REVA UNIVERSITY

Bengaluru, India

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

Data Structures Lab

For

Third Semester

B. Tech in CSE (AIML), ISE, SE and CSIT

INDEX

1. Lab Objectives:

The objectives of this course are to:

1. Provide hands on skills on implementing different data-structures.
2. Analyze and Implement different applications of data-structures.
3. Identify and illustrate various applications of different data-structures.
4. Generate softwares as per user requirements
5. Develop industry standard data-structure applications for various domains.

2. Lab Outcomes:

On successful completion of this course; student shall be able to:

1. Design and implement a different data-structures.
2. Use appropriate data-structure for solving different problems.
3. Design and build a GUI application using a front end tool.
4. Design and develop applications based on different data structures like string reversal, sorting, searching, creating databases etc.

3.Lab Requirements:

Following are the required hardware and software for this lab, which is available in the laboratory.

- **Hardware:** Desktop system or Virtual machine in a cloud with OS installed. Presently in the Lab, Pentium IV Processor having 1 GB RAM and 250 GB Hard Disk is available. Desktop systems are dual boot having Windows as well as Linux OS installed on them.

- **Software:** C-compiler. Many C-compilers are available. As we have dual boot, we have *gcc* compiler in Linux environment (fedora 8 to fedora 20), and Turbo C-IDE on Windows Environment.

This data structure programming manual is designed to increase the practical knowledge, thinkingability and creativity of students. Here, students are expected to read, think, design and implement the programs for given problems in "C". The Linux and windows environments are used as explainedbelow.
Linux environment: All the programs have been written using "C" and tested using the *cc* compiler in Linux environment.
Steps for the successful program completion are :

1. Open vi editor using the shell command

```
$vi <filename.c>
```

2. Write the C-code according to the program logic. Save the program and exit (Esc + :wq) to get shell command prompt.
3. Compile the program using gcc compiler using the following shell command

```
$cc <filename.c>
```

Where, *cc* is the C compiler and *filename.c* is the input file name.

4. Execute the program using the following shell command

```
$./exeFileName
```

Where, *exeFileName* is the input file name.

Windows environment: Programs can be edited, compiled and executed using Turbo C/C++ IDE. All operations like opening a file, saving a file, compiling the program and executing the program are listed under different menus inside the IDE. Keyboard shortcuts are also available for the above mentioned operations.

Note: Every problem in the Lab manual includes problem statement, learning outcomes, theoretical description, algorithm, program, program description, expected results, implementation phase,simulation of syntax and logical errors, final program with results.

Recommendation: It is advised to use Linux environment for executing the programs given as part of this manual.

4. Guidelines to Students

- Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
- Students are required to carry their observation / programs book with completed exercises while entering the lab.
- Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- Lab records need to be submitted on or before date of submission.
- Students are not supposed to use flash drives.

List of Lab Exercises

Sl. No.	Name of the Lab Program
1.	Consider the Software company “XYZ” with 100 employees. Design and develop a C program to construct an array to store the id of 100 employees and use different searching techniques to search for an employee with particular employee id.
2.	Polynomials are used to graph real world curves. For example, roller coaster designers may use polynomials to describe the curves in their rides. Combinations of polynomial functions are sometimes used in economics to do cost analysis. Design and develop a C program to add two polynomials of any degree using single dimensional array.
3.	Real-life quantities that are naturally described by complex numbers rather than real numbers. Design and develop a C program to add and subtract ‘n’ complex numbers using structure.
4.	Reversal of string is used to check for palindrome or not. Design and Develop a C program to reverse a string using stack.
5.	The compilers always convert infix expression into postfix to perform further operations like parsing, lexical analysis etc. Design and Develop a Cprogram to convert an infix expression into postfix using stack.
6.	Evaluation of postfix expressions is done by compilers during the compilation process. Design and Develop a C program to evaluate a postfix expression using stack.
7.	When jobs are entered in system they are stored in the job queue and whenever the processor is free the jobs are moved to ready queue for execution. Consider a college is having a printer. Design and develop a C program to implement job scheduling in the printer. **Hint:** Store the jobs based on their job id in queue.
8.	In a machine gun bullets are always inserted in circular queue. Design and develop a C program to process bullets in machine gun. **Hint:** Bullet insertion and firing (deletion) can be implemented using circular queue. Bullets are given with specific id.

9.	Consider the university "REVA" is having 100 students where, the details of each student like name, roll no and marks of 3 subjects is to be stored. Design and develop a C program to construct a singly linked list to enter records of different students in list, display the list and calculate the percentage of each student. Also count the number of students passed(scored >40 in all the subjects).
10.	Consider the company "abc" is having 100 employees where, the details of each employee like name and id is to be stored. Design and develop a C program to construct a doubly linked list to enter records of different employees in list, display the list and print the names of all employees with particular employee id.
15.	List of assignments
16.	List of viva questions

Program 1: Searching

1 Problem Statement

Consider a Software company "XYZ" with 100 employees. Design and develop a C program to construct an array to store the id of 100 employees and use different searching techniques to search for an employee with particular employee id.

2 Student Learning Outcomes

After successful execution of this program, the student shall able to

- define and create one-dimensional array.
- implement different searching algorithms, like linear and binary search.
- apply the knowledge of array and searching techniques in problem solving.

3 Design and development of the Program

3.1 Theory

A one-dimensional array (or single dimension array) is a type of linear array. Accessing its elements involves a single subscript which can either represent a row or column index.

Declaration Syntax : **datatype Array_name[sizeofArray];**

Example: **int array[10];** /*array can store 10 integer values*/

Searching: Searching technique deals with finding a target in a collection of elements, or determines the target does not exist. Here we consider data in arrays stored in the memory; while in real problems data may be stored in disk files, databases, or even distributed over the Internet. There are basically two type of searching algorithms exists:

- Linear Search
- Binary Search

Linear Search: A linear search searches an element or value from an array till the desired element or value is not found and it searches in a sequence order. It compares the element with all the other elements given in the list and if the element is matched it returns the position. Linear Search is applied on the unsorted or unordered list when there are fewer elements in a list.

