```c
/*
Program 1 :
Design, Develop and Implement a menu driven Program in C for the following
array operations:
a. Creating an array of N Integer Elements
b. Display of array Elements with Suitable Headings
c. Inserting an Element (ELEM) at a given valid Position (POS)
d. Deleting an Element at a given valid Position (POS)
e. Exit.
*/

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

/*
Maximum size of the array is defined as 10.
It Can be changed according to the needs.
*/
#define MAX 10

/*
a[MAX] represents Maximum array size;
pos:position, x:element to be inserted/deleted
*/

int a[MAX],pos,x;

int n = 0;

//defining all the 4 operations to be done as functions.
void create();
void display();
void insert();
void delete();


void main()
{
    int choice;
    while(1)
        {
            printf("\n\n ****MENU****  ");

            printf("\n 1. Create an array of N integers");
            printf("\n 2. Display of array elements");
            printf("\n 3. Insert an element at a given Position");
            printf("\n 4. Delete an element at a given Position");
            printf("\n 5. Exit");

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

            switch(choice)
            {
                case 1: create();
                        break;
                case 2: display();
                        break;
                case 3: insert();
                        break;
                case 4: delete();
                        break;
                case 5: exit(1);
                        break;

                default: printf("\n Please enter a valid choice:");
```

```
            }

        }//end of while

}//end of main function

/*
create function -
Takes in number of elements and elements to be inserted as input
*/

void create()
{
    int i;
    printf("\n Enter the number of elements: ");
    scanf("%d",&n);

    printf("\n Enter the elements:");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
}

/*
Display function -
First checks if the array is empty.
If not, displays all the elements in the array.
*/

void display()
{
    int i;
    if(n==0)                    //if n is zero, then array is empty.
        {
            printf("\n Array is empty");
            return;
        }

    printf("\n Array elements are: ");
    for(i=0;i<n;i++)
    printf("%d\t",a[i]);    // prints all the elements in the array.
}

/*
insert function-
First checks if the array is full. If yes, array full message is displayed.
If not, it takes the element to be inserted at a prescribed position until
the position is less than the size of the array.
This is done using do-while loop.
*/

void insert()
{
    int i;
    if(n==MAX)
        {
            printf("\n Array is full.Insertion not possible");
            return;
        }


    do
        {
            printf("\n Enter valid position where element to be inserted:");
            scanf("%d",&pos);
```

```
        }
    while(pos>n);

    printf("\n Enter the value to be inserted:");
    scanf("%d",&x);

    for(i=n-1;i>=pos;i--)
    {
        a[i+1] = a[i];
    }
        a[pos] = x;
        n = n+1;
        display();
}

/*
delete function-
It first checks if the array is empty. If yes, array empty message is displayed.
If not, it keeps on taking the element to be deleted at a prescribed position
until the position is less than or equal to the size of the array.
This is done using do-while loop.
*/

void delete()
{
    int i;

    if(n==0)
        {
            printf("\n Array is empty");
            return;
        }

    do
        {
            printf("\n Enter valid position from where element to be deleted:");
            scanf("%d",&pos);
        }
    while(pos>=n);

    x=a[pos];
    printf("\n Deleted element is %d\n",x);

    for(i=pos;i<n-1;i++)
        {
            a[i]=a[i+1];
        }

    n=n-1;
    display();
}
```

```
****MENU****
 1. Create an array of N integers
 2. Display of array elements
 3. Insert an element at a given Position
 4. Delete an element at a given Position
 5. Exit

 Enter your choice: 1

 Enter the number of elements: 5

 Enter the elements:10 20 30 40 50



****MENU****
 1. Create an array of N integers
 2. Display of array elements
 3. Insert an element at a given Position
 4. Delete an element at a given Position
 5. Exit

 Enter your choice: 3

 Enter valid position where element to be inserted:2

 Enter the value to be inserted:456

 Array elements are: 10 20      456     30      40      50



****MENU****
 1. Create an array of N integers
 2. Display of array elements
 3. Insert an element at a given Position
 4. Delete an element at a given Position
 5. Exit

 Enter your choice: 3

 Enter valid position where element to be inserted:5

 Enter the value to be inserted:121

 Array elements are: 10 20      456     30      40      121     50



 ****MENU****
```

```
1. Create an array of N integers
2. Display of array elements
3. Insert an element at a given Position
4. Delete an element at a given Position
5. Exit

Enter your choice: 4

Enter valid position from where element to be deleted:1

Deleted element is 20

Array elements are: 10 456     30      40      121     50



****MENU****
1. Create an array of N integers
2. Display of array elements
3. Insert an element at a given Position
4. Delete an element at a given Position
5. Exit

Enter your choice: 4

Enter valid position from where element to be deleted:4

Deleted element is 121

Array elements are: 10 456     30      40      50



****MENU****
1. Create an array of N integers
2. Display of array elements
3. Insert an element at a given Position
4. Delete an element at a given Position
5. Exit

Enter your choice: 4

Enter valid position from where element to be deleted:1

Deleted element is 456

Array elements are: 10 30      40      50
```

```
****MENU****
1. Create an array of N integers
2. Display of array elements
3. Insert an element at a given Position
4. Delete an element at a given Position
5. Exit

Enter your choice: 5
```

```c
/*
2. Design, Develop and Implement a Program in C for 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.
*/

/*
Input:
'Main string' represented by STR
'Pattern String' represented by PAT
'Replace String' represented by REP

Output:
Printing the main string with Replaced pattern. (or)
Pattern not found.

Main, Pattern and Replace String will get added by '\0' at the end of string.

we use gets() to make sure we get entire string as input.
ans[] will be storing the main string with replaced pattern.

int flag=0; means by default flag is set to zero indicating pattern not found.
if the pattern is found, flag is made 1.
*/


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

void main()
{
    char STR[100],PAT[100],REP[100],ans[100];
    int i,j,c,m,k;
    int flag=0; //by default flag is set to zero indicating pattern not found.

    printf("Enter the MAIN string:\n");
    gets(STR);

    printf("Enter a PATTERN string:\n");
    gets(PAT);

    printf("Enter a REPLACE string:\n");
    gets(REP);

    i = j= m = c = 0;

    while(STR[c]!='\0')             //Till we reach end of main string
        {
            if (STR[m]==PAT[i])   // Checking for Matching main with pattern
                {
                    i++;
                    m++;

                    if(PAT[i]=='\0')    //if we reach end of pattern string
                        {
                            //copy replace string in the ans string
                            for(k=0; REP[k]!='\0';k++,j++)
                            {
                                ans[j] = REP[k];
                                flag=1;         //indicates pattern found.
                            }
```

```
67
68                              i=0;
69                              c=m;
70                          }
71                  }
72                  else                                    //if there is mismatch
73                      {
74                          ans[j] = STR[c];
75                          j++;
76                          c++;
77                          m = c;
78                          i=0;
79                      }
80          }
81
82          if(flag==0)
83              {
84                  printf("Pattern doesn't found!!!");
85              }
86          else
87              {
88                  ans[j] = '\0';  //Attach '\0' to answer specifying end of string.
89                  printf("The resultant string is\n:%s" ,ans);
90              }
91  }
```

OUTPUT:

```
Enter the MAIN string:
I HATE DSA
Enter a PATTERN string:
HATE
Enter a REPLACE string:
LOVE
The resultant string is:I LOVE DSA


Enter the MAIN string:
Good Morning
Enter a PATTERN string:
Morning
Enter a REPLACE string:
Evening
The resultant string is:Good Evening


Enter the MAIN string:
How u doing
Enter a PATTERN string:
making
Enter a REPLACE string:
going
Pattern not found!


Enter the MAIN string:
Sheldon Cooper
Enter a PATTERN string:
Leonard
Enter a REPLACE string:
Penny
Pattern not found!
```

```c
/*
3. Design, Develop and Implement 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
*/

/*
The program has to do 4 functions. So we are going to write 4 functions.
*/


#define MAX 5
int stack[5];
int top=-1;                          //Indicates initially stack is empty

void main()
{
    int ch;
    while(1)
        {
            printf("\n STACK OPERATIONS \n");
            printf("\n 1.Push\n 2.Pop\n 3.Display\n 4.Palindrome\n 5.Exit\n");

            printf("Enter your choice\n");
            scanf("%d",&ch);

            switch(ch)
            {
                case 1:push();
                       break;

                case 2:pop();
                       break;

                case 3:display();
                       break;

                case 4:palindrome();
                       break;

                case 5:return;
                default: printf("Invalid choice\n");
            }
        }
} //end of main function


//Push an Element on to Stack
void push()
{
    int item;

    if(top==(MAX-1))                          // Stack Overflow situations
        printf("Stack Overflow\n");
    else
        {
            printf("Enter the element to be pushed :");
            scanf("%d",&item);

