for Tuesday:
Date: 11
Activity: Work 9am to 5pm from office
Enter details for Wednesday:
Date: 12
Activity: Outdoor activity
Enter details for Thursday:
Date: 12
Activity: Office work
Enter details for Friday:
Date: 13
Activity: Office wotk
Enter details for Saturday:
Date: 14
Activity: Meeting at 10 AM
Enter details for Sunday:
Date: Relax and Reachrge
Activity: Calendar for the week:
Monday (Date: 10): Work from home
Tuesday (Date: 11): Work 9am to 5pm from office
Wednesday (Date: 12): Outdoor activity
Thursday (Date: 12): Office work
Friday (Date: 13): Office wotk
Saturday (Date: 14): Meeting at 10 AM
Sunday (Date: 1995719452): Relax and Reachrge
```

**Practical:2**

**AIM:** Develop a Program in C for the following operations on Strings.

- a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
- b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR

Support the program with functions for each of the above operations. Don't use Built-in functions.

**PROGRAM CODE:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
char STR[100],PAT[100],REP[100],ans[100];
int i,j,c,m,k,flag=0;
printf("\nEnter the MAIN string: \n"); gets(STR);
printf("\nEnter a PATTERN string: \n"); gets(PAT);
printf("\nEnter a REPLACE string: \n"); gets(REP);
i = m = c = j = 0;
while ( STR[c] != '\0') {
// Checking for Match
if ( STR[m] == PAT[i]) {
i++; m++;
flag=1;
if ( PAT[i] == '\0')
{
//copy replace string in ans string
for(k=0; REP[k] != '\0';k++,j++)
ans[j] = REP[k];
i=0;
c=m;
}
```

```
} }  
else //mismatch  
{  
ans[j] = STR[c];  
j++; c++;  
m = c; i=0;  
} }  
if(flag==0)  
{  
printf("Pattern doesn't found!!!");  
}  
else  
{  
ans[j] = '\0';  
printf("\nThe RESULTANT string is:%s\n",ans);  
}  
getch();  
}
```



**OUTPUT:**

```
Enter the MAIN string:
welcome

Enter a PATTERN string:
come

Enter a REPLACE string:
done

The RESULTANT string is:weldone
```

```
Enter the MAIN string:
studio

Enter a PATTERN string:
queen

Enter a REPLACE string:
name
Pattern doesn't found!!!
```

**Practical:3**

**AIM:** Develop a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

- a. Push an Element on to Stack
- b. Pop an Element from Stack
- c. Demonstrate how Stack can be used to check Palindrome
- d. Demonstrate Overflow and Underflow situations on Stack
- e. Display the status of Stack
- f. Exit

Support the program with appropriate functions for each of the above operations.

**PROGRAM CODE:**

```
#include<stdlib.h>

#include<stdio.h>

#include<string.h>

#define max_size 5

int stack[max_size],top=-1,flag=1;

int i,temp,item,rev[max_size],num[max_size];

void push();

void pop();

void display();

void pali();

void main()

{

int choice;

printf("\n\n-----STACK OPERATIONS-----");

printf("1.Push\n");

printf("2.Pop\n"); printf("3.Palindrome\n"); printf("4.Display\n"); printf("5.Exit\n");

printf(" ");

while(1)

{
```

```
printf("\nEnter your choice:\t");
scanf("%d",&choice);
switch(choice)
{
case 1: push();break;
case 2: pop();
if(flag)
printf("\nThe popped element: %d\t",item);
temp=top; break;
case 3: pali();
top=temp; break;
case 4: display(); break;
case 5: exit(0); break;
default: printf("\nInvalid choice:\n"); break;
} }
//return 0;
getch();
}
```

```
void push() //Inserting element into the stack
{
if(top==(max_size-1))
{
printf("\nStack Overflow:");
}
else
{
printf("Enter the element to be inserted:\t");
```

```
scanf("%d",&item);
top=top+1;
stack[top]=item;
}
temp=top;
}
void pop() //deleting an element from the stack
{
if(top== -1)
{
printf("Stack Underflow:");
flag=0;
}
else
{
item=stack[top];
top=top-1;
}
}

void pali()
{ i=0;
if(top== -1)
{
printf("Push some elements into the stack first\n");
}
else
{
```

```
while(top!=-1)
{
rev[top]=stack[top]; pop();
}
top=temp; for(i=0;i<=temp;i++)
{
if(stack[top--]==rev[i])
{
if(i==temp)
{
printf("Palindrome\n"); return;
}
}
}
printf("Not Palindrome\n");
}

void display()
{
int i; top=temp;
if(top== -1)
{
printf("\nStack is Empty:");
}
else
{
printf("\nThe stack elements are:\n" );
for(i=top;i>=0;i--)
```

```
{  
printf("%d\n",stack[i]);  
}  
}  
}
```

**OUTPUT:**

```
-----STACK OPERATIONS-----1.Push  
2.Pop  
3.Palindrome  
4.Display  
5.Exit  
  
Enter your choice:      1  
Enter the element to be inserted:      2  
  
Enter your choice:      2  
  
The popped element: 2  
Enter your choice:      4  
  
Stack is Empty:  
Enter your choice:      1  
Enter the element to be inserted:      2  
  
Enter your choice:      1  
Enter the element to be inserted:      3  
  
Enter your choice:      1  
Enter the element to be inserted:      4  
  
Enter your choice:      4  
  
The stack elements are:  
4  
3  
2  
  
Enter your choice:
```

```
Enter your choice:      1
Enter the element to be inserted:      5

Enter your choice:      1
Enter the element to be inserted:      6

Enter your choice:      1

Stack Overflow:
Enter your choice:      4

The stack elements are:
6
5
4
3
2

Enter your choice:      3
Not Palindrome

Enter your choice:      1

Stack Overflow:
Enter your choice:
```

**Practical:4**

**AIM:** Develop a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, \*, /, % (Remainder), ^ (Power) and alphanumeric operands.

**PROGRAM CODE:**

```
#define SIZE 50 /* Size of Stack */

#include <ctype.h>

#include <stdio.h>

char s[SIZE];

int top = -1; /* Global declarations */

push(char elem) /* Function for PUSH operation */
{
    s[++top] = elem;
}

char pop() /* Function for POP operation */
{
    return (s[top--]);
}

int pr(char elem) /* Function for precedence */
{
    switch (elem)
    {
        case '#': return 0;
        case '(': return 1;
        case '+':
        case '-': return 2;
        case '*': case '/':
        case '%': return 3;
```



```
case '^': return 4;
} }

void main() /* Main Program */
{
char infx[50], pofx[50], ch, elem;
int i = 0, k = 0;
printf("\n\nRead the Infix Expression ? ");
scanf("%s", infx);
push('#');
while ((ch = infx[i++]) != '\0')
{
if (ch == '(') push(ch);
else if (isalnum(ch))
pofx[k++] = ch;
else if (ch == ')')
{
while (s[top] != '(')
pofx[k++] = pop();
elem = pop(); /* Remove ( */
}
else /* Operator */
{
while (pr(s[top]) >= pr(ch))
pofx[k++] = pop();
push(ch);
} }
while (s[top] != '#') /* Pop from stack till empty */
pofx[k++] = pop();
```

```
pofx[k] = '\0'; /* Make pofx as valid string */  
printf("\n\nGiven Infix Expn: %s Postfix Expn: %s\n", infix, pofx);  
}
```