Binary Search: Binary Search is applied on the **sorted array or list**. Binary search first compares the value with the elements in the middle position of the array. If the value is matched, then returns the position. If the value is less than the middle element, then it must lie in the lower half of the array and if it's greater than the element then it must lie in the upper half of the array. We repeat this procedure on the lower (or upper) half of the array. Binary Search is useful when there are large numbers of elements in an array.

3.2 Algorithm

Algorithm to search for an element in a Single dimensional array of size "n" using linear search.

```
Linear Search ( Array XYZ, Value id)
Step 1: Set i to 0 and n to the number of elements in array
Step 2: IF i > n then go to step 7
Step 3: IF XYZ[i] = id then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element id Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
```

Algorithm to search for an element in a Single dimensional array of size "n" using Binary search .

```
Binary Search(Array XYZ, Value id)
Step 1: Set n to the number of elements in array, low← 0 and high←n-1
Step 2: Set mid←⌊(low+high)/2⌋
Step 3: While low<=high , Repeat Step 3 to 12
Step 3: IF id=XYZ[mid]
            Print Element id Found at index mid
Step 4: ELSE IF id<XYZ[mid]
            Set high←mid-1
Step 5: ELSE
            Set low←mid+1
Step 6: Set mid← ⌊(low+high)/2 ⌋
Step 7: Exit
```

3.3 Coding using C Language

Program to search for the employee with given id in a Single dimensional array of size "n" using liner search.

```c
#include <stdio.h>
main()
{
        int XYZ[100], id, i, n, count=0;                /*Declaration*/
5.          printf("Enter the number of employees\t");
6.          scanf("%d",&n);

7.          printf("Enter %d employee id(s)\t", n);
8.          for (i = 0; i < n; i++)                              /*read the array elements*/
9.                 scanf("%d", &XYZ[i]);

10.         printf("Enter the id to search\t");
11.         scanf("%d", &id);                      /*Read the id to be searched*/

12.         for (i=0;i<n;i++)                      /*For loop to scan each element of array*/
13.          {
14.                 if(XYZ[i]== id)                  /* IF required element found */
15.                 {
16.                        /*Print the location of emp with given searchid*/
17.                        printf("employee with %d id is present at location %d.\n", id, i+1);
18.                        count=1;               /*count=1 denotes existance of employee in list*/
19.                 }
20.          }

21.          if (count==0)                        /*count= =0 denotes absence of employee in list*/
22.                 printf("employee with %d id is not present in array.\t", id);

23.    }
```

Explanation of the program:

- Line 1 includes header files using preprocessor directives
- Line 4 declares different variables and array
- Lines 5 and 6 read the number of employees in the array
- Lines 7 to 9 read the employee ids
- Lines 10 and 11 read the id to be searched
- Lines 12 to 20 search for the employee id in the array and prints its location if found
- Lines 21 to 22 print “employee is not present in array” if id does not exist in the array

<u>Program to search for the employee with given id in a Single dimensional array of size “n” using Binary search.</u>

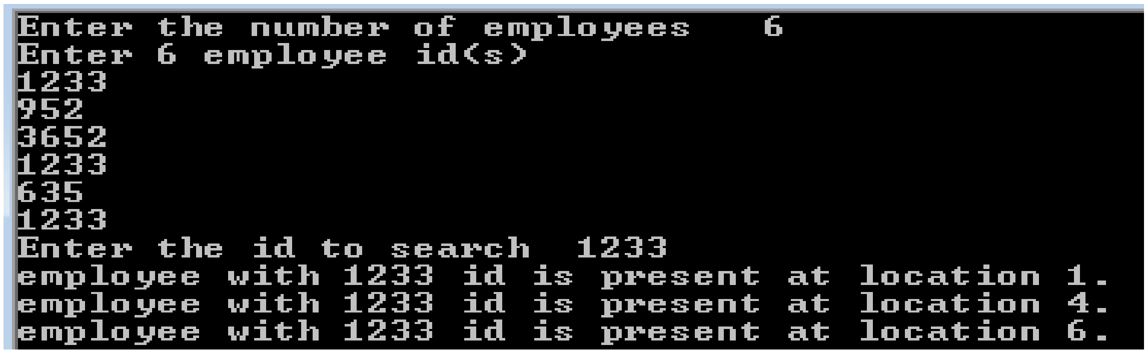
```
1.  #include<stdio.h>
2.  main()
3. {
4.     int XYZ[100], id,n,low,high,mid,i;                  /*Declaration*/

5.     printf("Enter the number of employees\t");              /*Reading number of employees*/
6.     scanf("%d",&n);
7.    printf("Enter %d employee id(s)\t", n);
8.     for (i = 0; i < n; i++)                          /*Read employee ids*/
9.          scanf("%d", &XYZ[i]);

10.   printf("Enter the id to search\t");
11.   scanf("%d", &id);                             /*Read the id to be searched*/

12.   low = 0;
13.   high = n - 1;
14.   mid = (low+high)/2;

15.   while (low <= high)
16.   {
17.          if (XYZ[mid] == id)
18.         {
19.                /*print the location if employee is found*/
20.                printf("employee with id %d is found at location %d.\n", id, mid+1);
21.                break;
22.         }
23.         else if (id < XYZ[mid])
24.                high = mid - 1;
25.         else
26.                low = mid + 1;
27.
28.          mid = (low + high)/2;
29.   }
30.   if (low > high)
31.         printf("employee with id %d is not present in the list.\n", id);
32. }
```

Explanation of the program:

- Lines 1 includes header files using preprocessor directives
- Line 4 declares different variables and array
- Lines 5 and 6 read the number of employees in the array
- Lines 7 to 9 read the employee ids
- Lines 10 and 11 read the id to be searched
- Lines 12 to 29 search for the employee id in the array and prints its location if found
- Lines 30 to 31 print “employee is not present in array” if id does not exist in the array