```

```
            stack[++top]=item;          // pushing element to the top of stack
        }
}

/*
Pop an Element from Stack. Element is poped from the top of stack.
The last element entered is popped out. (LIFO)
*/

void pop()
{
    if(top==-1)
        printf("Stack Underflow\n");
    else
        printf("The poped element is %d\n",stack[top--]);
}

/*
Display the status of Stack.
When displaying, we start from the last element and keep decrementing till Zero.
*/
void display()
{
    int i;

    if(top==-1)
        printf("Stack Empty\n");
        else
            {
                printf("The elements of the stack are:\n");
                for(i=top;i>=0;i--)
                    printf("%d\n",stack[i]);
            }
}

//To show how Stack can be used to check Palindrome.

void palindrome()
{
    int i;
    int count=0;

    for(i=0; i<=(top/2); i++)
        {
            if(stack[i] == stack[top-i])
            count++;
        }

        if((top/2 +1) == count)
            printf("Stack contents are Palindrome\n");
        else
            printf("Stack contents are not palindrome\n");
}

```

OUTPUT:

```
 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
2
Stack Underflow

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
3
Stack Empty

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :10

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :20

 STACK OPERATIONS

 1.Push
 2.Pop
```

```
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :30

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :40

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :50

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Stack Overflow

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
3
The elements of the stack are:
```

```
50
40
30
20
10

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
2
The poped element is 50

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
2
The poped element is 40

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
2
The poped element is 30

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
2
The poped element is 20
```

```
 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
2
The poped element is 10

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
2
Stack Underflow

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
3
Stack Empty

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :1

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
```

```
 5.Exit
Enter your choice
1
Enter the element to be pushed :2

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :3

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :2

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
1
Enter the element to be pushed :1

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
3
The elements of the stack are:
1
2
```

```
3
2
1

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
4
Stack contents are Palindrome

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
2
The poped element is 1

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
4
Stack contents are not palindrome

 STACK OPERATIONS

 1.Push
 2.Pop
 3.Display
 4.Palindrome
 5.Exit
Enter your choice
5

Process returned 4199352 (0x4013B8)   execution time : 647.215 s
Press any key to continue.
```

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

/*
In infix notation, Example: a - b + c, operators are used in between operands.
It is easy for humans to read, write and speak in infix notation. An algorithm to
process infix notation could be difficult and costly in terms of time and space.
*/

/*
The program has 5 functions. First is the main function.
push(char) function is used to push the symbol into stack.
'char' here represents that only one symbol can be pushed into stack at once.
pop() function is used to pop the symbol from the stack.
priority() functon is used to depict the priority of operators.
infixtopostfix() is the actual function that converts infix to postfix.
*/

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

void push(char);

char pop();

int priority(char);

void infixtopostfix();


int top=-1;               //Indicates that initially the stack is empty.

char infix[30];          // Represents Infix Expression
char postfix[30];       //  Represents postfix Expression
char stack[30];        // The stack that is used for conversion

/*
The stack is used to hold operators rather than numbers.
The purpose of the stack is to reverse the order of the operators in the
expression.
It also serves as storage structure,since no operator can be printed until
both of its operands have appeared.
*/

void main()
{
    printf("Enter the valid Infix expression \n");
    scanf("%s",infix);

    infixtopostfix();                                  //Function call

    printf("The Infix expression is : %s\n",infix);
    printf("The Postfix expression is: %s\n",postfix);
}

// Function to push the symbol on to the stack
void push(char item)
{
    stack[++top]=item;
}

//Function to pop the item from the stack.
```

```c
char pop()
{
    return stack[top--];
}

/*
Function to check the priority of the operators.
Higher priorities operators will be executed first in conversion.
*/

int priority(char symb)
{
    int p;                          //Represents priority number.
    switch(symb)
    {
        case '+':
        case '-': p=1;
        break;

        case '*':
        case '/':
        case '%': p=2;
        break;

        case '^': p=3;
        break;

        case '(':
        case ')': p=0;
        break;

        case '#': p=-1;
        break;

    }
    return p;
}

/*
First the infix expression is scanned from first to last.
In case of '(' , we push all symbols inside the '(' to top of stack.
In case of  ')', we pop symbol from top of stack.
We continue to pop all symbols from top of stack until '(' is encounterd.
We later store all these popped symbols in postfix.
*/

void infixtopostfix()
{
    int i=0,j=0;
    char symb;
    char temp;

    push('#');                          // Pushing operators into stack

    for(i=0;infix[i]!='\0';i++)             //Scan infix from first to last.
        {
            symb=infix[i];

            switch(symb)
            {
                case '(': push(symb);
                          break;

                case ')': temp=pop();

                while(temp!='(')
```

```
                        {
                            postfix[j++]=temp;
                            temp=pop();
                        }
                        break;

                case'+':
                case'-':
                case'*':
                case'/':
                case'%':
                case'^':
                case'$': while(priority(stack[top])>=priority(symb))
                         {
                             temp=pop();
                             postfix[j++]=temp;

                         }

                push(symb);
                break;

                default: postfix[j++]=symb;
            }
        }

/*
while scanning of all symbols is done but still some symbols are in
stack(top>0), then we pop all the remaining all symbols
from top of stack and store to postfix.
*/
        while(top>0)
            {
                temp=pop();
                postfix[j++]=temp;
            }
            postfix[j]='\0';                     // end string postfix
}
```

OUTPUT:

```
Enter the valid Infix expression
a+b
The Infix expression is : a+b
The Postfix expression is: ab+


Enter the valid Infix expression
a+b(c*d)
The Infix expression is : a+b(c*d)
The Postfix expression is: abcd*+


Enter the valid Infix expression
(A+B/C*(D+C)-F)
The Infix expression is : (A+B/C*(D+C)-F)
The Postfix expression is: ABC/DC+*+F-
```

```c
/*
Program 5a: Design, Develop and Implement a Program in C for the
Evaluation of Suffix expression with single digit operands and
operators: +, -, *, /, %,^
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

//Here evaluate is a user defined function.

double evaluate(char symbol, double op1, double op2)
{
    switch(symbol)                          //Various operations
    {
        case '+': return (op1+op2);
        case '-': return (op1-op2);
        case '*': return (op1*op2);
        case '/': return (op1/op2);
        case '$':
        case '^': return pow(op1,op2); //same operation even for upward arrow symbol
    }
}

/*
Here $ and ^ have the same meaning and does the same function of calcaulating
the Power of the operands.
i.e Operand 1 to the power of operand 2.
This is why we have not mentioned return value for $ in switch sttement above
*/

void main()
{
    double A[20];               //Name of the stack.
    double result;             // Stores the evaluated result
    double op1, op2;          // Stores the two operators
    int i, top;
    char postfix[20];          /* Stores the postfix expression.
                                  Here Postfix expression is stored as a string */

    char symbol;              //Stores the symbols - +,-,*,/,%,^

    printf("Enter the postfix expression:\n");
    scanf("%s",postfix);

    top=-1;                           //Refers to top of the stack or empty stack

    for(i=0;i<strlen(postfix);i++)     //Till the entire length of Stack
    {
        symbol=postfix[i];

/*
Checking for a digit. Since postfix is considered as a string,
we are going to subtract ASCII value of 0.(ASCII value of 0 is 48)

if the scanned symbol is an operand, we push symbol directly onto stack.
*/
        if(isdigit(symbol))
            A[++top]=symbol-'0';      //If only operand is encountered, add to Stack

/*
if the scanned symbol is an operator, we do evaluation first and
then the result of the evaluation is put back in the stack
*/
```

```
67          else
68          {
69              op2=A[top--];               //If operator is encountered, do evaluation.
70              op1=A[top--];
71              result=evaluate(symbol,op1,op2);
72              A[++top]=result;            //Push the evaluated result also onto stack.
73          }
74      }
75
76      result=A[top--];                   //Get the final result that is there in stack.
77
78      printf("The value is : %f",result); //Prints the final result.
79
80  }
```

Output:

```
Enter the postfix expression:
456*+
The value is : 34.000000


Enter the postfix expression:
231*+9-
The value is : -4.000000


Enter the postfix expression:
623+-382/+*2^3+
The value is : 52.000000
```

```c
/*
5b: Design,Develop and Implement a Program for Solving Tower of Hanoi problem
    with n disks.
*/

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

void towers(int,char,char,char);                //Tower of Hanoi Function

int main()
{
    int n;                                      //Number of disks.

    printf("Enter the number of disks\n");
    scanf("%d", &n);

    printf("The sequence of moves are:\n\n");

    towers(n,'A','C','B');              /*Calling tower of Hanoi Function
                                        A : Source, B: Intermediate, C: Destination*/
    return 0;
}

void towers(int n, char frompeg, char topeg, char auxpeg)
{
    if(n==1)                    //If only one disk is there, just moving it directly.
    {
        printf("Move the disk 1 from peg %c to peg %c\n",frompeg,topeg);
        return;
    }

/*
The below function is recursive in nature.
(1) Moving the top n - 1 disks from peg A to peg B :A?B
(2) Moving the top disk from peg A to peg C: A?C.
(3) Moving the top n - 1 disks from peg B to peg C : B?C
*/

    towers(n-1,frompeg,auxpeg,topeg);           //if number of disks is n>1

    printf("Move the disk %d from peg %c to peg %c\n", n, frompeg,topeg);
    //(Printing the moves of discs)

    towers(n-1,auxpeg,topeg,frompeg);

}
```

OUTPUT:

```
Enter the number of disks
1
The sequence of moves are:

Move the disk 1 from peg A to peg C


Enter the number of disks
2
The sequence of moves are:

Move the disk 1 from peg A to peg B
Move the disk 2 from peg A to peg C
Move the disk 1 from peg B to peg C


Enter the number of disks
3
The sequence of moves are:

Move the disk 1 from peg A to peg C
Move the disk 2 from peg A to peg B
Move the disk 1 from peg C to peg B
Move the disk 3 from peg A to peg C
Move the disk 1 from peg B to peg A
Move the disk 2 from peg B to peg C
Move the disk 1 from peg A to peg C
```

```c
/*
6. Design, Develop and Implement a menu driven Program in C
for the following operations on Circular QUEUE of Characters
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
*/

/*
Initially front is initialised to Zero & rear is initialised to -1 indicating queue is empty
We are going to create 3 functions- insert, delete, display.
(front == 0 && rear == MAX-1) means front is pointing at first element and rear is pointing at last element.
(front>0 && rear == front-1)
(front==0)&&(rear==-1)) indicates that queue is empty.
*/
#include <stdio.h>
#include <stdlib.h>

#define MAX 5 // max size of queue is 5, can store 5 elements.
int Q[5];

int front=0;
int rear=-1;

void main()
{
    void insert();  //Insert function
    void delete();  //Delete funcion
    void dispay();  //Display function

    int ch;

    printf("Circlar Queue operations\n");

    printf("\n 1.Insert\n 2.Delete\n 3.Display\n 4.Exit\n");

    while(1)     //The loop keeps on executing recursively based on the choices.
    {
        printf("Enter your choice:\n");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1: insert();
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
            case 4: exit(1);

            default: printf("Invalid option\n");

        }
    }

} //end of main function.


//Defining Insert Function
void insert()
{
```

```
65      int x; //element to be inserted.
66      if((front == 0 && rear == MAX-1)||(front>0 && rear == front-1)) //Checking for
Overflow condition
67
68          printf("Queue overflow\n"); //if both the conditions are true
69
70      else
71          {
72          printf("Enter the element to be inserted\n"); //If Queue is not full, we
insert values.
73          scanf("%d",&x);
74          }
75
76          if(front>0 && rear == MAX-1)
77          {
78              rear=0;
79              Q[rear]=x; //inserting the element at Q[0] position.
80          }
81
82          else
83          {
84              if((front == 0 && rear==-1)||(rear!= front-1))
85              Q[++rear]=x; //increment rear and place the element pointed by rear
86          }
87  } //end of insert function.
88
89
90  //Defining Delete Function
91  void delete()
92  {
93      int a;
94      if((front == 0)&&(rear == -1)) //queue is empty.
95      {
96          printf("Queue underflow\n");
97          exit(1);
98      }
99
100     if(front == rear) //both front and rear pointing to same location.
101     {
102         a=Q[front];
103         rear=-1;
104         front=0;
105     }
106
107     else if(front == MAX-1)
108     {
109         a=Q[front];
110         front=0;
111     }
112
113     else
114         a=Q[front++];
115
116     printf("Deleted element is %d\n",a);
117 }//end of delete function.
118
119 //Defining Display Function
120 void display()
121 {
122     int i,j; //i is used with rear and j is used with front end.
123
124     if(front == 0 && rear == -1) //queue is empty.
125     {
126         printf("Queue doesnt have any element\n");
127         exit(1);
128     }
```

```
129
130     if(front>rear) //Some elements are already inserted in queue
131     {
132         for(i=0;i<=rear;i++)
133             printf("\t%d",Q[i]);
134
135 /*
136 starting from first element (i=0) upto last element inserted which is pointed by
rear (i<=rear),
137 we need to display all the elements.
138 */
139         for(j=front;j<=MAX-1;j++)
140             printf("\t%d",Q[j]);
141 /*
142 starting from element pointed by front(j=front) upto last element(MAX-1),
143 we need to display all the elements.
144 */
145         printf("\n front is at %d position\n", Q[front]);
146         printf("\n Rear is at %d position\n", Q[rear]);
147     }
148
149     else
150     {
151         for(i=front;i<=rear;i++)
152             printf("\t%d",Q[i]);
153
154         printf("\n front is at %d position\n", Q[front]);
155         printf("\n Rear is at %d position\n", Q[rear]);
156     }
157 } //end of display function.
```

OUTPUT:

```
Circlar Queue operations

 1.Insert
 2.Delete
 3.Display
 4.Exit
Enter your choice:
2
Queue underflow


 1.Insert
 2.Delete
 3.Display
 4.Exit
Enter your choice:
3
Queue doesnt have any element.


  1.Insert
 2.Delete
 3.Display
 4.Exit
Enter your choice:
1
Enter the element to be inserted
10
Enter your choice:
1
Enter the element to be inserted
20
Enter your choice:
1
Enter the element to be inserted
30
Enter your choice:
1
Enter the element to be inserted
40
Enter your choice:
1
Enter the element to be inserted
50
Enter your choice:
1
Queue overflow
```

```
Enter your choice:
3
        10      20      30      40      50
 front is at 10 position

 Rear is at 50 position


Enter your choice:
2
Deleted element is 10
Enter your choice:
2
Deleted element is 20
Enter your choice:
2
Deleted element is 30
Enter your choice:
2
Deleted element is 40
Enter your choice:
3
        50
 front is at 50 position

 Rear is at 50 position
Enter your choice:
1
Enter the element to be inserted
60
Enter your choice:
1
Enter the element to be inserted
70
Enter your choice:
1
Enter the element to be inserted
80
Enter your choice:
3
        60      70      80      50
 front is at 50 position

 Rear is at 80 position


Enter your choice:
2
Deleted element is 50
Enter your choice:
```

```
2
Deleted element is 60
Enter your choice:
2
Deleted element is 70
Enter your choice:
2
Deleted element is 80
Enter your choice:
2
Queue underflow
```

```c
/*
7. Design, Develop and Implement a menu driven Program in C for the following
   operations on Singly Linked List (SLL) of Student Data with the fields:
   USN, Name, Branch, 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
*/

/*
We create a structure to make sure different data types are used.
usn, name, branch are of 'char' datatype.
sem,phno are of 'int' datatype.
It also has a link pointer variable.

FISRT is a pointer variable that points to the first node in the linked list.

The program has 10 functions-
1. main function
2. Creating a node.
3. getnode()
4. read()
5. CreateSLL()
6. displaycount()
7. insertfront()
8. insertend()
9. deletefront()
10. deleteend()

getnode() is used to create a new node everytime the function is called.
This new node is initialised to NULL.
*/

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

//creating a structure node
struct node             // structure to create a NODE of Student info
{
    char usn[20],name[10],branch[5];
    int sem,phno;
    struct node *link;
};

typedef struct node * NODE;            //renaming node as NODE

/*
We declare two structure variables temp & FIRST.
temp will be used to create the new node.
FIRST represents the First node in the Single Linked List.
*/
NODE temp;
NODE FIRST=NULL; //FIRST is initialised to NULL.

void main()
{
    int ch;
    while(1)
        {
            printf("1- Create SLL\n");
            printf("2- Display SSL\n");
            printf("3- Insertfront\n");
            printf("4- Insert end\n");
            printf("5- Delete front\n");
```

```
            printf("6- Delete end\n");
            printf("7- EXIT\n");

            printf("Enter your choice\n");
            scanf("%d",&ch);

            switch(ch)
            {
                case 1:createSLL();
                       break;
                case 2:displaycount();
                       break;
                case 3:insertfront();
                       break;
                case 4:insertend();
                       break;
                case 5:deletefront();
                       break;
                case 6:deleteend();
                       break;
                case 7:return;
                default:printf("\n Invalid choice\n");
            }
        }
}//end of main function.

/*
getnode() function is used to create nodes in the SLL.
The memory is allocated dynamically using malloc() function.
*/
NODE getnode() //to create a linked list.
{
    NODE x;                                  //Create a node x
    x=(NODE)malloc(sizeof(struct node)); //dynamically allocate size
    x->link=NULL;                    //create a node that doesn't have a next node.
    return x;
}//end of getnode() function.

void read()                          // Read student details
{
    temp=getnode();             //create a new node and put following details

    printf("Enter USN:\n");
    scanf("%s",temp->usn);

    printf("Enter Name\n");
    scanf("%s",temp->name);

    printf("Enter Branch\n");
    scanf("%s",temp->branch);

    printf("Enter Semester\n");
    scanf("%d",&temp->sem);

    printf("Enter phno:\n");
    scanf("%d",&temp->phno);
}//end of read function

/*
Creating a SLL of N Students data by using front insertion.
First we check whether the list is empty.
If the list is empty, the newnode inserted will itself become first node.
Otherwise we insert the node at first position
*/
void createSLL()
{
```

```c
    int i,n;

    printf("Enter the number of students\n");
    scanf("%d",&n);

    for(i=1;i<=n;i++) //i=1 represents 1st student.
        {
            printf("Enter the details of student %d\n",i);
            read();            //function call to read the student details.

            if(FIRST==NULL) //if the first points to NULL
                FIRST=temp; //make temp as first node.
            else
                {
                    temp->link=FIRST;
                    FIRST=temp;         //making new node as first node.
                }
        }
}