**OUTPUT:**

```
Read the Infix Expression ? (a+b)-(c^d)/e  
  
Given Infix Expn: (a+b)-(c^d)/e Postfix Expn: ab+cd^e/-
```

**Practical:5**

**AIM:** Develop a Program in C for the following Stack Applications

- a. Evaluation of Suffix expression with single digit operands and operators: +, -, \*, /, %, ^
- b. Solving Tower of Hanoi problem with n disks

**PROGRAM CODE:**

//Evaluation of Suffix expression

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

```
#include <conio.h>
```

```
#include <math.h>
```

```
#define MAX 20
```

```
struct stack
```

```
{
```

```
int top;
```

```
float str[MAX];
```

```
}s;//stack
```

```
char postfix[MAX];//postfix
```

```
void push(float);
```

```
float pop();
```

```
int isoperand(char);
```

```
float operate(float,float,char);
```

```
int main()
```

```
{
```

```
int i=0;
```

```
printf("Enter Expression:");
```

```
scanf("%s",postfix);
```

```
float ans,op1,op2;
```

```
while(postfix[i]!='\0')
```

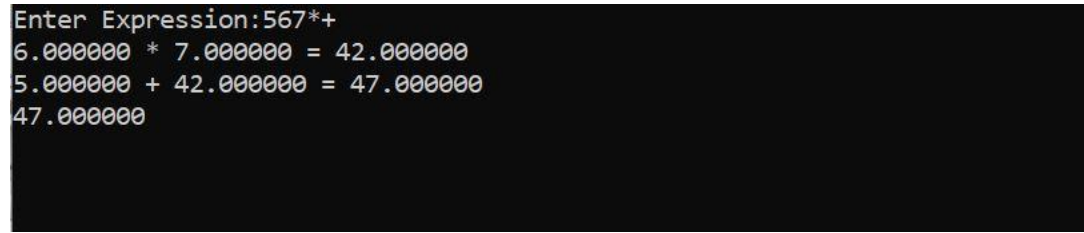
```
{
```

```
if(isoperand(postfix[i]))
push(postfix[i]-48);
else
{
op1=pop();
op2=pop();
ans=operate(op1,op2,postfix[i]);
push(ans);
printf("%f %c %f = %f\n",op2,postfix[i],op1,ans);
}
i++;
}
printf("%f",s.str[s.top]);
getch();
}

int isoperand(char x)
{
if(x>='0' && x<='9')
return 1;
else return 0;
}

void push(float x)
{
if(s.top==MAX-1)
printf("Stack is full\nStack overflow\n");
else
{
s.top++;
```

```
s.str[s.top]=x;
} }
float pop()
{
if(s.top==-1)
{
printf("Stack is empty\nSTACK UNDERFLOW\n");
getch();
}
else
{
s.top--;
return s.str[s.top+1];
} }
float operate(float op1,float op2,char a)
{
switch(a)
{
case '+': return op2+op1;
case '-': return op2-op1;
case '*': return op2*op1;
case '/': return op2/op1;
case '^': return pow(op2,op1);
} }
}
```

**OUTPUT:**

```
Enter Expression:567*+
6.000000 * 7.000000 = 42.000000
5.000000 + 42.000000 = 47.000000
47.000000
```

**PROGRAM CODE:**

```
//5B Solving Tower of Hanoi problem with n disks

#include <stdio.h>

#include <conio.h>

void tower(int n, int source, int temp,int destination)
{
    if(n == 0)
        return;
    tower(n-1, source, destination, temp);
    printf("\nMove disc %d from %c to %c", n, source, destination);
    tower(n-1, temp, source, destination);
}

int main()
{
    int n;
    printf("\nEnter the number of discs: \n");
    scanf("%d", &n);
    tower(n, 'A', 'B', 'C');
    printf("\n\nTotal Number of moves are: %d", (int)pow(2,n)-1);
    return 0;
}
```

**OUTPUT:**

```
Enter the number of discs:
2

Move disc 1 from A to B
Move disc 2 from A to C
Move disc 1 from B to C

Total Number of moves are: 3
```

```
Enter the number of discs:
4

Move disc 1 from A to B
Move disc 2 from A to C
Move disc 1 from B to C
Move disc 3 from A to B
Move disc 1 from C to A
Move disc 2 from C to B
Move disc 1 from A to B
Move disc 4 from A to C
Move disc 1 from B to C
Move disc 2 from B to A
Move disc 1 from C to A
Move disc 3 from B to C
Move disc 1 from A to B
Move disc 2 from A to C
Move disc 1 from B to C

Total Number of moves are: 15
```

**Practical:6**

**AIM:** Develop a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

- a. Insert an Element on to Circular QUEUE
- b. Delete an Element from Circular QUEUE
- c. Demonstrate Overflow and Underflow situations on Circular QUEUE
- d. Display the status of Circular QUEUE
- e. Exit

Support the program with appropriate functions for each of the above operations

**PROGRAM CODE:**

```
#include<stdio.h>

# define MAX 5

int cqueue_arr[MAX];

int front = -1;

int rear = -1;

/*Begin of insert*/

void insert(int item)

{

    if((front == 0 && rear == MAX-1) || (front == rear+1))

    {

        printf("Queue Overflow \n");

        return;

    }

    if (front == -1) /*If queue is empty */

    {

        front = 0;

        rear = 0;

    }

    else

    {
```



```
        if(rear == MAX-1)    /*rear is at last position of queue */
            rear = 0;
        else
            rear = rear+1;
    }
    cqueue_arr[rear] = item ;
}
/*End of insert*/
/*Begin of del*/
void del()
{
    if (front == -1)
    {
        printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",cqueue_arr[front]);
    if(front == rear) /* queue has only one element */
    {
        front = -1;
        rear=-1;
    }
    else
    {
        if(front == MAX-1)
            front = 0;
        else
            front = front+1;
    }
}
```

```
        }
    }
/*End of del() */
/*Begin of display*/
void display()
{
    int front_pos = front, rear_pos = rear;
    if(front == -1)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements :\n");
    if( front_pos <= rear_pos )
        while(front_pos <= rear_pos)
        {
            printf("%d ",cqueue_arr[front_pos]);
            front_pos++;
        }
    else
    {
        while(front_pos <= MAX-1)
        {
            printf("%d ",cqueue_arr[front_pos]);
            front_pos++;
        }
        front_pos = 0;
        while(front_pos <= rear_pos)
```

```
        {
            printf("%d ",cqueue_arr[front_pos]);
            front_pos++;
        }
    }
    printf("\n");
}
/*End of display*/
/*Begin of main*/
int main()
{
    int choice,item;
    do
    {
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display\n");
        printf("4.Quit\n");

        printf("Enter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1 :
                printf("Input the element for insertion in queue : ");
                scanf("%d", &item);
```

```
        insert(item);
        break;
    case 2 :
        del();
        break;
    case 3:
        display();
        break;
    case 4:
        break;
    default:
        printf("Wrong choice\n");
    }
}while(choice!=4);
return 0;
}
/*End of main*/
```