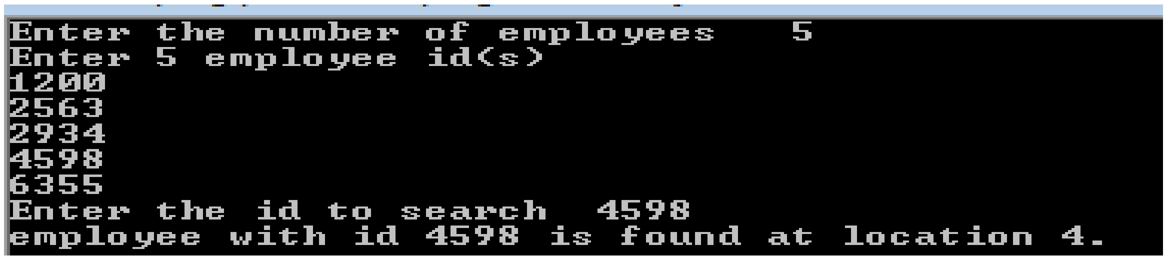
3.4	Expected Results

Result of Linear Search:



```
Enter the number of employees   6
Enter 6 employee id(s)
1233
952
3652
1233
635
1233
Enter the id to search  1233
employee with 1233 id is present at location 1.
employee with 1233 id is present at location 4.
employee with 1233 id is present at location 6.
```

Result of Binary Search:



```
Enter the number of employees   5
Enter 5 employee id(s)
1200
2563
2934
4598
6355
Enter the id to search  4598
employee with id 4598 is found at location 4.
```

Program 2: Application of Arrays

1 Problem Statement

Polynomials are used to graph real world curves. For example, roller coaster designers may use polynomials to describe the curves in their rides. Combinations of polynomial functions are sometimes used in economics to do cost analysis. Design and develop a C program to add two polynomials of any degree using single dimensional array.

2 Student Learning Outcomes

After successful execution of this program, the student shall able to

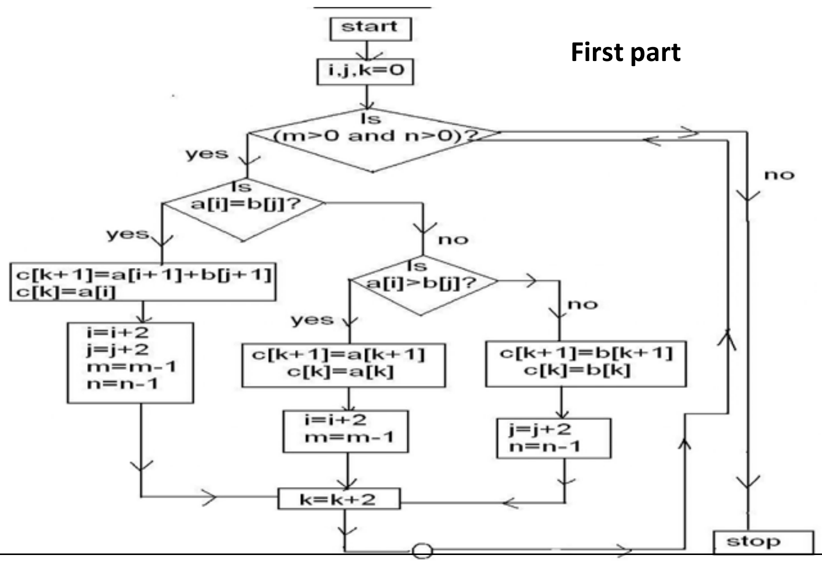
- Impart the knowledge of array in solving mathematical problems.
- Describe different applications of array.
- Analyze and create solutions for operations on arrays.

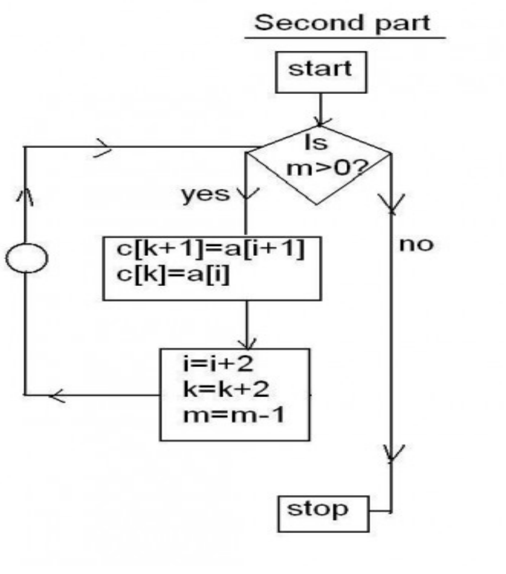
3 Design and development of the Program

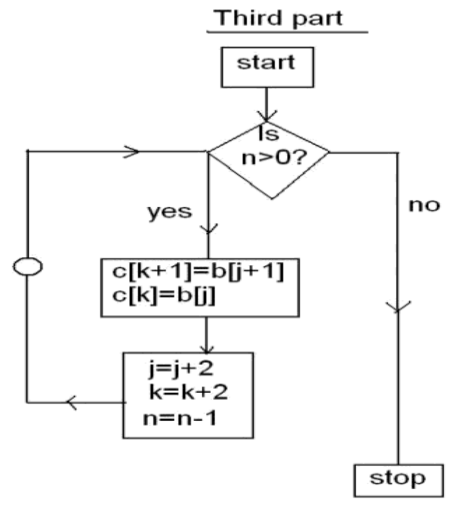
3.1 Theory

Polynomial Addition: A polynomial **f(x)** is a mathematical expression in the form $\mathbf{a_nx^n + a_{n-1}x^{n-1} + ... + a_0}$. The highest exponent of *x* is the **degree** of the polynomial. For example, the degreeof $\mathbf{x^5 + 3x^3 + 4}$ is **5**.
In a polynomial, a_n, a_{n-1}, ... , a_0 are called **coefficients**. If in a polynomial, the coefficients a_n, a_{n-1}, ... , a_1 are all 0, or in other words, the polynomial is in the form of a_0, we call this polynomial a **constant**. We can add, subtract polynomials by combine the terms in the polynomials with the same powers. For example:
Polynomial addition: $(x^5 + 3x^3 + 4)+(6x^6 + 4x^3) = 6x^6 + x^5 + 7x^3 + 4$