/*
Displaying the status of SLL and count the number of nodes in it.
count is used to store the number of nodes.
First we check if the list is empty.
If list is empty, we consider new node as first node.
If not, we display the student details until temp points to NULL.
NULL means we have reached the end of the Linked List.
*/
void displaycount()
{
    int count=0;               //Initially the number of nodes in linked list=0
    temp=FIRST;               //make temp as the first node

    if(FIRST==NULL)                         // check for empty list
        printf("Student List is empty\n");
    else
        {
            printf("Student details is:\n");
            printf("USN\t Name\t Branch\t Sem\t Phno\n");

            while(temp!=NULL)          //Till we reach the NULL(last) node.
                {
                    count++;             //Keep incrementing no.of nodes.

                    printf("%s\t %s\t %s\t %d\t %d\n",
                           temp->usn,temp->name,temp->branch,
                           temp->sem,temp->phno);

                    temp=temp->link;      //Move ahead from node to node.
                }
            printf("The number of nodes is %d\n",count);
        }
    return;
}//end of displaycount function


/*
Performing Insertion at the front of SLL.
First we check list is empty.
If list is empty, we consider new node as first node.
Otherwise we insert at front of linked list.
*/
void insertfront()
{
    printf("Enter the details of student\n");
    read();
```

```c

    if(FIRST == NULL)
        FIRST=temp;
    else
        {
            temp->link=FIRST;
            FIRST=temp;
        }
}

/*
Performing Insertion at the end of SLL.
First we check for empty list.
If list is empty, we consider new node as first node.
'last->link!= NULL' is used to reach the last node.
Once it reaches the last node, we make that as 'last' node.
Then after the last node, we insert newnode using temp.
*/
void insertend()
{
    NODE last=FIRST; //making last node as the first node.

    printf("Enter the details of student\n");
    read();
    if(FIRST==NULL)
        FIRST=temp;
    else
        {
            while(last->link!= NULL)   // loop to reach last node
            last=last->link;
            last->link = temp;
        }
}

/*
Performing deletion at the front of SLL.
First we check for empty list.
If not empty, we start to delete from front node of list.
Deletion is done based on unique number 'usn'.
We make the second node as first node & then delete first node.
*/
void deletefront()
{
    temp=FIRST;                //First node is assigned to temp.

    if(FIRST==NULL)
        printf("List is empty\n");
    else
        {
            printf("deleted element is %s\n",temp->usn);
            FIRST=FIRST->link;    //making second node as first node
            free(temp);          // deleting first node.
        }
    return;
}

/*
Performing Deletion at the end of SLL.
We assign last node to NULL.
First we check for empty list.
If not empty,we check if linked list has only one node & delete that.
If both these conditions are false, we delete node from the end of list.
*/

void deleteend()
{
```

```
265      NODE last=NULL;       //last node made as NULL.
266
267      temp=FIRST;          //First node is assigned to temp.
268
269      if(FIRST == NULL)
270          printf("List is empty\n");
271
272      else if(FIRST->link == NULL)    //Means has only one node.
273          {
274              printf("Deleted element is %s\n",temp->usn);
275              free(FIRST);
276              FIRST=NULL;
277          }
278          else
279              {
280                  while(temp->link!=NULL)   // loop to reach last node
281                  {
282                      last=temp;
283                      temp=temp->link;     //Reaches last node.
284                  }
285                  last->link=NULL;
286                  printf("Deleted element is %s\n",temp->usn);
287                  free(temp);
288              }
289              return;
290  }
```

OUTPUT:

```
1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
2
Student List is empty

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
5
List is empty

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
6
List is empty

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
1
Enter the number of students
3
Enter the details of student 1
Enter USN:
111
Enter Name
```

```
ABC
Enter Branch
CSE
Enter Semester
3
Enter phno:
9890
Enter the details of student 2
Enter USN:
222
Enter Name
XYZ
Enter Branch
ISE
Enter Semester
5
Enter phno:
7650
Enter the details of student 3
Enter USN:
333
Enter Name
PQR
Enter Branch
ECE
Enter Semester
7
Enter phno:
6789

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
2
Student details is:
USN      Name    Branch  Sem     Phno
333      PQR     ECE     7       6789
222      XYZ     ISE     5       7650
111      ABC     CSE     3       9890
The number of nodes is 3

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
```

```
5- Delete front
6- Delete end
7- EXIT
Enter your choice
3
Enter the details of student
Enter USN:
444
Enter Name
JKL
Enter Branch
ME
Enter Semester
4
Enter phno:
9234

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
3
Student details is:
USN      Name    Branch  Sem     Phno
444      JKL     ME      4       9234
333      PQR     ECE     7       6789
222      XYZ     ISE     5       7650
111      ABC     CSE     3       9890
The number of nodes is 4

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
4
Enter the details of student
Enter USN:
666
Enter Name
DEF
Enter Branch
CV
```

```
Enter Semester
2
Enter phno:
6543

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
2
Student details is:
USN      Name    Branch  Sem     Phno
444      JKL     ME      4       9234
333      PQR     ECE     7       6789
222      XYZ     ISE     5       7650
111      ABC     CSE     3       9890
666      DEF     CV      2       6543
The number of nodes is 5

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
5
deleted element is 444

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
6
Deleted element is 666

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
```

```
6- Delete end
7- EXIT
Enter your choice
2
Student details is:
USN      Name    Branch  Sem     Phno
333      PQR     ECE     7       6789
222      XYZ     ISE     5       7650
111      ABC     CSE     3       9890
The number of nodes is 3

1- Create SLL
2- Display SSL
3- Insertfront
4- Insert end
5- Delete front
6- Delete end
7- EXIT
Enter your choice
7
```

```c
/*
Q8. Design, Develop and Implement 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.
*/

/*
We are creating 8 functions:
1. main() function.
2. getnode() function
3. read() function
4. CreateDLL() function.
5. displaycount() function
6. Insertionfront() function
7. Deletionfront() function
8. Deletionend() function
*/

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

struct node                  //structure to store employee details
{
    char ssn[10],name[10],dept[15],desig[10];
    int phno, sal;
    struct node *next;
    struct node *prev;
};

typedef struct node *NODE;         // Renaming struct node as NODE

NODE temp;
NODE FIRST=NULL;
NODE END=NULL;

void main()
{
    int ch;
    while(1)
        {
            printf("1 - Create DLL of N Employees\n");
            printf("2 - Display DLL\n");
            printf("3 - Insertion at front\n");
            printf("4 - Insertion at end\n");
            printf("5 - Deletion at front\n");
            printf("6 - Deletion at end\n");
            printf("7 - Exit\n");

            printf("Enter Your Choice: ");
            scanf("%d",&ch);

            switch(ch)
            {
                case 1:CreateDLL();
                       break;

                case 2:displaycount();
                       break;

                case 3:Insertionfront();
```

```
                        break;

                case 4:Insertionend();
                        break;

                case 5:Deletionfront();
                        break;

                case 6:Deletionend();
                        break;

                case 7:return;
                default: printf("Invalid Choice\n");

            }
        }
}//end of main

/*
Creating a node x with both left and right links.
Initialize them with NULL values.
A single node will not have a address of preceding element or the next element.
Hence both the values are initialised to NULL.
*/

NODE getnode()
{
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    x->next=NULL; //next node address
    x->prev=NULL; //previous node address.
    return x;
}

/*
Based on the number of employees, we create that many nodes.
For each node, we need to put in all the employee details.
All this is done using read() function.
temp is used generate nodes.
*/

void read()                         // read details of employee
{
    temp=getnode();

    printf("Enter SSN:");
    scanf("%s",temp->ssn);
    printf("Enter Name:");
    scanf("%s",temp->name);
    printf("Enter Dept:");
    scanf("%s",temp->dept);
    printf("Enter Designation:");
    scanf("%s",temp->desig);
    printf("Enter Phno:");
    scanf("%d",&temp->phno);
    printf("Enter Salary:");
    scanf("%d",&temp->sal);

    return;
}

/*
Creating a DLL of 'n' Employees by using endinsertion.
First we check whether the list empty or not.
If its empty, then the new node that we inserted will be first node.
Otherwise we find the last node and insert the new node after that.
```

```c
*/

void CreateDLL()
{
    int n;
    int i=1;
    printf("Enter the number of employees\n");
    scanf("%d",&n);
    while(i<=n)
        {
            printf("Enter the details of number %d employee\n", i++);
            read();
            if(FIRST==NULL)
                {
                    FIRST=temp;
                    END=temp;
                }
            else
                {
                    END->prev=temp;
                    temp->next=END;
                    END=temp;
                }
        }                           // end of while statement
}                                  //  end of create() function

/*
Display the status of DLL and count the number of nodes in it.
First we check wheter the list is empty or not.
If empty, we say 'No employee data'.
Otherwie we display all nodes in the list.
*/
void displaycount()
{
    temp=FIRST;
    int count=0;

    if(FIRST==NULL) //
        {
            printf("No employee data\n");
        }
    else
        {
            while(temp!=NULL) //
            {
                count++;
                printf("Employee details:\n");
                printf("%s\t%s\t%s\t%s\t%d\t%d\n",
                       temp->ssn, temp->name, temp->dept,
                       temp->desig, temp->phno, temp->sal);

                temp=temp->prev;
            }

            printf("Employee count is %d\n",count);

        }                           // end of else statement.
return;
}                                  // end of display() function.