**OUTPUT:**

```
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 3
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 2
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 5
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 3
Queue elements :
3 2 5
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 2
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 6
1.Insert
2.Delete
3.Display
4.Quit
```

```
Enter your choice : 2
Element deleted from queue is : 6
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 2
Queue Underflow
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 2
Queue Underflow
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 2
Queue Underflow
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 2
Queue Underflow
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 2
Queue Underflow
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 3
Queue is empty
1.Insert
2.Delete
3.Display
```

**Practical:7**

**AIM:** Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Programme, Sem, PhNo*

- a. Create a SLL of N Students Data by using *front insertion*.
- b. Display the status of SLL and count the number of nodes in it
- c. Perform Insertion / Deletion at End of SLL
- d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
- e. Exit

**PROGRAM CODE:**

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int count=0;

struct stud

{

long long int ph;

int sem;

char name[15],usn[15],brnch[8];

struct stud *next;

}

*head=NULL,*tail=NULL,*temp=NULL,*temp1;

void create(long long int n,int s,char na[20],char u[15],char b[5])

{

if(head==NULL)

{

head=(struct stud*)malloc(1*sizeof(struct stud));

head->ph=n;

head->sem=s;

strcpy(head->name,na);
```

```
strcpy(head->usn,u);
strcpy(head->brnch,b);
head->next=NULL;
tail=head;
count++;
}
else
{
temp=(struct stud*)malloc(1*sizeof(struct stud));
temp->ph=n;
temp->sem=s;
strcpy(temp->name,na);
strcpy(temp->usn,u);
strcpy(temp->brnch,b);
temp->next=NULL;
tail->next=temp;
tail=temp;
count++;
} }
void display()
{
temp1=head;
if(temp1==NULL)
{
printf("\nlist is empty\n");
}
else
{
```



```
printf("student details are as follows:\n");
while(temp1!=NULL)
{
printf(" \n");
printf("NAME:%s\nUSN:%s\nBRANCH:%s\nSEM:%d\nPHONE NO.:%lld\n",temp1-
>name,temp1->usn,temp1->brnch,temp1->sem,temp1->ph);
printf(" \n");
temp1=temp1->next;
}
printf("no. of nodes=%d\n",count);
} }

void insert_head(long long int n,int s,char na[15],char u[15],char b[8])
{
temp=(struct stud*)malloc(1*sizeof(struct stud));
temp->ph=n;
temp->sem=s;
strcpy(temp->name,na);
strcpy(temp->usn,u);
strcpy(temp->brnch,b);
temp->next=head;
head=temp;
count++;
}

void insert_tail(long long int n,int s,char na[15],char u[15],char b[8])
{
temp=(struct stud*)malloc(1*sizeof(struct stud));
temp->ph=n;
temp->sem=s;
```

```
strcpy(temp->name,na);
strcpy(temp->usn,u);
strcpy(temp->brnch,b);
tail->next=temp;
temp->next=NULL;
tail=temp;
count++;
}

void delete_head()
{
temp1=head;
if(temp1==NULL)
{
printf("list is empty\n");
}
else
{
head=head->next;
printf("deleted node is:\n");
printf(" \n");
printf("NAME:%s\nUSN:%s\nBRANCH:%s\nSEM:%d\nPHONE NO.:%lld\n",temp1->name,
temp1->usn,temp1->brnch,temp1->sem,temp1->ph);
printf(" \n");
free(temp1);
count--;
} }

void delete_tail()
{
```

```
temp1=head;
if(temp1==NULL)
{
printf("list is empty\n");
}
while(temp1->next!=tail)
{
temp1=temp1->next;
}
printf("deleted node is:\n"); printf(" \n");
printf("NAME:%s\nUSN:%s\nBRANCH:%s\nSEM:%d\nPHONE NO.:%lld\n",tail->name,tail->usn,tail->brnch,tail->sem,tail->ph); printf(" \n");
free(tail);
tail=temp1;
tail->next=NULL;
count--;
}
void main()
{
int choice;
long long int ph; int sem;
char name[20],usn[15],brnch[5]; printf("-----MENU \n");
printf("1.create\n2.Insert from head\n3.Insert from tail\n4.Delete from head\n5.Delete from tail\n6.display\n7.exit\n");
printf(" \n");
while(1)
{
printf("enter your choice\n");
scanf("%d",&choice);
```

```
switch(choice)
{
case 1:
printf("enter the name usn branch sem phno. of the student respectively\n");
scanf("%s%s%s%sd%lld",name,usn,brnch,&sem,&ph);
create(ph,sem,name,usn,brnch);
break;
case 2:
printf("enter the name usn branch sem phno. of the student respectively\n");
scanf("%s%s%s%sd%lld",name,usn,brnch,&sem,&ph);
insert_head(ph,sem,name,usn,brnch);
break;
case 3:
printf("enter the name usn branch sem phno. of the student respectively\n");
scanf("%s%s%s%sd%lld",name,usn,brnch,&sem,&ph);
insert_tail(ph,sem,name,usn,brnch);
break;
case 4:
delete_head();
break;
case 5:
delete_tail();
break;
case 6:
display();
break;
case 7:
exit(0);
```

```
default:printf("invalid option\n");  
} } }
```

**OUTPUT:**

```
-----MENU  
1.create  
2.Insert from head  
3.Insert from tail  
4.Delete from head*.Delete from tail  
6.display  
7.exit  
  
enter your choice  
1  
enter the name usn branch sem phno. of the student respectively  
harsh 1 cse 3 7568797867  
enter your choice  
6  
student details are as follows:  
  
NAME:harsh  
USN:1  
BRANCH:cse  
SEM:3  
PHONE NO.:7568797867  
  
no. of nodes=1
```

```
enter your choice
6
student details are as follows:

NAME:anjali
USN:4
BRANCH:aiml
SEM:3
PHONE NO.:9934546757
```

```
NAME:harsh
USN:1
BRANCH:cse
SEM:3
PHONE NO.:7568797867
```

```
no. of nodes=2
enter your choice
4
deleted node is:
```

```
NAME:anjali
USN:4
BRANCH:aiml
SEM:3
PHONE NO.:9934546757
```

```
enter your choice
4
deleted node is:
```

```
NAME:harsh
USN:1
BRANCH:cse
SEM:3
PHONE NO.:7568797867
```

```
enter your choice
```

**Practical:8**

**AIM:** Develop a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: *SSN, Name, Dept, Designation, Sal, PhNo*

- a. Create a DLL of N Employees Data by using *end insertion*.
- b. Display the status of DLL and count the number of nodes in it
- c. Perform Insertion and Deletion at End of DLL
- d. Perform Insertion and Deletion at Front of DLL
- e. Demonstrate how this DLL can be used as Double Ended Queue.
- f. Exit

**PROGRAM CODE:**

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Enode
{
char ssn[15];
char name[20];
char dept[5];
char designation[10];
int salary;
long long int phno;
struct Enode *left;
struct Enode *right;
}*head=NULL;

struct Enode *tail,*temp1,*temp2;

void create(char [],char [],char [],char [],int ,long long int);
void ins_beg(char [],char [],char [],char [],int ,long long int);
void ins_end(char [],char [],char [],char [],int ,long long int);
void del_beg();
```

```
void del_end();
void display();
int count=0;
void main()
{
int choice;
char s[15],n[20],dpt[5],des[10];
int sal;
long long int p;
printf("1.Create\n2.Display\n3.Insert at beginning\n4.Insert at End\n5.Delete at
beginning\n6.Deleteat End\n7.Exit\n");
while(1)
{
printf("\nEnter your choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter the required data(Emp no,Name,Dept,Desig,sal,phone\n");
scanf("%s%s%s%s%d%lld",s,n,dpt,des,&sal,&p);
create(s,n,dpt,des,sal,p);
break;
case 2:
display();
break;
case 3:
printf("Enter the required data (Emp no,Name,Dept,Desig,sal,phone\n");
scanf("%s%s%s%s%s%d%lld",s,n,dpt,des,&sal,&p);
```



```
ins_beg(s,n,dpt,des,sal,p);
break;
case 4:
printf("Enter the required data(Emp no,Name,Dept,Desig,sal,phone\n");
scanf("%s%s%s%s%s%d%lld",s,n,dpt,des,&sal,&p);
ins_end(s,n,dpt,des,sal,p);
break;
case 5:
del_beg();
break;
case 6:
del_end();
break;
case 7:
exit(0);
} } }

void create(char s[15],char n[20],char dpt[5],char des[10],int sal,long long int p)
{
if(head==NULL)
{
head=(struct Enode *)malloc(1*sizeof(struct Enode));
strcpy(head->ssn,s);
strcpy(head->name,n);
strcpy(head->dept,dpt);
strcpy(head->designation,des);
head->salary=sal;
head->phno=p;
head->left=NULL;
```

```
head->right=NULL;
tail=head;
}
else
{
temp1=(struct Enode *)malloc(1*sizeof(struct Enode));
strcpy(temp1->ssn,s);
strcpy(temp1->name,n);
strcpy(temp1->dept,dpt);
strcpy(temp1->designation,des);
temp1->salary=sal;
temp1->phno=p;
tail->right=temp1;
temp1->right=NULL;
temp1->left=tail;
tail=temp1;
} }
void display()
{
temp1=head;
printf("Employee Details \n");
while(temp1!=NULL)
{
printf(" \n");
printf("%s\n%s\n%s\n%s\n%d\n%lld\n",temp1->ssn,temp1->name,temp1->dept,temp1->designation,temp1->salary,temp1->phno);
printf(" ");
temp1=temp1->right;
```

```
}  
}  
  
void ins_beg(char s[15],char n[20],char dpt[5],char des[10],int sal,long long int p)  
{  
    temp1=(struct Enode *)malloc(1*sizeof(struct Enode));  
    strcpy(temp1->:ssn,s);  
    strcpy(temp1->name,n);  
    strcpy(temp1->dept,dpt);  
    strcpy(temp1->designation,des);  
    temp1->salary=sal;  
    temp1->phno=p;  
    temp1->right=head;  
    head->left=temp1;  
    head=temp1;  
    temp1->left=NULL;  
}  
  
void ins_end(char s[15],char n[20],char dpt[5],char des[10],int sal,long long int p)  
{  
    temp1=(struct Enode *)malloc(1*sizeof(struct Enode));  
    strcpy(temp1->:ssn,s);  
    strcpy(temp1->name,n);  
    strcpy(temp1->dept,dpt);  
    strcpy(temp1->designation,des);  
    temp1->salary=sal;  
    temp1->phno=p;  
    tail->right=temp1;  
    temp1->left=tail;  
    temp1->right=NULL;
```

```
tail=temp1;
}
void del_beg()
{
temp1=head->right;
free(head);
head=temp1;
head->left=NULL;
}
void del_end()
{
temp1=tail->left;
free(tail);
tail=temp1;
tail->right=NULL;
}
```

**OUTPUT:**

```
1.Create
2.Display
3.Insert at beginning
4.Insert at End
5.Delete at beginning
6.Deleteat End
7.Exit

Enter your choice
1
Enter the required data(Emp no,Name,Dept,Desig,sal,phone
123
abc
cde
efg
12345
123456768

Enter your choice
2
Employee Details

123
abc
cde
efg
12345
123456768

Enter your choice
3
Enter the required data (Emp no,Name,Dept,Desig,sal,phone
234
xyz
one
two
2323
123123123
```

**Practical:9**

**AIM:** Develop a Program in C for the following operations on Singly Circular Linked List (SCLL)

with header nodes

a. Represent and Evaluate a Polynomial  $P(x,y,z) = 6x^2y^2z - 4yz^5 + 3x^3yz + 2xy^5z - 2xyz^3$

b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)

Support the program with appropriate functions for each of the above operations

**PROGRAM CODE:**

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

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

```
#include<math.h>
```

```
typedef struct poly_node
```

```
{
```

```
    float coef;
```

```
    int expx;
```

```
    int expy;
```

```
    int expz;
```

```
    struct poly_node *link;
```

```
} POLY;
```

```
POLY *getNode();
```

```
void read_poly(POLY *head, int n);
```

```
void print_poly(POLY *head);
```

```
POLY *add_poly(POLY *h1, POLY *h2);
```

```
int compare(POLY *temp1, POLY *temp2);
```

```
void attach(float cf, POLY *exptemp, POLY **tempres);
```

```
POLY* delete(POLY *head, POLY *temp);
```

```
void evaluate(POLY *head);
```

```
void main()
```

```
{
```

```
    int n1, n2;
```

```
    POLY *POLY1 = getNode();
```

```
    POLY *POLY2 = getNode();
```

```
    POLY *POLYSUM = getNode();
```

```
    POLY1->expx = -1;
```

```
    POLY1->link = POLY1;
```

```
    POLY2->link = POLY2;
```

```
    POLYSUM->link = POLYSUM;
```

```
    printf("\nEnter the number of terms for both polynomials\n");
```

```
    scanf("%d%d",&n1, &n2);
```

```
    printf("\nEnter 1st Polynomial\n");
```

```
    read_poly(POLY1, n1);
```

```
    printf("\n1st Polynomial is\n");
```

```
    print_poly(POLY1);
```

```
    printf("\nEnter 2nd Polynomial\n");
```

```
    read_poly(POLY2, n2);
```

```
    printf("\n2nd Polynomial is\n");
```

```
    print_poly(POLY2);
```

```
    POLYSUM = add_poly(POLY1, POLY2);
```

```
    printf("\nThe Resultant polynomial is\n");
```

```
    print_poly(POLYSUM);
```

```
    evaluate(POLYSUM);
```

```
}
```

```
POLY *getNode()
{
    POLY *temp = (POLY *) malloc(sizeof(POLY));
    if(temp == NULL)
    {
        printf("No Memory\n");
        exit(0);
    }
    return temp;
}
```

```
void read_poly(POLY *head, int n)
{
    int i;
    POLY *new = NULL;
    POLY *temp = head;
    for(i=0; i<n; i++)
    {
        new = getNode();
        printf("Enter Coef and Exps\n");
        scanf("%f%d%d", &(new->coef), &(new->expx), &(new->expy), &(new->expz));
        (temp->link) = new;
        temp = temp->link;
    }
    temp->link = head;
    return;
}
```



```
}
```

```
void print_poly(POLY *head)
```

```
{
```

```
    POLY *temp = head->link;
```

```
    while(temp != head)
```

```
    {
```

```
        printf("%f*X^%d*Y^%d*Z^%d\t", temp->coef, temp->expx, temp->expy, temp->expz);
```

```
        temp = temp->link;
```

```
    }
```

```
    printf("\n");
```

```
    return;
```

```
}
```

```
POLY *add_poly(POLY *h1, POLY *h2)
```

```
{
```

```
    float cf;
```

```
    POLY *temp1 = h1->link, *temp2 = NULL;
```

```
    POLY *result = getNode();
```

```
    POLY *tempres = result;
```

```
    while(temp1 != h1)
```

```
    {
```

```
        temp2 = h2->link;
```

```
        while(temp2 != h2)
```

```
        {
```

```
            switch(compare(temp1, temp2))
```

```
            {
```