3.2 Flow Chart







Explanation of the Flow Chart:

```
1)  Declare array "a" for storing degree and coefficients of polynomial "first"
2)  Declare array "b" for storing degree and coefficients of polynomial "second"
3)  Declare array "c" for storing degree and coefficients of polynomial "sum"
4)  Declare variable "m" number of terms in first polynomial; "n"number of terms in second
    polynomial
5)  Variables i, j ,k are counters for array a, array b, array c and all initialized to 0;
6)  While ( m>0 and n>0)      {number of terms in both polynomials is >0}
7)      IF(a[i]=b[j])              {degree of first terms in both polynomials are same }
8)          c[k+1]=a[i+1]+b[i+1]    {add coefficients of first terms of both polynomials}
9)          c[k]=a[i]                  {degree of first term of "c" as degree of first term of "a"}
10)         i+=2                     {move to next term's degree in first poly "a"}
11)         j+=2                     {move to next term's degree in second poly "b"}
12)         m- -                     {decrease number of terms to be added in "a"}
13)         n- -                     {decrease number of terms to be added in "b"}
14)     END OF IF
15)     ELSE IF(a[i]>b[j])  {degree of first terms of "a" > degree of first terms of "b"}
16)         c[k+1]=a[i+1]  {coefficient of first terms of "a" as coefficient of first term of "c"}
17)         c[k]=a[i]         {degree of first term of "c" will be degree of first term of "a"}
18)         i+=2              {move to next term's degree in first poly "a"}
19)         m- -              {decrease number of terms to be added in "a"}
20)    END OF ELSE IF
21)    ELSE(a[i]<b[j])  {degree of first terms of 1st  poly < degree of first terms of 2nd  poly}
22)        c[k+1]=b[j+1] { coefficient of first terms of "b" as coefficient of first term of "c"}
23)        c[k]=b[j]           {degree of first term of "c" will be degree of first term of "b"}
24)        j+=2               {move to next term's degree in first poly "b"}
```

```
25)       n- -              {decrease number of terms to be added in "b"}
26)    END OF ELSE
27)    k+=2                 {move to next term of poly "c"}
28) END OF WHILE
29) IF(m>0)                 {only first polynomial "a" is present}
30)    c[k+1]=a[i+1]  { coefficient of first terms of "a" as coefficient of first term of "c"}
31)    c[k]=a[i]            {degree of first term of "c" will be degree of first term of "a"}
32)    i+=2                 {move to next term's degree in first poly "a"}
33)    k+=2                 {move to next term of poly "c"}
34)    m - -                {decrease number of terms to be added in "a"}
35) END IF

36) IF(n>0)                 {only second polynomial "b" is present}
37)   c[k+1]=b[j+1]    { coefficient of first terms of "b" as coefficient of first term of "c"}
38)   c[k]=b[j]             {degree of first term of "c" will be degree of first term of "b"}
39)   j+=2                  {move to next term's degree in first poly "b"}
40)   k+=2                  {move to next term of poly "c"}
41)   n - -                 {move to next term of poly "c"}
42) END IF
```

3.3	**Coding using C Language**

```c
#include<stdio.h>
main()
{
        int a[10], b[10], c[10],m,n,k,k1,i,j,x;       /*Declaration*/
        printf("\n\tPolynomial Addition\n");
        printf("\t===================\n");
        printf("\n\tEnter the no. of terms of the polynomial:");
        scanf("%d", &m);            /*Read the number of terms in first polynomial "a"*/
       printf("\n\tEnter the degrees and coefficients:");
       for (i=0;i<2*m;i++)
           scanf("%d", &a[i]);          /*Read the degrees and coefficients of "a"*/
       /*Print the first polynomial of the above given degrees and co-efficients*/
       printf("\n\tFirst polynomial is:");
       k1=0;
       if(a[k1+1]==1)
           printf("x^%d", a[k1]);
       else
           printf("%dx^%d", a[k1+1],a[k1]);
       k1+=2;
```

```
    while (k1<i)
     {
           printf("+%dx^%d", a[k1+1],a[k1]);
           k1+=2;
     }
    printf("\n\n\n\tEnter the no. of terms of 2nd polynomial:");
    scanf("%d", &n);            /*Read the number of terms in second polynomial "b"*/

    printf("\n\tEnter the degrees and co-efficients:");
    for(j=0;j<2*n;j++)
     scanf("%d", &b[j]);        /*Read the degrees and coefficients of "a"*/
     /*Print the second polynomial of the above given degrees and co-efficients*/
    printf("\n\tSecond polynomial is:");
    k1=0;
   if(b[k1+1]==1)
          printf("x^%d", b[k1]);
   else
          printf("%dx^%d",b[k1+1],b[k1]);
   k1+=2;
   while (k1<2*n)
   {
         printf("+%dx^%d", b[k1+1],b[k1]);
         k1+=2;
   }
    i=0;
    j=0;
    k=0;
   while (m>0 && n>0)                         /*number of terms in "a" and "b" >0*/
   {
         if (a[i]==b[j])                /*degree of ith term of "a" and "b" are equal*/
         {
              c[k+1]=a[i+1]+b[j+1];
              c[k]=a[i];
              m--;
              n--;
              i+=2;
              j+=2;
         }
        else if (a[i]>b[j])
        {
             c[k+1]=a[i+1];
```