/*
Performing Insertion at front of DLL.
First we check for empty list.
If empty we set new node as first node.
Otherwise if list is already present, we insert the node at front of list.
*/
```

```c

void Insertionfront()
{
    printf("Enter the details of the employee\n");
    read();

    if(FIRST==NULL)
        FIRST=temp;

    else                          //Inserting at front of the list.
        {
            temp->prev=FIRST;
            FIRST->next=temp;
            FIRST=temp;
        }
}                               // end of insertionfront() function.

/*
Performing Insertion at the end of DLL.
First we check for empty list.
If empty, newnode itself will be first and last node.
Otherwise if list already present,we find the last node & insert after that.
*/

void Insertionend() //Perform Insertion at End of DLL
{
    printf("Enter the details of the new employee\n");
    read();
    if(FIRST==NULL)           // check for empty list
        {
            FIRST=temp;
            END=temp;
        }
    else        // otherwise find the last node and insert the new node
        {
            END->prev=temp;
            temp->next=END;
            END=temp;
        }
    return ;
}                                // end of insertionend() function.

/*
Deleting the node from the front of the DLL.
First we check if the list is empty.
If not we check, if the list has only one node.
If only one node, we delete that node & initialise list to NULL.
Deletion of the list is done based on unique number ssn.
Otherwise we go delete first node from the list.
*/
void Deletionfront()
{
    temp = FIRST;
    if(FIRST == NULL)                       // check for empty list
        printf("List is empty\n");

    else if(FIRST == END)              // check for single node in list
        {
            printf("deleted employee is %s\n", temp->ssn);
            FIRST = NULL;
            END = NULL;
            free(temp);
        }
        else                  // otherwise delete node from front of DLL
            {
                printf("deleted employee is %s\n", temp->ssn);
```

```
                FIRST = FIRST->prev;
                FIRST->next = NULL;
                free(temp);
            }

        return;
}                                   // end of deletefront() function.
/*
Deleting the node from the end of the DLL.
First we check if the list is empty.
If not, we check if the list has only one node.
If only one node, we delete that node & initialise list to NULL.
Deletion of the list is done based on unique number ssn.
Otherwise we go delete last node from the list.
*/
void Deletionend()
{
    temp = END;
    if(FIRST==NULL)                     // check for empty list
        printf("List is empty\n");

    else if(FIRST==END)                 // check for single node in list
        {
            printf("deleted employee is %s\n", temp->ssn);
            FIRST=NULL;
            END=NULL;
            free(temp);
        }
        else                        // otherwise delete end node from DLL
            {
                printf("deleted employee is %s\n", temp->ssn);
                END = END->next;
                END->prev = NULL;
                free(temp);
            }
        return ;
}                               // end of deleteend() function.
```

OUTPUT:

```
1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 2
No employee data

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 5
List is empty

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 6
List is empty

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 1
Enter the number of employees
3
Enter the details of employee 1
Enter SSN:111
Enter Name:ABC
Enter Dept:ISE
Enter Designation:PROF
Enter Phno:9870
Enter Salary:5000
Enter the details of employee 2
```

```
Enter SSN:222
Enter Name:XYZ
Enter Dept:CSE
Enter Designation:PRIN
Enter Phno:3456
Enter Salary:6000
Enter the details of employee 3
Enter SSN:333
Enter Name:PQR
Enter Dept:ECE
Enter Designation:HODD
Enter Phno:6750
Enter Salary:4000

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 2
Employee details:
111     ABC     ISE     PROF    9870    5000
Employee details:
222     XYZ     CSE     PRIN    3456    6000
Employee details:
333     PQR     ECE     HODD    6750    4000
Employee count is 3

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 3
Enter the details of the employee
Enter SSN:444
Enter Name:STR
Enter Dept:MEE
Enter Designation:ATTD
Enter Phno:7658
Enter Salary:3000

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
```

```
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 2
Employee details:
444     STR     MEE     ATTD    7658    3000
Employee details:
111     ABC     ISE     PROF    9870    5000
Employee details:
222     XYZ     CSE     PRIN    3456    6000
Employee details:
333     PQR     ECE     HODD    6750    4000
Employee count is 4

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 4
Enter the details of the new employee
Enter SSN:555
Enter Name:KLM
Enter Dept:CVV
Enter Designation:INST
Enter Phno:5432
Enter Salary:2000

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 2
Employee details:
444     STR     MEE     ATTD    7658    3000
Employee details:
111     ABC     ISE     PROF    9870    5000
Employee details:
222     XYZ     CSE     PRIN    3456    6000
Employee details:
333     PQR     ECE     HODD    6750    4000
Employee details:
555     KLM     CVV     INST    5432    2000
Employee count is 5
```

```
1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 5
deleted employee is 444

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 2
Employee details:
111     ABC     ISE     PROF    9870    5000
Employee details:
222     XYZ     CSE     PRIN    3456    6000
Employee details:
333     PQR     ECE     HODD    6750    4000
Employee details:
555     KLM     CVV     INST    5432    2000
Employee count is 4

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 6
deleted employee is 555

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 2
Employee details:
111     ABC     ISE     PROF    9870    5000
Employee details:
222     XYZ     CSE     PRIN    3456    6000
```

```
Employee details:
333     PQR     ECE     HODD    6750    4000
Employee count is 3

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 2
Employee details:
111     ABC     ISE     PROF    9870    5000
Employee details:
222     XYZ     CSE     PRIN    3456    6000
Employee details:
333     PQR     ECE     HODD    6750    4000
Employee count is 3

1 - Create DLL of N Employees
2 - Display DLL
3 - Insertion at front
4 - Insertion at end
5 - Deletion at front
6 - Deletion at end
7 - Exit
Enter Your Choice: 7
```

```c
/*
DS9:Design, Develop and Implement a Program in C for the following operations on
    Singly Circular Linked List (SCLL) with header nodes.
    a. Represent & Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
    b. Find the sum of two polynomials POLY1(x,y,z) & 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.
*/

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

#define COMPARE(x, y) ((x == y)? 0 : (x > y)? 1 : -1)

/*
coef:Coefficient of each term.
xexp,yexp,zexp: Powers of x,y,z respectively.

The following functions are used in the program:
get node() function
NODE attach() function
NODE readpoly() function
void display() function
int polyevaluate() function
NODE polysum() function
main() function
*/

struct node
{
    int coef;
    int xexp, yexp, zexp;
    struct node *link;
};

typedef struct node *NODE;           //struct node is renamed to NODE

//get node() function is used to allocate memeory to each node that is created.
NODE getnode()
{
    NODE x;
    x = (NODE) malloc(sizeof(struct node));
    return x;
}

//temp is pointer variable that is used to add new node or display node.
NODE attach(int coef, int xexp, int yexp, int zexp, NODE head)
{
    NODE temp;
    NODE cur;   //cur is pointer variable that represents the current node

    temp = getnode();

    temp->coef = coef;
    temp->xexp = xexp;
    temp->yexp = yexp;
    temp->zexp = zexp;

    cur = head->link;                    //Points to the first node.

    while(cur->link != head)            //till the end of circular list
        {
            cur = cur->link;           //keep on moving forward.
        }
```

```
        cur->link = temp;          //attach the new node to the circular list.
        temp->link = head;        //link the attached node back to head.
        return head;
}

/*
Reads the Polynomial.
Takes the input of Number of terms, Co-efficient and powers of X,Y,Z)
*/
NODE readpoly(NODE head)
{
    int i, j;
    int n;
    int coef, xexp, yexp, zexp;

    printf("\n Enter the no of terms in the polynomial: ");
    scanf("%d",&n);

    for(i=1; i<=n; i++)                    //from 1st term to nth term
        {
            printf("\n Enter the %d term: ",i);

            printf("\n Coef =  ");
            scanf("%d", &coef);

            printf("\n Enter Pow(x) Pow(y) and Pow(z): ");
            scanf("%d", &xexp);
            scanf("%d", &yexp);
            scanf("%d", &zexp);

            head = attach(coef, xexp, yexp, zexp, head); //Put all this in a node
        }

        return head; //All the details will be in the head node.
}

void display(NODE head)
{
    NODE temp;

    if(head->link == head)         //if there is no circular linked list
        {
            printf("\n Polynomial does not exist");
            return;
        }

        temp = head->link;             //Initialise from first node

        while(temp != head)           //till the end of circular list
            {
                printf("%dx^%dy^%dz^%d", temp->coef,temp->xexp,temp->yexp,temp->zexp);
                temp = temp->link;         //Keep moving to next term

                if(temp != head)
                    printf(" + ");  //print '+' after printing every term
            }
}

// This part evaluates the polynomial and returns the sum.
// pow is defined under math.h
int polyevaluate(NODE head)
{
    NODE poly;            //here poly is same as temp. Pointer varaible.
    int x,y,z;
    int sum = 0;
```