case 1:

```
cf = temp1->coef + temp2->coef;
if(cf)
{
    attach(cf, temp1, &tempres);
}
temp1 = temp1->link;
h2 = delete(h2, temp2);
temp2 = h2->link;
break;
```

case 2:

```
temp2 = temp2->link;
break;
}
}
if(temp1 != h1)
{
    attach(temp1->coef, temp1, &tempres);
    temp1 = temp1->link;
}
}
temp2 = h2->link;
while(temp2 != h2)
{
    attach(temp2->coef, temp2, &tempres);
    temp2 = temp2->link;
}
```

```
    tempres->link = result;

    return result;

}

int compare(POLY *temp1, POLY *temp2)
{
    if((temp1->expx == temp2->expx) && (temp1->expy == temp2->expy) && (temp1->expz == temp2->expz))
    {
        return 1;
    }
    return 2;
}

void attach(float cf, POLY *exptemp, POLY **tempres)
{
    POLY *new = getNode();
    new->coef = cf;
    new->expx = exptemp->expx;
    new->expy = exptemp->expy;
    new->expz = exptemp->expz;
    (*tempres)->link = new;
    *tempres = new;
    return;
}

POLY* delete(POLY *head, POLY *temp)
{

```

```
POLY *previous = head, *present = head->link;
while(present != temp)
{
    previous = present;
    present = present->link;
}
previous->link = present->link;
free(present);
return head;
}

void evaluate(POLY *head)
{
    float result = 0.0;
    int x,y,z;
    POLY *temp = head->link;
    printf("\nEnter exponents\n");
    scanf("%d%d%d", &x, &y, &z);
    while(temp != head)
    {
        result += (temp->coef)*pow(x, temp->expx)*pow(y, temp->expy)*pow(z, temp-
>expz);
        temp = temp->link;
    }
    printf("\nResult after evaluation is %f\n", result);
    return;
}
```

**OUTPUT:**

```
Enter the number of terms for both polynomials
2
3

Enter 1st Polynomial
Enter Coef and Exps
2
3
5
6
Enter Coef and Exps
1
2
5
3

1st Polynomial is
2.000000*X^3*Y^5*Z^6    1.000000*X^2*Y^5*Z^3

Enter 2nd Polynomial
Enter Coef and Exps
1
2
3
4
Enter Coef and Exps
2
5
6
8
Enter Coef and Exps
2
1
5
7

2nd Polynomial is
1.000000*X^2*Y^3*Z^4    2.000000*X^5*Y^6*Z^8    2.000000*X^1*Y^5*Z^7

The Resultant polynomial is
2.000000*X^3*Y^5*Z^6    1.000000*X^2*Y^5*Z^3    1.000000*X^2*Y^3*Z^4    2.000000*X^5*Y^6*Z^8    2.000000*X^1*Y^5*Z^7

Enter exponents
2
2
1

Result after evaluation is 4896.000000
```

**Practical:10**

**AIM:** Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers .

- a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
- b. Traverse the BST in Inorder, Preorder and Post Order
- c. Search the BST for a given element (KEY) and report the appropriate message
- d. Exit

**PROGRAM CODE:**

```
#include<stdio.h>

#include<stdlib.h>

struct BST
{
    int data;
    struct BST *lchild;
    struct BST *rchild;
};

typedef struct BST * NODE;

NODE create()
{
    NODE temp;
    temp = (NODE) malloc(sizeof(struct BST));
    printf("\nEnter The value: ");
    scanf("%d", &temp->data);

    temp->lchild = NULL;
    temp->rchild = NULL;
    return temp;
}
```

```
}
```

```
void insert(NODE root, NODE newnode);
```

```
void inorder(NODE root);
```

```
void preorder(NODE root);
```

```
void postorder(NODE root);
```

```
void search(NODE root);
```

```
void insert(NODE root, NODE newnode)
```

```
{
```

```
    /*Note: if newnode->data == root->data it will be skipped. No duplicate nodes are allowed */
```

```
    if (newnode->data < root->data)
```

```
    {
```

```
        if (root->lchild == NULL)
```

```
            root->lchild = newnode;
```

```
        else
```

```
            insert(root->lchild, newnode);
```

```
    }
```

```
    if (newnode->data > root->data)
```

```
    {
```

```
        if (root->rchild == NULL)
```

```
            root->rchild = newnode;
```

```
        else
```

```
            insert(root->rchild, newnode);
```

```
    }
```

```
}
```

```
void search(NODE root)
{
    int key;
    NODE cur;
    if(root == NULL)
    {
        printf("\nBST is empty.");
        return;
    }
    printf("\nEnter Element to be searched: ");
    scanf("%d", &key);
    cur = root;
    while (cur != NULL)
    {
        if (cur->data == key)
        {
            printf("\nKey element is present in BST");
            return;
        }
        if (key < cur->data)
            cur = cur->lchild;
        else
            cur = cur->rchild;
    }
    printf("\nKey element is not found in the BST");
}
```



```
void inorder(NODE root)
```

```
{  
    if(root != NULL)  
    {  
        inorder(root->lchild);  
        printf("%d ", root->data);  
        inorder(root->rchild);  
    }  
}
```

```
void preorder(NODE root)
```

```
{  
    if (root != NULL)  
    {  
        printf("%d ", root->data);  
        preorder(root->lchild);  
        preorder(root->rchild);  
    }  
}
```

```
void postorder(NODE root)
```

```
{  
    if (root != NULL)  
    {  
        postorder(root->lchild);  
        postorder(root->rchild);  
        printf("%d ", root->data);  
    }  
}
```

```
}
```

```
void main()
```

```
{
```

```
    int ch, key, val, i, n;
```

```
    NODE root = NULL, newnode;
```

```
    while(1)
```

```
    {
```

```
        printf("\n~~~~~BST MENU~~~~~");
```

```
        printf("\n1.Create a BST");
```

```
        printf("\n2.BST Traversals:");
```

```
        printf("\n3.Search ");
```

```
        printf("\n4.Exit");
```

```
        printf("\nEnter your choice: ");
```

```
        scanf("%d", &ch);
```

```
        switch(ch)
```

```
        {
```

```
            case 1:    printf("\nEnter the number of elements: ");
```

```
                      scanf("%d", &n);
```

```
                      for(i=1;i<=n;i++)
```

```
                      {
```

```
                          newnode = create();
```

```
                          if (root == NULL)
```

```
                              root = newnode;
```

```
                          else
```

```
                              insert(root, newnode);
```

```
                      }
```

```
        break;
    case 2:    if (root == NULL)
                printf("\nTree Is Not Created");
            else
            {
                printf("\nThe Preorder display : ");
                preorder(root);
                printf("\nThe Inorder display : ");
                inorder(root);
                printf("\nThe Postorder display : ");
                postorder(root);
            }

        break;
    case 3:    search(root);
                break;

    case 4:    exit(0);
            }
        }
    }
```