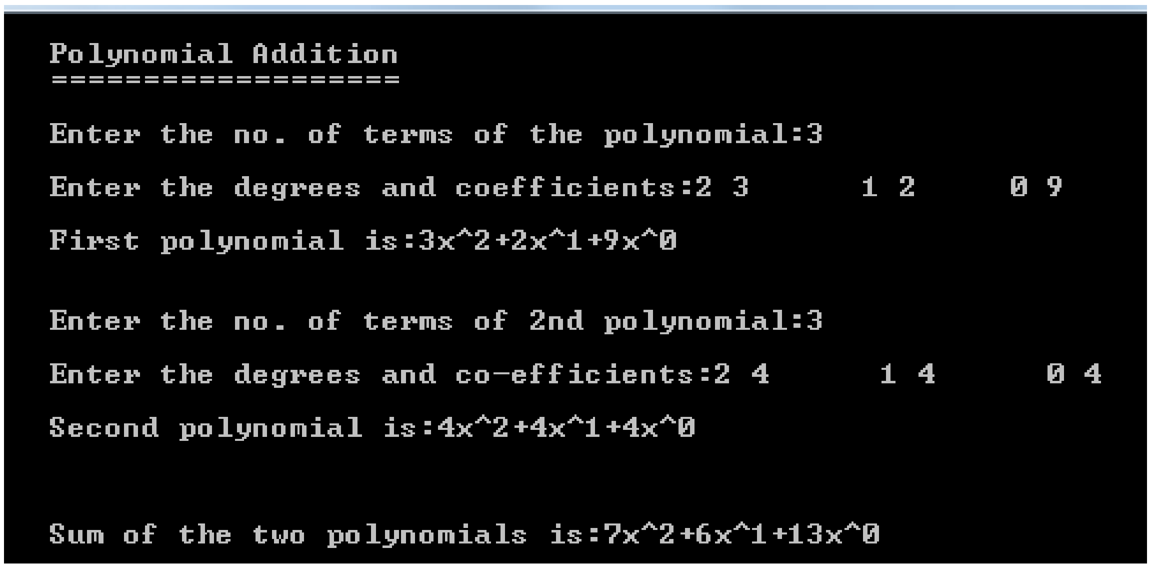
```
                c[k]=a[i];
                m--;
                i+=2;
            }
           else
            {
                c[k+1]=b[j+1];
                c[k]=b[j];

               n--;
               j+=2;
           }
           k+=2;
       }
      while (m>0)                                  /*only "a" is present*/
      {
            c[k+1]=a[i+1];
            c[k]=a[i];
            k+=2;
            i+=2;
            m--;
       }
       while (n>0)                                  /*only "b" is present*/
       {
            c[k+1]=b[j+1];
            c[k]=b[j];
             k+=2;
            j+=2;
           n--;
        }
      /*Print the sum of two polynomials*/
      printf("\n\n\n\n\tSum of the two polynomials is:");
        k1=0;
        if (c[k1+1]==1)
                printf("x^%d", c[k1]);
        else
                printf("%dx^%d", c[k1+1],c[k1]);
        k1+=2;
        while (k1<k)
        {
                if (c[k1+1]==1)
```

```
97.            printf("+x^%d", c[k1]);
98.      else
99.            printf("+%dx^%d", c[k1+1], c[k1]);
100.               k1+=2;
101.           }
102.           }
```

Explanation of the program:

- Lines 8 and 9 read the number of terms in polynomial “first”
- Lines 10 to 12 read the degrees and coefficients of polynomial “first”
- Lines 14 to 25 print polynomial “first”
- Lines 26 and 27 read the number of terms in polynomial “second”
- Lines 28 to 31 read the degrees and coefficients of polynomial “second”
- Lines 32 to 43 print the polynomial “second”
- Lines 44 to 89 calculate the sum of polynomial “first” and polynomial “second”
- Line 90 to 104 prints the sum of polynomial “first” and polynomial “second”

3.4 **Expected Results**



```
Polynomial Addition
===================

Enter the no. of terms of the polynomial:3

Enter the degrees and coefficients:2 3      1 2     0 9

First polynomial is:3x^2+2x^1+9x^0

Enter the no. of terms of 2nd polynomial:3

Enter the degrees and co-efficients:2 4      1 4      0 4

Second polynomial is:4x^2+4x^1+4x^0


Sum of the two polynomials is:7x^2+6x^1+13x^0
```

Program 3: Structures

1 Problem Statement

Real-life quantities that are naturally described by complex numbers rather than real numbers. Design and develop a C program to add and subtract 'n' complex numbers using structure.

2 Student Learning Outcomes

After successful execution of this program, the student shall able to

- Implement structure.
- Impart the knowledge of data structures in mathematical problems and solutions.
- Create databases using structure.
- Implement other data structures like stack, linked list, tree etc using structure.

3 Design and development of the Program

3.1 Theory

Structure: Structure is a user defined data type, which allows combining data items of different types. To define a structure, the **struct** statement is used. The struct statement defines a new datatype, with more than one member for a program. The format of the struct statement is:

```
struct [structure tag]
{
        member definition;
        member definition;
        ...
        member definition;
} [one or more structure variables];
```

3.2 Algorithm

Algorithm to add and subtract two complex numbers using strcuture:

```
Step 1: Declare structure "complex" with variables "realp and imagp".
Step 2: Declare array of complex numbers 'comp'
Step 3: Read realp and imagp for complex numbers
Step 4: FOR i←0 to i<n Repeat Step 5          where n=Count of complex numbers
Step 5:      Read realp and imagp for n complex numbers.
Step 6: FOR i←0 to i<n Repeat Step 7
Step 7:      Print n complex numbers.
Step 8: Set sub_c.realp←0, sub_c.imagp←0
Step 9: FOR i←0 to i<n Repeat Step 10 and 11
Step 10:     sum_c.realp←sum_c.realp+comp[i].realp
Step 11:     sum_c.imagp←sum_c.imagp+comp[i].imagp
Step 12: Set sub_c.realp←comp[0].realp , sub_c.imagp←comp[0].imagp
Step 13: FOR i←0 to i<n Repeat Step 14 and 15
Step 14:     sub_c.realp←sub_c.realp-comp[i].realp
Step 15:     sub_c.imagp←sub_c.imagp-comp[i].imagp
Step 16: Print sum_c and sub_c
Step 17: Exit
```