```
    printf("\n Enter the value of x,y and z: ");
    scanf("%d%d%d", &x,&y,&z);

    poly = head->link;          //

    while(poly != head)                    //Till the end of linked list
        {
            sum += poly->coef * pow(x,poly->xexp)* pow(y,poly->yexp) * pow(z,poly->zexp);
            poly = poly->link;             //move to next term
        }
        return sum;
}

/*
This part calculates the sum of two polynomials and returns the sum.
sum can be calculated only when exponentials (powers) of x,y,z are same
in both the terms.
If they are same, sum is calculated by adding their coefficients.
That sum is stored in new polynomial represented by head3.
*/
NODE polysum(NODE head1, NODE head2, NODE head3)
{
    NODE a,b;
    int coef;

    a = head1->link;                    //a represents first polynomial
    b = head2->link;                   //b represents second polynomial

    while(a!=head1 && b!=head2) //Till the last nodes in both polynomials
    {
        while(1)
            {
                if(a->xexp == b->xexp && a->yexp == b->yexp && a->zexp == b->zexp)
                {
                    coef = a->coef + b->coef;       //add coefficients
                    head3 = attach(coef, a->xexp, a->yexp, a->zexp, head3);
                    a = a->link;              //Move to next term in 1st polynomial
                    b = b->link;             //Move to next term in 1st polynomial
                    break;
                }

                if(a->xexp!=0 || b->xexp!=0) //if pow(x) is not equal to zero
                    {
                        switch(COMPARE(a->xexp, b->xexp)) //compare pow(x) in both poly
                        {
                            case -1: head3 = attach(b->coef, b->xexp, b->yexp, b->zexp,
head3);
                                     b = b->link;
                                     break;

                            case 0 : if(a->yexp > b->yexp)
                                     {
                                       head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                       a = a->link;
                                       break;
                                     }

                                    else if(a->yexp < b->yexp)
                                        {
                                            head3 = attach(b->coef, b->xexp, b->yexp, b->
zexp, head3);
                                            b = b->link;
                                            break;
                                        }
```

```
                                    else if(a->zexp > b->zexp)
                                        {
                                            head3 = attach(a->coef, a->xexp, a->yexp, a->
zexp, head3);
                                            a = a->link;
                                            break;
                                        }

                                    else if(a->zexp < b->zexp)
                                        {
                                            head3 = attach(b->coef, b->xexp, b->yexp, b->
zexp, head3);
                                            b = b->link;
                                            break;
                                        }

                                case 1 : head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                         a = a->link;
                                         break;
                            }
                            break;

                        } //end of comparing power of x in polynomial.

                    if(a->yexp!=0 || b->yexp!=0)
                     {
                        switch(COMPARE(a->yexp, b->yexp))
                        {
                            case -1 :   head3 = attach(b->coef, b->xexp, b->yexp, b->zexp,
head3);
                                        b = b->link;
                                        break;

                            case 0 :   if(a->zexp > b->zexp)
                                {
                                    head3 = attach(a->coef, a->xexp, a->yexp, a->zexp, head3
);
                                    a = a->link;
                                    break;
                                }

                                    else if(a->zexp < b->zexp)
                                    {
                                        head3 = attach(b->coef, b->xexp, b->yexp, b->zexp
, head3);
                                        b = b->link;
                                        break;
                                    }

                            case 1 :    head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                        a = a->link;
                                        break;
                        }
                                        break;

                    } //end of comparing power of y in polynomial.

                    if(a->zexp!=0 || b->zexp!=0)
                        {
                            switch(COMPARE(a->zexp, b->zexp))
                            {
                                case -1: head3 = attach(b->coef,b->xexp,b->yexp,b->zexp,head3
);
                                         b = b->link;
```

```
                                    break;

                            case 1 : head3 = attach(a->coef, a->xexp, a->yexp, a->zexp,
head3);
                                    a = a->link;
                                    break;
                        }
                            break;

                    }//end of comparing power of z in polynomial.
            }//end of while loop

    }//end of while condition

    while(a!= head1) //till the end of expression in 1st polynomial.
        {
            head3 = attach(a->coef,a->xexp,a->yexp,a->zexp,head3);
            a = a->link;
        }

    while(b!= head2) //till the end of expression in 2nd polynomial.
        {
            head3 = attach(b->coef,b->xexp,b->yexp,b->zexp,head3);
            b = b->link;
        }

    return head3; //return head3 that stores sum polynomial.
}

void main()
{
    NODE head, head1, head2, head3;
    int res, ch;

    head =  getnode();                      // For polynomial evalaution

    head1 = getnode();                    // To hold POLY 1
    head2 = getnode();                   // To hold POLY 2
    head3 = getnode();                  // To hold POLYSUM

//initially only head node will be present in Linked List.(Empty Linked List)
    head->link=head;
    head1->link=head1;
    head2->link=head2;
    head3->link= head3;

    while(1)
        {
            printf("\n 1 - Represent and Evaluate a Polynomial P(x,y,z)");
            printf("\n 2 - Find the sum of two polynomials POLY(x,y,z)");

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

            switch(ch)
            {
                case 1: printf("Polynomial evaluation P(x,y,z)\n");
                        head = readpoly(head);

                        printf("\n Representation of Polynomial for evaluation: \n");
                        display(head);

                        res = polyevaluate(head);
                        printf("\n Result of polynomial evaluation is: %d\n", res);
                        break;
```

```
                case 2: printf("Enter the POLY1(x,y,z):\n");
                        head1 = readpoly(head1);
                        printf("\n Polynomial 1 is:\n");
                        display(head1);

                        printf("\n Enter the POLY2(x,y,z):\n");
                        head2 = readpoly(head2);
                        printf("\n Polynomial 2 is:\n");
                        display(head2);

                        printf("\n Polynomial sum is:\n");
                        head3 = polysum(head1,head2,head3);
                        display(head3);
                        break;

                case 3: exit(0);

            }//end of switch case

        }//end of while loop

}//end of main function
```

OUTPUT 1:

```
1 - Represent and Evaluate a Polynomial P(x,y,z)
2 - Find the sum of two polynomials POLY(x,y,z)
Enter your choice:1
Polynomial evaluation P(x,y,z)

Enter the no of terms in the polynomial: 5

Enter the 1 term:
Coef =  6

Enter Pow(x) Pow(y) and Pow(z): 2 2 1

Enter the 2 term:
Coef =  -4

Enter Pow(x) Pow(y) and Pow(z): 0 1 5

Enter the 3 term:
Coef =  3

Enter Pow(x) Pow(y) and Pow(z): 3 1 1

Enter the 4 term:
Coef =  2

Enter Pow(x) Pow(y) and Pow(z): 1 5 1

Enter the 5 term:
Coef =  -2

Enter Pow(x) Pow(y) and Pow(z): 1 1 3

Representation of Polynomial for evaluation:
6x^2y^2z^1 + -4x^0y^1z^5 + 3x^3y^1z^1 + 2x^1y^5z^1 + -2x^1y^1z^3
Enter the value of x,y and z: 1 1 1

Result of polynomial evaluation is: 5

1 - Represent and Evaluate a Polynomial P(x,y,z)
2 - Find the sum of two polynomials POLY(x,y,z)
 Enter your choice:2
 Enter the POLY1(x,y,z):

 Enter the no of terms in the polynomial: 3

 Enter the 1 term:
 Coef =  4
```

```
 Enter Pow(x) Pow(y) and Pow(z): 3 3 3

 Enter the 2 term:
 Coef =  3

 Enter Pow(x) Pow(y) and Pow(z): 2 2 2

 Enter the 3 term:
 Coef =  2

 Enter Pow(x) Pow(y) and Pow(z): 1 1 1

 Polynomial 1 is:
4x^3y^3z^3 + 3x^2y^2z^2 + 2x^1y^1z^1
 Enter the POLY2(x,y,z):

 Enter the no of terms in the polynomial: 3

 Enter the 1 term:
 Coef =  8

 Enter Pow(x) Pow(y) and Pow(z): 3 3 3

 Enter the 2 term:
 Coef =  5

 Enter Pow(x) Pow(y) and Pow(z): 2 2 2

 Enter the 3 term:
 Coef =  4

 Enter Pow(x) Pow(y) and Pow(z): 1 1 1

 Polynomial 2 is:
8x^3y^3z^3 + 5x^2y^2z^2 + 4x^1y^1z^1

 Polynomial sum is:
12x^3y^3z^3 + 8x^2y^2z^2 + 6x^1y^1z^1

**********************************************************************

OUTPUT 2:

1 - Represent and Evaluate a Polynomial P(x,y,z)
2 - Find the sum of two polynomials POLY(x,y,z)
Enter your choice:1
Polynomial evaluation P(x,y,z)

Enter the no of terms in the polynomial: 3
```

```
Enter the 1 term:
Coef =  2

Enter Pow(x) Pow(y) and Pow(z): 3 2 1

Enter the 2 term:
Coef =  2

Enter Pow(x) Pow(y) and Pow(z): 2 3 1

Enter the 3 term:
Coef =  2

Enter Pow(x) Pow(y) and Pow(z): 1 2 3

Representation of Polynomial for evaluation:
2x^3y^2z^1 + 2x^2y^3z^1 + 2x^1y^2z^3
Enter the value of x,y and z: 1 2 1