**OUTPUT:**

```
~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
4.Exit
Enter your choice: 1

Enter the number of elements: 12

Enter The value: 6
Enter The value: 9
Enter The value: 5
Enter The value: 2
Enter The value: 8
Enter The value: 15
Enter The value: 24
Enter The value: 14
Enter The value: 7
Enter The value: 8
Enter The value: 5
Enter The value: 2

~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
4.Exit
Enter your choice: 2
```

```
Enter your choice: 2

The Preorder display : 12 3 2 5 6 9 8 7 46 34 21 15 14 24 78
The Inorder display : 2 3 5 6 7 8 9 12 14 15 21 24 34 46 78
The Postorder display : 2 7 8 9 6 5 3 14 15 24 21 34 78 46 12
~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
4.Exit
Enter your choice: 3

Enter Element to be searched: 12

Key element is present in BST
~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
4.Exit
Enter your choice: 3

Enter Element to be searched: 1

Key element is not found in the BST
~~~~BST MENU~~~~
1.Create a BST
2.BST Traversals:
3.Search
4.Exit
Enter your choice:
```

**Practical:11**

**AIM:** Develop a Program in C for the following operations on Graph(G) of Cities

- a. Create a Graph of N cities using Adjacency Matrix.
- b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

**PROGRAM CODE:**

```
#include<stdio.h>

#include<stdlib.h>

int a[50][50], n, visited[50];

int q[20], front = -1, rear = -1;

int s[20], top = -1, count=0;

void bfs(int v)
{
    int i, cur;
    visited[v] = 1;
    q[++rear] = v;
    while(front!=rear)
    {
        cur = q[++front];
        for(i=1;i<=n;i++)
        {
            if((a[cur][i]==1)&&(visited[i]==0))
            {
                q[++rear] = i;
                visited[i] = 1;
                printf("%d ", i);
            }
        }
    }
}
```

```
        }
    }
}

void dfs(int v)
{
    int i;
    visited[v]=1;
    s[++top] = v;
    for(i=1;i<=n;i++)
    {
        if(a[v][i] == 1&& visited[i] == 0 )
        {
            printf("%d ", i);
            dfs(i);
        }
    }
}
```

```
int main()
{

    int ch, start, i,j;
    printf("\nEnter the number of vertices in graph: ");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=1; i<=n; i++)
```

```
{
    for(j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
}

for(i=1;i<=n;i++)
    visited[i]=0;
printf("\nEnter the starting vertex: ");
scanf("%d",&start);

printf("\n==>1. BFS: Print all nodes reachable from a given starting node");
printf("\n==>2. DFS: Print all nodes reachable from a given starting node");
printf("\n==>3.Exit");
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch)
{
    case 1: printf("\nNodes reachable from starting vertex %d are: ", start);
            bfs(start);
            for(i=1;i<=n;i++)
            {
                if(visited[i]==0)
                    printf("\nThe vertex that is not reachable is %d" ,i);
            }
            break;

    case 2: printf("\nNodes reachable from starting vertex %d are:\n",start);
```



```
        dfs(start);  
        break;  
    case 3: exit(0);  
    default: printf("\nPlease enter valid choice:");  
}  
}
```

**OUTPUT:**

```
Enter the number of vertices in graph: 4  
  
Enter the adjacency matrix:  
0 1 0 1  
0 0 1 0  
0 0 0 1  
0 0 0 0  
  
Enter the starting vertex: 1  
  
==>1. BFS: Print all nodes reachable from a given starting node  
==>2. DFS: Print all nodes reachable from a given starting node  
==>3:Exit  
Enter your choice: 1  
  
Nodes reachable from starting vertex 1 are: 2 4 3
```

```
Enter the number of vertices in graph: 4  
  
Enter the adjacency matrix:  
0 1 0 1  
0 0 1 0  
0 0 0 1  
0 0 0 0  
  
Enter the starting vertex: 2  
  
==>1. BFS: Print all nodes reachable from a given starting node  
==>2. DFS: Print all nodes reachable from a given starting node  
==>3:Exit  
Enter your choice: 2  
  
Nodes reachable from starting vertex 2 are:  
3 4
```

**Practical:12**

**AIM:** Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Develop a Program in C that uses Hash function  $H: K \rightarrow L$  as  $H(K)=K \bmod m$  (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

**PROGRAM CODE:**

```
#include<stdio.h>

#include<stdlib.h>

int key[20],n,m;
int *ht,index;
int count = 0;

void insert(int key)
{
    index = key % m;
    while(ht[index] != -1)
    {
        index = (index+1)%m;
    }
    ht[index] = key;
    count++;
}

void display()
{
```

```
    int i;
    if(count == 0)
    {
        printf("\nHash Table is empty");
        return;
    }

    printf("\nHash Table contents are:\n ");
    for(i=0; i<m; i++)
        printf("\n T[%d] --> %d ", i, ht[i]);
}

void main()
{
    int i;
    printf("\nEnter the number of employee records (N) : ");
    scanf("%d", &n);

    printf("\nEnter the two digit memory locations (m) for hash table: ");
    scanf("%d", &m);

    ht = (int *)malloc(m*sizeof(int));
    for(i=0; i<m; i++)
        ht[i] = -1;

    printf("\nEnter the four digit key values (K) for N Employee Records:\n ");
    for(i=0; i<n; i++)
```

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

for(i=0;i<n;i++)
{
    if(count == m)
    {
        printf("\n~~~Hash table is full. Cannot insert the record %d key~~~",i+1);
        break;
    }
    insert(key[i]);
}

//Displaying Keys inserted into hash table
display();
}
```

**OUTPUT:**

```
Enter the number of employee records (N) : 12

Enter the two digit memory locations (m) for hash table: 15

Enter the four digit key values (K) for N Employee Records:
1234
5678
3456
2345
6799
1245
6786
2345
1234
5689
3421
6785

Hash Table contents are:

T[0] --> 1245
T[1] --> 3421
T[2] --> -1
T[3] --> -1
T[4] --> 1234
T[5] --> 2345
T[6] --> 3456
T[7] --> 6799
T[8] --> 5678
T[9] --> 6786
T[10] --> 2345
T[11] --> 1234
T[12] --> 5689
T[13] --> 6785
T[14] --> -1
```

**MODEL VIVA QUESTIONS & ANSWERS****Title of the Practical: Program to search an element of array using linear search**

Q1 Define searching process?