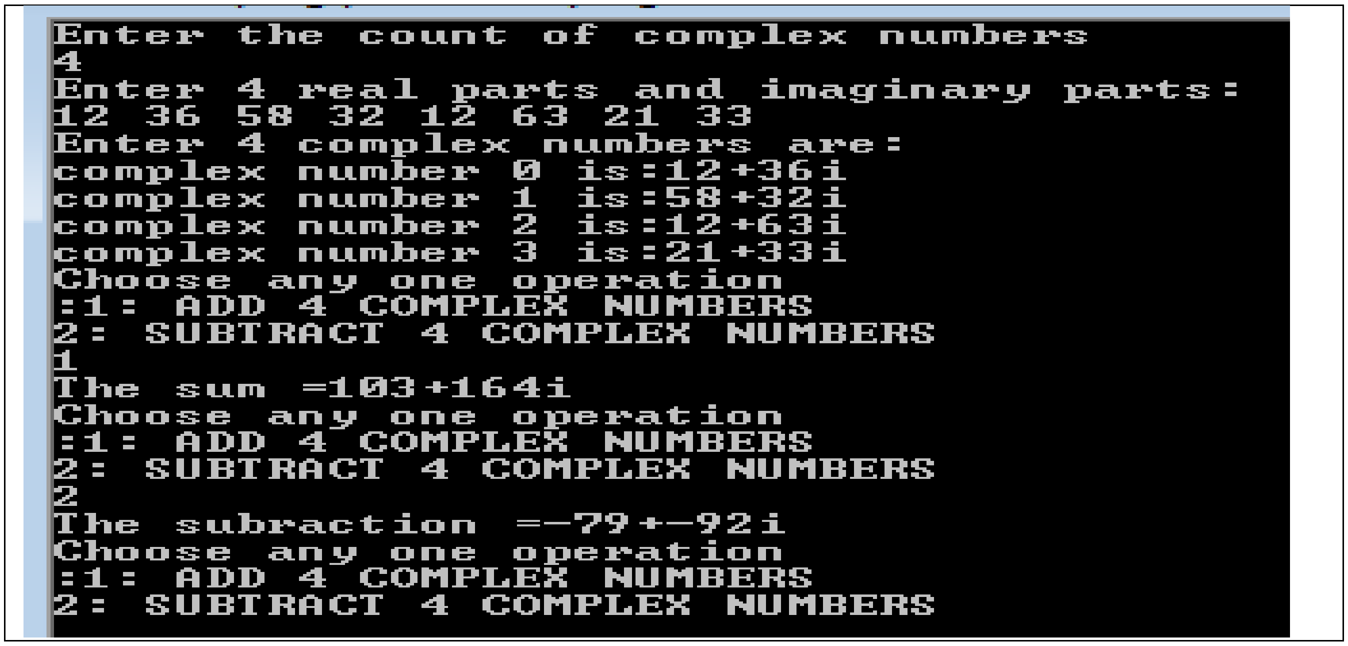
3.3 Coding using C Language

```c
#include <stdio.h>
struct complex                          /*Declare an structure with tagname ‘complex’*/
{
     int realp;                         /*Member declaration*/
     int imagp;
}comp[50],sum_c,sub_c;                  /*Declare array of complex numbers*/
main()
{
    int choice,i,n;
    printf("Enter the count of complex numbers\n");
    scanf("%d",&n);

    printf("Enter %d real parts and imaginary parts:\n",n);
    for(i=0;i<n;i++)
        scanf("%d%d",&comp[i].realp,&comp[i].imagp);

    printf("Enter %d complex numbers are:\n",n);
    for(i=0;i<n;i++)
        printf("complex number %d is:%d+%di\n",i,comp[i].realp,comp[i].imagp);

    while(1)
    {
          printf("Choose any one operation\n:");
          printf("1: ADD %d COMPLEX NUMBERS\n",n);
          printf("2: SUBTRACT %d COMPLEX NUMBERS\n",n);
          printf("3: Quit\n”);
          scanf("%d",&choice);

          switch(choice)
          {
                 case 1: sum_c.realp=0;
                         sum_c.imagp=0;
28.                        for(i=0;i<n;i++)
29.                        {
30.                            sum_c.realp=sum_c.realp+comp[i].realp;
31.                            sum_c.imagp=sum_c.imagp+comp[i].imagp;
32.                        }
33.                        printf("The sum =%d+%di\n",sum_c.realp,sum_c.imagp);
34.                        break;
35.                case 2: sub_c.realp=comp[0].realp;
36.                        sub_c.imagp=comp[0].imagp;
37.                        for(i=1;i<n;i++)
38.                        {
39.                            sub_c.realp=sub_c.realp-comp[i].realp;
40.                            sub_c.imagp=sub_c.imagp-comp[i].imagp;
41.                        }
42.                        printf("The subraction =%d+%di\n",sub_c.realp,sub_c.imagp);
43.                        break;
44.
45.                case 3: return 0;
46.
47.                default: printf("Invalid Input\n");
45.           }
46.    }
47. }
```

Explanation of the program

- Lines 2 to 6 declare a structure and declaration of structure type array and variables.
- Lines 10 and 11 read the count of complex numbers
- Lines 12 to 14 read the complex numbers
- Lines 26 to 34 add complex numbers
- Line 35 to 43 subtract complex numbers

3.4	**Expected Results**



```
Enter the count of complex numbers
4
Enter 4 real parts and imaginary parts:
12 36 58 32 12 63 21 33
Enter 4 complex numbers are:
complex number 0 is:12+36i
complex number 1 is:58+32i
complex number 2 is:12+63i
complex number 3 is:21+33i
Choose any one operation
:1: ADD 4 COMPLEX NUMBERS
2: SUBTRACT 4 COMPLEX NUMBERS
1
The sum =103+164i
Choose any one operation
:1: ADD 4 COMPLEX NUMBERS
2: SUBTRACT 4 COMPLEX NUMBERS
2
The subraction =-79+-92i
Choose any one operation
:1: ADD 4 COMPLEX NUMBERS
2: SUBTRACT 4 COMPLEX NUMBERS
```

Program 4: Stack

1 Problem Statement

Reversal of string is used to check for palindrome or not. Design and Develop a C program to reverse a string using stack.