Result of polynomial evaluation is: 32

1 - Represent and Evaluate a Polynomial P(x,y,z)
2 - Find the sum of two polynomials POLY(x,y,z)
Enter your choice:2
Enter the POLY1(x,y,z):

Enter the no of terms in the polynomial: 3

Enter the 1 term:
Coef =  2

Enter Pow(x) Pow(y) and Pow(z): 1 2 3

Enter the 2 term:
Coef =  4

Enter Pow(x) Pow(y) and Pow(z): 2 3 1

Enter the 3 term:
Coef =  -6

Enter Pow(x) Pow(y) and Pow(z): 3 2 1

Polynomial 1 is:
2x^1y^2z^3 + 4x^2y^3z^1 + -6x^3y^2z^1
Enter the POLY2(x,y,z):

Enter the no of terms in the polynomial: 3

Enter the 1 term:
```

```
Coef =  1

Enter Pow(x) Pow(y) and Pow(z): 3 2 1

Enter the 2 term:
Coef =  -2

Enter Pow(x) Pow(y) and Pow(z): 1 2 3

Enter the 3 term:
Coef =  3

Enter Pow(x) Pow(y) and Pow(z): 2 3 1

Polynomial 2 is:
1x^3y^2z^1 + -2x^1y^2z^3 + 3x^2y^3z^1

Polynomial sum is:
0x^1y^2z^3 + 7x^2y^3z^1 + -5x^3y^2z^1
```

```c
/*
DS10: Design, Develop and Implement 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
*/

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

/*
'typedef with structure' is used to define a new data type and
then use that data type to define structure variables.
If we use the typedef keyword followed by a new name, we can use the struct
by that name without writing the struct keyword.
Here we are creating a structure by name BST.
*/

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

/*
The first step is to create NODES.
NODES are places where we enter the values.
Value of elements to be inserted is asked,along with creation of left & right trees.

temp represents temporary places where values will be inserted.
Initially all values will be NULL.
*/

NODE create()    //creation of nodes which in turn creates a tree.
{
    NODE temp;
    temp = (NODE) malloc(sizeof(struct BST)); //dynamic memory allocation.
    printf("\n Enter The value: ");          //Value of elements in BST.
    scanf("%d",&temp->data);

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

//Function calls.

void insert(NODE root, NODE newnode);          //Inserting element into BST.
void inorder(NODE root);                       //Tree Traversals.
void preorder(NODE root);
void postorder(NODE root);
void search(NODE root);                     //Searching for an element in BST

/*
For insertion of elements into the node, we check two conditions:
1.If element to be inserted is less than root value, we insert as left child.
2.If element to be inserted is greater than root value, we insert as right child.
*/

void insert(NODE root, NODE newnode)
{
    if(newnode->data < root->data)           //less than root element.
```

```
        {
            if(root->lchild == NULL)
            root->lchild = newnode;          //insert as left child
            else
            insert(root->lchild, newnode);
        }

        if(newnode->data > root->data)     //greater than root element.
        {
            if (root->rchild == NULL)
            root->rchild = newnode;       //insert as right child
        else
            insert(root->rchild, newnode);
        }
}

/*
3 Traversals. Root must not be NULL for the traversals to be performed.
All 3 functions are similar except for the definition.
*/

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

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

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

/*
In search function, we perform following opeartions:
1. First check whether root is empty(whether tree is empty).
2. If not empty, we search for the element.
3. If the element to be searched is at 'root', we say element found.
4. If element not found in root, we search 'left' and 'right' tree.
*/

void search(NODE root)
{
    int key;                    //represents key element to be searched.
    NODE x;                    // x is used as a proxy to root.
    if(root == NULL)
    {
        printf("\n BST is empty");
```

```
        return;
    }

    printf("\n Enter Element to be searched: ");
    scanf("%d",&key);

    x=root;                  //Initialising root as x
    while (x!=NULL)         //While root(tree) is not empty, we search element.
    {
        if (x->data == key)            //if key is equal to root
        {
            printf("\n Key element is present in BST");
            return;
        }

        if (key > x->data)              //if key is less than root
        x = x->rchild;
        else
        x = x->lchild;
    }

        printf("\n Key element is not found in the BST"); //element not found
}

/*
Main function will have the menu.
Initially the root node is initialised to NULL meaning tree is empty.
i=1 in for loop representing tree starts from first element(root).

In 'case 1',if no node is there, we create newnode or
we insert values into already craeted nodes.
*/

void main()
{
    int ch, i, n;
    NODE root = NULL, newnode;
    while(1)
        {
            printf("\n ~~~~BST MENU~~~~ ");
            printf("\n 1. Create a BST");
            printf("\n 2. BST Traversals:");
            printf("\n 3. Search an Element");
            printf("\n 4. Exit");

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

            switch(ch)
            {
                case 1: printf("\n Enter the number of elements: ");
                        scanf("%d", &n);
                        for(i=1; i<=n; i++) //starting from 1st node/element
                            {
                                newnode = create();
                                if (root == NULL)
                                    root = newnode;
                                else
                                    insert(root,newnode);
                            }
                            break;

                case 2: if (root == NULL)
                        printf("\n Tree Is Not Created");
                        else
                            {
```

```
                                printf("\n The Preorder display: ");
                                preorder(root);
                                printf("\n The Inorder display: ");
                                inorder(root);
                                printf("\n The Postorder display: ");
                                postorder(root);
                            }
                            break;

                case 3: search(root);
                        break;

                case 4: exit(0);

            }

        }
}
```

OUTPUT 1:

```
 ~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 2
 Tree Is Not Created

 ~~~~BST MENU~~~~
 1. Create a BST
 2. BST Traversals:
 3. Search an Element
 4. Exit
 Enter your choice: 3
 BST is empty

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

The Preorder display: 6        5       2       9       8       7       15      14
  24
The Inorder display:  2        5       6       7       8       9       14      15
  24
The Postorder display: 2       5       7       8       14      24      15      9
  6

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

Enter Element to be searched: 20

Key element is not found in the BST

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

Enter Element to be searched: 5

Key element is present in BST

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

OUTPUT 2:

```
~~~~BST MENU~~~~
```

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

Enter the number of elements: 6

Enter The value: 2

Enter The value: 1

Enter The value: 3

Enter The value: 5

Enter The value: 7

Enter The value: 9

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

The Preorder display: 2        1       3       5       7       9
The Inorder display:  1        2       3       5       7       9
The Postorder display: 1       9       7       5       3       2

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

Enter Element to be searched: 3

Key element is present in BST

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

Enter Element to be searched: 8
```

```
 Key element is not found in the BST

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

OUTPUT 3:

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

 Enter the number of elements: 5

 Enter The value: 5

 Enter The value: 2

 Enter The value: 1

 Enter The value: 8

 Enter The value: 6

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

 Enter Element to be searched: 4

 Key element is not found in the BST

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

```
 Enter Element to be searched: 2

 Key element is present in BST

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

 The Preorder display: 5        2       1       8       6
 The Inorder display:  1        2       5       6       8
 The Postorder display: 1       2       6       8       5

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

```c
/*
DS11:Design, Develop and Implement 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.

The program contains 3 functions:
DFS() function
BFS() function
main() function

DFS:
Input : Adjacency matrix representation of the graph.
output: Nodes/vertices connected
        Whether graph is connected or not.
BFS:
Input : Adjacency matrix representation of the graph.
        Starting vertex
output: All the nodes/vertices that can be reached from starting vertex.
*/

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

int a[10][10];          // Two dimensional array for adjacency matrix
int q[10];             // Queue used for BFS function.
int visited[10];      // Stores all visited nodes.
int reach[10];       // Stores final reached nodes
int n;              // Number of nodes
int i,j;
int f=0,r=-1;    // f:front,r:rear (Used in queue function of BFS)

int count=0;   //Stores the number of nodes visited.
/*
if count == n-1 then all the nodes in a graph is connected.
otherwise the graph has node(s) that are not connected by any nodes.

if(0) means the statement following if condition will not be executed.
if(1) means the statement following if condition will be executed.
*/

void DFS(int v)                                 //DFS function
{
    int i; reach[v]=1;
    for(i=1;i<=n;i++)
        {
            if(a[v][i] && !reach[i])
            {
                printf("\n %d->%d",v,i);
                count++;
                DFS(i);                     //Recursive function call
            }
        }
}

void BFS(int v)                             //BFS function definition
{
    for(i=1;i<=n;i++)
        if(a[v][i] && !visited[i])
            q[++r]=i;

            if(f<=r)
                {
                    visited[q[f]]=1;
                    BFS(q[f++]);          //Recursive function call
```

```
                }
}

/*
For both DFS and BFS, the common input is -
number of vertices and adjacenecy matrix representing a graph.
*/
void main()
{
    int v, ch;

    printf("\n Enter the number of vertices:");
    scanf("%d",&n);