A1 searching is the process of finding an element within the list of elements stored in any order or randomly.

Q2 How many types of searching are there?

A2 There is basically two types of searching:-linear search and Binary search.

Q3 Define: linear search?

A3 In linear search, we access each elements of an array one by one sequentially and see weather it is desired element or not.

Q4 Why binary search method is more efficient then liner search?

A4 It is because less time is taken by linear search to search an element from the sorted list of elements.

Q5 Efficiency of linear search ?

A5 The time taken or the number of comparisons made in searching a record in a search table determines the efficiency of the technique.

Q6 What do you understand by the term “linear search is unsuccessful”?

A6 search will be unsuccessful if all the elements are accessed and the desired elements are not found.

Q7 What is worse case?

A7 In the worse case, the no. of average case we may have to scan half of the size of the array ( $n/2$ ).

Q8 What is the drawback of linear search?

A8 There is no requisite for the linear Search.

Q9 During linear search, when the record is present in first position then how many comparisons are made ?

A9 Only one comparison.

Q10 During linear search, when the record is present in last position then how many comparisons are made?

A10  $n$  comparisons have to be made.

**Title of the Practical: Program to reverse the element of array. Insertion and deletion on array at specified position.**

Q1 What is an array & how many types of arrays represented in memory?

A1 An array is a structured datatype made up of a finite, fixed size, collection of homogeneous ordered elements. Types of array: One-Dimensional array, Two-Dimensional array, Multi-Dimensional array.

Q2 How can be declared an array?

A2 `data_type var_name[ Expression];`

Q3 How can be insert an element in an array?

A3 Insertion of a new element in an array can be done in two ways:

\* Insertion at the end of array. \* Insertion at required position.

Q4 How can be delete an element in an array?

A4- Deleting an element at the end of an array presents no difficulties, but deleting element somewhere in the middle of the array.

Q5 How many types of implementation of a two-dimensional array?

A5 \* Row-major implementation \* Column-major implementation

Q6 What is array of pointers?

A6 Array of pointer refers to homogeneous collection of pointer that is collection of pointer of same datatype.

Q7 What are limitations of array?

A7 \* These are static structures. \* It is very time consuming.

Q8 Where the elements of the array are stored respectively in successive?

A8 Memory locations.

Q9 How can merge two arrays?

A9 Simplest way of merging two arrays is that first copy all elements of one array into a third empty array and then copy all the elements of other array into third array.

Q10 What is the operations of array?

A10 Insertion, Deletion, Traversing, Merging.

**Title of the Practical: Program based on structure union**

Q1 What are structures?

A1 Structures are used to store different types.

Q2 What are arrays?

A2 Arrays are used to store similar data types.

Q3 Is array of structures possible?

A3 Is array of structures are possible.

Q4 How structures are declared?

A4 Declaration: struct book { char name; Float price; Int pages; };

Q5 Why we use functions?

A5 Writing functions avoids rewriting the same code over and over.

Q6 Name some library functions?

A6 printf(), scanf() are examples of library function.

Q7 What are user defined functions?

A7 Functions declared by the user are called user defined functions.

Q8 Explain call by value?

A8 When the value is passed in the function is called call by value.

Q9 Explain call by reference?

A9 When the address of the value is passed is called call by reference.

Q10 Why we used keyword „break“?

A10 When break is encountered inside any loop, control automatically passes to the first statement after the loop.

**Title of the Practical: Program to implement PUSH and POP operation on stack.**

Q1 What is stack?

A1 Stack is an ordered collection of elements like array.

Q2 What are the operations performed on stack?

A2 Push for insertion and pop for deletion.

Q3 What is push operation?

A3 When an element is inserted into the stack is called push operation

Q4 What is pop operation.

A4 When an element is deleted from the stack is called pop operation.

Q5 How stacks are implemented?

A5 Stacks are implemented in two ways: static implementation –array Dynamic implementation –pointers.



Q6 What are the applications of stack?

A6 infix , post fix and prefix notations are the applications of stack.

Q7 What is recursion.

A7 Recursion is defined as function calling itself.

Q8 What are the different types of stack

A8 Direct: a system calls itself from within itself. Indirect: two functions mutually calls one another

Q9 Define “Top of stack”

A9 Pointer indicating the top element of the stack

Q10 Is stack is primitive or non primitive data structure ?

A10 Non primitive data structure.

**Title of the Practical: Program based on infix to prefix and post fix notation.**

Q1 What is Stack ?

A1 Stack is ordered collection of element like arrays out it has a special feature that deletion and insertion of element can be done only from one end called top of stack .It is also called LIFO(last in first out).

Q2 Application of Stack ?

A2 Infix Prefix Postfix

Q3 Terms used in Stack ?

A3 > Context > Stack frames > Maxsize

Q4 Explain infix in Stack ?

A4 The operator is written in between the operands. Ex:- A+B

Q5 Explain Postfix in Stack ?

A5 The operator is written after the operands.It is also called suffix notation. Ex:-  
AB+

Q6 Define operations on Stack ?

A6 The basic operation that can be performed on Stack are as follows: >PUSH >POP

Q7 Give postfix form for  $(A+B)*C/D$

A7  $AB+C*D/$

Q8 Give postfix form for  $A+B/C-D$

A8  $ABC/+D$

Q9 Give prefix form for  $A/B^C+D$   
A9  $+/A^BCD$

Q10 Give prefix form for  $A*B+C$   
A10  $+*ABC$

**Title of the Practical: Program based on queue & their operations for an application**

Q1 Define queue.

A1 Queue is an homogeneous collection of elements. It is logically a first in first out (FIFO) type of list. Queue means a line.

Q2 In how many ways queue is implemented?

A2 Queues can be implemented in two ways: 1. static implementation (using array)  
2. dynamic implementation (using pointers)

Q3 What is the rear end in a queue?

A3 In a queue new elements are added at one end called the rear end.

Q4 What is a front end?

A4 The existing elements in a queue are deleted from an end called the front end.

Q5 What is the value of front and rear end in an empty queue?

A5 Front = -1 and rear = -1.

Q6 Write an algorithm for deleting a node from a queue.