2 Student Learning Outcomes

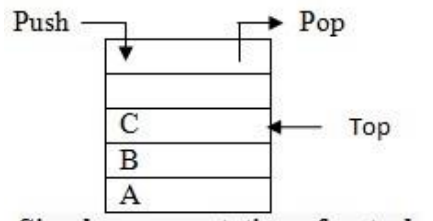
After successful execution of this program, the student shall able to

- study the working of stack
- implement the stack operations
- apply the knowledge of stack in different applications

3 Design and development of the Program

3.1 Theory

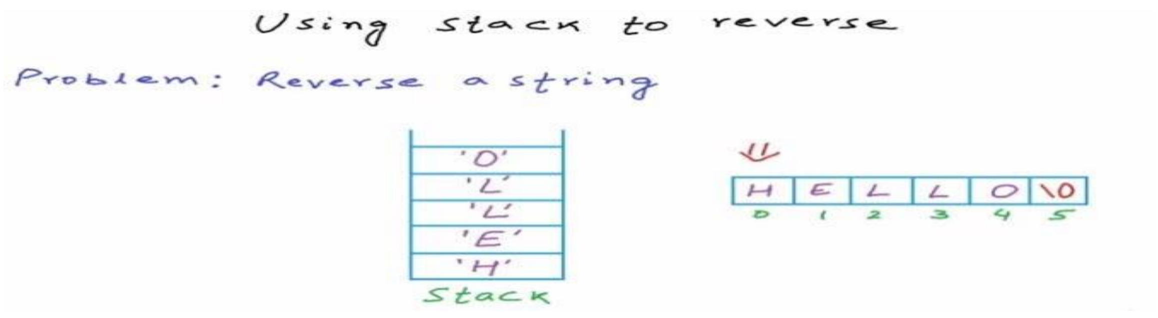
STACK: A Stack or LIFO (last in, first out) is an abstract data type that serves as a collection of elements, with two principal operations: **push**, which adds an element to the collection, and **pop**, which removes the last element that was added.



Simple representation of a stack

It is a linear data structure, or more abstractly a sequential collection in which the push and pop operations occur only at one end of the structure, referred to as the top of the stack. Additionally, a peek operation may give access to the top. Stack can be implemented either using array or linked list.

Reversal of string:



3.2 Algorithm

```
Step 1: Create an empty stack.
Step 2: Read String 'str'
Step 3: FOR i←0 to i<Length of string, Repeat Perform Step 4 to 7 to Push characters of string to stack.
Step 4: IF top = STACK_SIZE-1
Step 5:     Print 'stack overflow'
Step 6: ELSE
Step 7:     Set stack[++top]←str[i]
Step 8: FOR i←0 to i<Length of string, Repeat Step 9 to 13 to Pop all characters from stack and put them back to string.
Step 9: IF top = -1
Step 11:     Print stack underflow
Step 12: ELSE
Step 13: Set str[i]← stack[top--]
Step 14: Print string str
Step 15: Exit
```

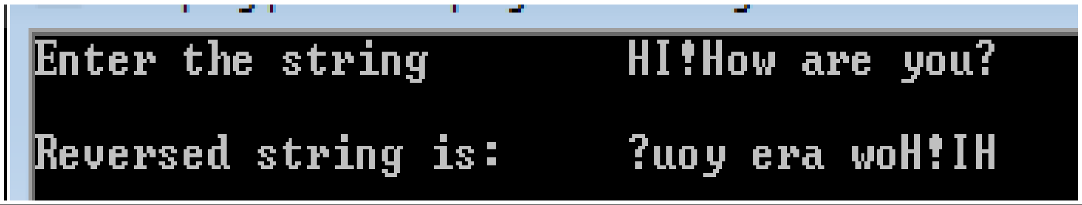
3.3 Coding using C Language

```c
#include<stdio.h>
#include<string.h>
#define MAX 20

int top = -1;
char stack[MAX];                                  /*Declaration*/
char pop();                                       /*Function Declaration*/
void push(char);                                  /*Function Declaration*/

main()
{
        char str[20];                             /*array declaration for string*/
        int i;
        printf("Enter the string\t");
        gets(str);                                /*Read the string*/

        /*Push characters of the string str on the stack */
        for(i=0;i<strlen(str);i++)
                push(str[i]);
        /*Pop characters from the stack and store in string str */
16.     for(i=0;i<strlen(str);i++)
17.             str[i]=pop();

18.     printf("\nReversed string is:\t");
19.     puts(str);
20. }                                           /*End of main()*/
    /*push function definition*/
21. void push(char item)
22. {
23.     if(top == (MAX-1))
24.     {
25.             printf("Stack Overflow\n");
26.             return;
27.     }
28.     stack[++top] =item;
29. }                                           /*End of push()*/
    /*pop function definition*/
30. char pop()
31. {
32.     if(top == -1)
33.     {
34.             printf("Stack Underflow\n");
35.             return;
36.     }
37.     return stack[top--];
38. }                                           /*End of pop()*/
```

Explanation of the program

- Lines 4 and 5 declare variable
- Lines 6 and 7 declare push and pop functions
- Lines 12 and 13 read the string
- Lines 14 and 15 call push function
- Lines 16 and 17 call pop function
- Lines 18 and 19 print reversed string
- Lines 21 to 29 define function push
- Lines 30 to 38 define function pop

3.4	**Expected Result**



```
Enter the string         HI!How are you?

Reversed string is:      ?uoy era woH!IH
```

Program 5: Application of Stack

1	**Problem Statement**

The compilers always convert infix expression into postfix to perform further operations like parsing, lexical analysis etc. Design and Develop a C program to convert an infix expression into postfix using stack.

2	**Student Learning Outcomes**

After successful execution of this program, the student shall able to

- identify the various application where the stack concepts are used
- differentiate between infix/postfix expressions.
- implement the expression conversion using stack operations

3	**Design and development of the Program**
3.1	**Theory**

Application of Stack:

- Parsing
- Recursive functions
- Expression Evaluation (Postfix evaluation)
- Expression conversion
    - ✓ Infix to prefix
    - ✓ Infix to postfix
    - ✓ Postfix to infix
    - ✓ Prefix to infix
- Tower of Hanoi

Conversion of infix to postfix expression:

Expression Representation:

Infix	**Prefix**	**Postfix**	**Infix**
a+b	+ab	ab+	a+b
a+b*c	+a*bc	abc*+	a+b*c
(a+b)*(c-d)	*+ab-cd	ab+cd-*	(a+b)*(c-d)

There is an algorithm to convert an infix expression into a postfix expression. It uses a stack; but in this case, 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 a storage structure, since no operator can be printed until both of its operands have appeared.
In this algorithm, all operands are printed (or sent to output) when they are read. There are more complicated rules to handle operators and parentheses. Example: A * B + C becomes A B * C +
The order in which the operators appear is not reversed. When the '+' is read, it has lower precedence than the '*', so the '*' must be printed first.
The first column will show the symbol currently being read. The second will show what is on the stack and the third will show the current contents of the postfix string. The stack will be written from left to right with the 'bottom' of the stack to the left.