/*
i=1 means starting from 1st vertex.
Initially all values of 'q' array, 'visited' array and 'reach' array
are assigned with 0 value.
This value will change as we evaluate step by step.
*/
    for(i=1;i<=n;i++)
        {
            q[i]=0;
            visited[i]=0;
        }

    for(i=1;i<=n-1;i++)
        reach[i]=0;

    printf("\n Enter graph data in matrix form:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&a[i][j]);           //adjacency matrix

    printf("1.DFS\n 2.BFS\n 3.Exit\n");
    printf("Enter the choice\n");
    scanf("%d",&ch);


    switch(ch)
    {
        case 1: DFS(1);                    //Start from node 1
                if(count == n-1)
                    printf("\n Graph is connected");
                else
                    printf("\n Graph is not connected");
                    break;

        case 2: printf("\n Enter the starting vertex:");
                scanf("%d",&v);

                BFS(v);  //function call for BFS function with v value.
/*
if starting vertex 'v' is less than 1 or
greater than number of vertices 'n', then BFS is not possible.
*/
                if((v<1)||(v>n))
                {
                    printf("\n BFS not possible");
                }

                else
                    {
                        printf("\n The nodes which are reachable from %d are:\n",v);
                        for(i=1;i<=n;i++)
                            if(visited[i])
```

```
133                                 printf("%d\t",i);//Printing reachable nodes.
134                         }
135                         break;
136
137             case 3: exit(0);
138         }
139     }
```

Output 1:

```
Enter the number of vertices:4

Enter graph data in matrix form:
0 1 0 1
0 0 1 0
1 0 0 0
0 0 0 0

1.DFS
2.BFS
3.Exit
Enter the choice
1

1->2
2->3
1->4
Graph is connected


Enter the number of vertices:4

Enter graph data in matrix form:
0 1 0 0
0 0 1 0
1 0 0 0
0 0 0 0

1.DFS
2.BFS
3.Exit
Enter the choice
1

1->2
2->3
Graph is not connected


Enter the number of vertices:4

Enter graph data in matrix form:
0 1 0 1
0 0 1 0
1 0 0 0
0 0 0 0

1.DFS
```

```
2.BFS
3.Exit
Enter the choice
2

Enter the starting vertex:2
The nodes which are reachable from 1 are:
1       2       3       4
```

Output 2:

```
Enter the number of vertices:4

Enter graph data in matrix form:
0 0 1 0
0 0 1 1
1 1 0 1
0 1 1 0

1.DFS
2.BFS
3.Exit
Enter the choice
2

Enter the starting vertex:1
The nodes which are reachable from 1 are:
1       2       3       4


Enter the number of vertices:4

Enter graph data in matrix form:
0 0 1 0
0 0 1 1
1 1 0 1
0 1 1 0
1.DFS
2.BFS
3.Exit
Enter the choice
1

1->3
3->2
2->4
Graph is connected
```

```c
/*
12. 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.
Design and develop a Program in C that uses Hash function H: K -> L as
H(K)=K mod 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.
*/

#include <stdio.h>
#include <math.h>

#define MAX 10                          //Maximum size of hash table is 10 [0 to 9]]

/*
The program has 3 parts: 1-main function, 2-Linear probing, 3- Display
First we initialise the entire hash table with -11111.
Now -11111 here indicates empty places in hash table.
*/

void main()
{
    int a[MAX];
    int num;
    int i;
    int ch;

    for (i=0;i<MAX;i++)
        a[i] = -11111;              //initialize entire HT with -11111 entries

    while(1)
        {
            printf("\n ***Collision handling by Linear Probing***\n");

            printf("1 - Insert into Hash table\n");
            printf("2 - Display Hash table\n");
            printf("3 - Exit\n");

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

            switch (ch)
            {
                case 1: linearprob(a,num);  //Function call to linearprob() function
                        break;

                case 2: display(a);         //Function call to display() function
                        break;

                case 3: return;
                default: printf("Invalid Choice\n");
            }
        }
}

/*
num is 4 digit key. Not to be confused with 'key'.
key is the index value of where the number must be stored.

flag is used to specify whether key is entered or not.
By default, flag=0 specifying key is not entered.

If the collision is detected,we check for next available empty location.
If collision is detected & also space is full,we say Hash table FULL.
```

```c
*/

void linearprob(int a[MAX], int num)          //linearprob() function
{
    int flag;
    int i;
    int key;
    int count;
    char ans;
    do
        {
            flag=0;               //Specifies initially no number is entered
            count=0;             //keeps the numbers put into hash table

            printf("Enter 4 digit Key : ");
            scanf("%4d", &num);                    // reads 4-digit a number

            key=num%10;             //generates single digit key for given number

            if(a[key]== -11111)         // check for empty entry in Hash table
                a[key] = num;     //if yes then add
            else                    // if entry exits then its must avoid collision
                {
                    printf(" Collision Detected...!!!\n");

                    i=0;
                    while(i<MAX)   // check for next available empty location in HT
                        {
                            if (a[i]!=-11111)
                            count++;     //increment locations that are filled up
                            i++;
                        }                                   // end of while

                    if(count == MAX) // if HT is full then display HT and return
                        {
                            printf("\n Hash table is full \n");
                            display(a);                              // Display HT
                            return;
                        }

                    printf("Collision avoided successfully using LINEAR PROBING\n");
/*
If there is empty space after a place where collision is detected in HT then make a
entry of the num in that place in the HT.
This is represented by i=key+1 and if a[i]==-1111.
i=key+1 means start from one place after where the collision was detected.
if a[i]==-1111 means if that place is empty.
*/
                    for(i=key+1; i<MAX; i++)
                        if(a[i] == -11111)
                        {
                            a[i] = num;         //insert the num in HT
                            flag =1;          //Mark the location as occupied
                            break;
                        }
                        i=0;
/*
Check for empty space before key in HT then make a entry in HT.
If there is empty space after a place where collision is detected in HT then make a
entry of the num in that place in the HT.
This is represented by i<key and flag==0.
i<key represents the look for an empty space prior to where collision occured.
and check if its empty (flag==0)
*/
                        while((i<key) && (flag==0))
                            {
```

```
                                if(a[i] == -11111)        //if location is empty
                                {
                                    a[i] = num;        //insert the num in HT
                                    flag=1;      //Mark the location as occupied
                                    break;
                                }
                                i++;
                            }
                }

        } while(ans== 'y'||ans == 'Y');         // end of do-while statement

} // end of if statement

void display(int a[MAX])                         // display() function
{
    int i;
    printf("The hash table is \n Key \t Value\n");

    for(i=0; i<MAX; i++)
    printf(" %d\t %d\n", i, a[i]);
}
```

OUTPUT:

```
 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :2
The hash table is
 Key     Value
 0       -11111
 1       -11111
 2       -11111
 3       -11111
 4       -11111
 5       -11111
 6       -11111
 7       -11111
 8       -11111
 9       -11111

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 1234

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 2346

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 7777

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 9999

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
```

```
2 - Display Hash table
3 - Exit
Enter your Choice :2
The hash table is
 Key     Value
 0       -11111
 1       -11111
 2       -11111
 3       -11111
 4       1234
 5       -11111
 6       2346
 7       7777
 8       -11111
 9       9999

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 3456
 Collision Detected...!!!
Collision avoided successfully using LINEAR PROBING

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :2
The hash table is
 Key     Value
 0       -11111
 1       -11111
 2       -11111
 3       -11111
 4       1234
 5       -11111
 6       2346
 7       7777
 8       3456
 9       9999

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 1244
 Collision Detected...!!!
```

```
Collision avoided successfully using LINEAR PROBING

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :2
The hash table is
 Key     Value
 0       -11111
 1       -11111
 2       -11111
 3       -11111
 4       1234
 5       1244
 6       2346
 7       7777
 8       3456
 9       9999

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 5555
 Collision Detected...!!!
Collision avoided successfully using LINEAR PROBING

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :2
The hash table is
 Key     Value
 0       5555
 1       -11111
 2       -11111
 3       -11111
 4       1234
 5       1244
 6       2346
 7       7777
 8       3456
 9       9999

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
```

```
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 3457
 Collision Detected...!!!
Collision avoided successfully using LINEAR PROBING

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :2
The hash table is
 Key     Value
 0       5555
 1       3457
 2       -11111
 3       -11111
 4       1234
 5       1244
 6       2346
 7       7777
 8       3456
 9       9999

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 2222

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :2
The hash table is
 Key     Value
 0       5555
 1       3457
 2       2222
 3       -11111
 4       1234
 5       1244
 6       2346
 7       7777
 8       3456
 9       9999

 ***Collision handling by Linear Probing***
```

```
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 3333

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :2
The hash table is
 Key     Value
 0       5555
 1       3457
 2       2222
 3       3333
 4       1234
 5       1244
 6       2346
 7       7777
 8       3456
 9       9999

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
Enter your Choice :1
Enter 4 digit Key : 5666
 Collision Detected...!!!

 Hash table is full
The hash table is
 Key     Value
 0       5555
 1       3457
 2       2222
 3       3333
 4       1234
 5       1244
 6       2346
 7       7777
 8       3456
 9       9999

 ***Collision handling by Linear Probing***
1 - Insert into Hash table
2 - Display Hash table
3 - Exit
```

Enter your Choice :3