A6 1. If (front == null) Write queue is empty and exit Else Temp = start Value = temp  
->no Start = start -> next Free (temp) Return(value) 2. exit.

Q7 What are the different variations in a queue?

A7 Major variations are: 1. circular queue 2. double ended queue 3. priority queue.

Q8 What is the full form of dequeue?

A8 DEQUEUE : Double ended queue. It is another form of queue in which both insertion and deletion are performed at the either end.

Q9 When an element is added to the dequeue with n memory cells, what happens to LEFT or RIGHT.

A9 If the element is added on the left, then LEFT is decreased by 1 (mod n). If the element is added on the right, then RIGHT is increased by 1 (mod n)

Q10 What are the applications of queue.

A10 Applications are: 1. round robin technique for processor scheduling 2. All types of customer services(eg. RTC ) 3. Printer server routines.

**Title of the Practical: Program based on the implementation of circular queue.**

Q1 What is Circular Queue?

A1 A Circular Queue is one in which the insertion of a new element is done at the very first variation of the Queue if the last location of the Queue is full.

Q2 Why use of Circular Queue?

A2 A Circular Queue over comes the problem of un utilized space in linear Queue implemented as Array.

Q3 Explain Dequeue ?

A3 It is also a homogenous list of element in which insertion deletion of element are perform both the ends we insert element from the rear or from the front end.

Q4 What is Queue ?

A4 It is a non primitive linear data structure. In a Queue new element are added at the one end called rear end and the element are removed from another end called front end.

Q5 Variation in a Queue ?

A5 Circular Queue -> Dequeue ->Priority Queue

Q6 What is Priority Queue ?

A6 Priority Queue determines the order in which the exist in the Queue the highest Priority items are removed first.

Q7 Application of Queue ?

A7 1> Customer service center 2> Ticket counter 3> Queue is used for determine space complexity & time complexity.

Q8 What is Queue implementation?

A8 >> Static implementation(Array) >>Dynamic implementation(Pointer)

Q9 Define operations on queue?

A9 The basic operation that can be performed on queue are – 1>To insert an element in a Queue 2>To delete an element from the Queue

Q10 What is Stack ?

A10 Stack is ordered collection of element like arrays out it has a special feature that deletion and insertion of element can be done only from one end called top of stack .It is also called LIFO(last in first out).

**Title of the Practical: Program based on list operations and its applications.**

Q1 Define linked list.

A1 Linked list are special list of some data elements linked to one another .The logical ordering is represented by having each element pointing to the next element. Each element is called a node.

Q2 What does a node define?

A2 Node has two parts: INFO – it stores the information and POINTER – which points to the next element.

Q3 What are the different types of linked list?

A3 Linked list are of four types: 1. singly linked list 2. doubly linked list 3. circular linked list 4. circular doubly linked list.

Q4 What are the different operations performed on a linked list?

A4 Basic operations are: creation, insertion, deletion , traversing, searching , concatenation and display.

Q5 What are the advantages of linked list?

A5 Advantages are: 1. linked lists are dynamic data structures 2. Efficient memory utilizations 3. Insertion and deletions are easier and efficient

Q6 What is null pointer?

A6 The link field of the last node contains NULL rather than a valid address. It is a null pointer and indicates the end of the list.

Q7 What is external pointer?

A7 It is a pointer to the very first node in the linked list, it enables us to access the entire linked list.

Q8 What are the different notations used in a linked list.

A8 Node(p) : a node pointed to by the pointer p Data (p) : data of the node pointed by p Link (p) : address of the next node that follows the node pointed to by the pointer p.

Q9 What are advantages of circular linked list.

A9 1.Nodes can be accessed easily. 2. deletion of node is easier 3. concatenation and splitting of circular list is more efficient.

Q10 A doubly linked list contains how many fields?

A10 Doubly linked list consists of three fields: Data : contains the information Next: contains the address of the next node Prev: contains the address of the previous node.

### **Title of the Practical: Program based on pointers in C.**

Q1 Which of the following abstract data types are NOT used by Integer Abstract Data type group?

A1. A)Short B)Int C)float D)long

Q2 What pointer type is used to implement the heterogeneous linked list in C?

A2 The answer is the void pointer. The heterogeneous linked list contains different data types in its nodes and we need a link, pointer, to connect them. Since we can't use ordinary pointers for this, we use the void pointer. Void pointer is a generic pointer type, and capable of storing pointer to any type.

Q3: What issue do auto\_ptr objects address?

A3: If you use auto\_ptr objects you would not have to be concerned with heap objects not being deleted even if the exception is thrown.

Q4.: What is a dangling pointer?

A4: A dangling pointer arises when you use the address of an object after its lifetime is over. This may occur in situations like returning addresses of the automatic variables from a function or using the address of the memory block after it is freed.

Q5: What is the difference between a pointer and a reference?

A5: A reference must always refer to some object and, therefore, must always be initialized; pointers do not have such restrictions. A pointer can be reassigned to point to different objects while a reference always refers to an object with which it was initialized.

Q6: What is the difference between `const char *myPointer` and `char *const myPointer`?

A6: `Const char *myPointer` is a non constant pointer to constant data; while `char *const myPointer` is a constant pointer to non constant data.

Q7: From which is the pointer to object of a base class type compatible ?

A7: Pointers to object of a base class type is compatible with pointer to object of a derived class . Therefore , we can use single pointer variable to point to objects of base class as well as derived class.

Q8: What is a this pointer?

A8: A special pointer known as this pointer stores the address of the object that is currently invoking a member function.

Q9: How are objects passed to functions?

A9: Objects can be passed to functions through call-by-value as well as call-by-reference mechanism.

Q10: Why is Arrow operator (“->”) used?

A10: The arrow operator is used to access the public members of the class with a pointer to an object.

**Title of the Practical: Implementation of tree using linked list.**

Q1 What is a tree?

A1 A tree is a non linear data structure in which items are arranged in a sorted sequence. It is a finite set of one or more data items.

Q2 What is a root in a tree?

A2 A root is the first node in the hierarchical arrangement of data items.

Q3 What do you mean by degree of a tree?

A3 It is a maximum degree of nodes in a given tree .

Q4 What are non terminal nodes?

A4 Any node (Except the root node) is not zero is called non terminal node. Non terminal nodes are the intermediate nodes in traversing the given tree.

Q5 Who are called siblings in a tree?

A5 The children nodes of a given parent node are called siblings. They are also called brothers.

Q6 . During linear search, when the record is present somewhere in search table, then how many comparisons are made?

Ans-  $(n+1)/2$ .

Q7 How can access one-dimensional array elements?

A7 `array_name[ index or subscript ]`;

Q8 What is the traversing of an array?

A8 Traversing means to access all the elements of the array, starting from first element upto the last element in the array one-by-one.

Q9 Explain the types of searching?

A9 There are two types of searching: - 1> linear searching, 2> binary searching.

Q10 What is the linear search?

A10 We access each element of and see whether. It is desire element or not a search will be

unsuccessful. If the all elements are access and the desired element is not found.