S.No.	current symbol	operator stack	postfix string
1	A		A
2	*	*	A
3	B	*	AB
4	+	+	AB*
5	C	+	AB*C
6			AB*C+

3.2 Algorithm

Input: Infix expression
Variables used:s->stack,ch->token read, element->character used to store the element poped
Output: Equivalent Postfix Expression

```
Step1:   Stack S
Step2:  Char  ch
Step3:  Char  element
Step4: WHILE(Tokens are Available), Repeat from Step5 to Step12
Step5:    ch = Read(Token);
Step6:    IF(ch is Operand)
Step7:            Print ch ;
```

```
Step 8: End IF
Step 9:  ELSE
Step10:         while(Priority(ch) <= Priority(Top Most Stack)), Repeat Step 10 to 11
Step11:                 element = Pop(S);
Step12:                 Print(ele);
Step13:         Push(S,ch);
Step 14: End ELSE
Step15: WHILE(!Empty(S)), Repeat Step 16 and 17
Step16:         element = Pop(S);
Step17:         Print(ele);
Step 18:END WHILE
Step 19: END
```

3.3 Coding using C Language

```c
#define SIZE 50                                   /* Size of Stack */
#include<stdio.h>
#include<string.h>
#include <ctype.h>
char s[SIZE];
int top=-1;                                        /* Global declarations */

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

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

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

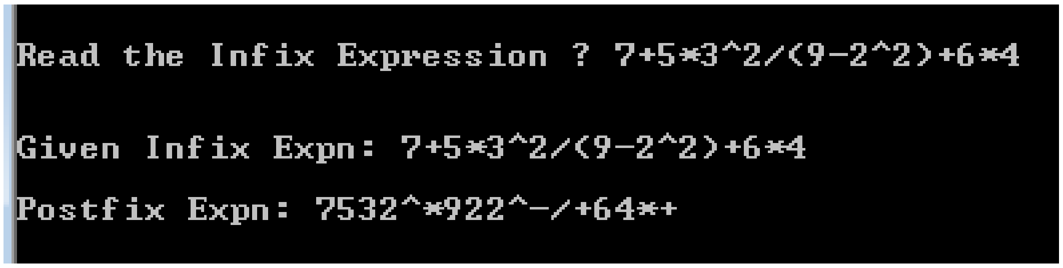
```
25.         case '/': return 3;
26.         case '$':
27.         case '^': return 4;
28.     }
29. }

30. main()
31. {
32.     char infx[50],pofx[50],ch,elem;
33.     int i=0,k=0;
34.     printf("\n\nRead the Infix Expression ? ");
35.     scanf("%s",infx);
36.     push('#');
37.     while( (ch=infx[i++]) != '\0')
38.     {
39.        if( ch == '(' )                          /* For left parentheses */
40.                 push(ch);
41.        else if(isalnum(ch))                     /* For operand */
42.                 pofx[k++]=ch;
43.        else if( ch == ')')                      /* For Right parentheses */
44.        {
45.                 while( s[top] != '(' )
46.                 pofx[k++]=pop();
47.                 elem=pop();                      /* Remove ( */
48.        }
49.        else
50.        {                                        /* For Operator */
51.                 while( pr(s[top]) >= pr(ch) )
52.                 pofx[k++]=pop();
53.                 push(ch);
54.        }
55.     }
56.     while( s[top] != '#')                      /* Pop from stack till empty */
57.        pofx[k++]=pop();
58.     pofx[k]='\0';                              /* Make pofx as valid string */
59.     printf("\n\nGiven Infix Expn: %s  Postfix Expn: %s\n",infx,pofx);
60.     }
```

Explanation of the program

- Lines 7 to 29 define the functions used in program

- Lines 34 to 35 read infix expression
- Lines 36 to 57 perform conversion algorithm
- Line 59 prints the infix and postfix expressions

3.4	**Expected Result**



```
Read the Infix Expression ? 7+5*3^2/(9-2^2)+6*4

Given Infix Expn: 7+5*3^2/(9-2^2)+6*4

Postfix Expn: 7532^*922^-/+64*+
```

Program 6: Application of Stack

1	**Problem Statement**

Evaluation of postfix expressions is done by compilers during the compilation process. Design and Develop a C program to evaluate a postfix expression using stack.

2	**Student Learning Outcomes**

After successful execution of this program, the student shall able to

- identify the various application where the stack concepts are used
- differentiate between infix/postfix expressions.
- implement the postfix expression evaluation using stack

3	**Design and development of the Program**
3.1	**Theory**

Evaluation of postfix Expression:
Postfix notation (also known as Reverse Polish Notation or RPN in short) is a mathematical notation in which operators follow all of its operands. It is different from infix notation in which operators are placed between its operands. Consider the following example: **234+*6-**
The exact steps of the algorithm are put in the table below:

Input token	**Operation**	**Stack contents (TOP=right)**	**Details**
2	Push on the stack	2	
3	Push on the stack	2, 3	
4	Push on the stack	2, 3, 4	
+	Add	2, 7	Pop two values: 3 and 4 and push the result 7 on the stack
*	Multiply	14	Pop two values: 2 and 7 and push the result 14 on the stack
6	Push on the stack	14, 6	
–	Subtract	8	Pop two values: 14 and 6 and push the result 8 on the stack
(End of tokens)	(Return the result)	8	Pop the only value 8 and return it

3.2 Algorithm

Input: A postfix expression
Variables used: op1->operator1, op2->operator2, token-> get next token
Output : Result of the postfix expression

```
Step1: create a new stack
Step2: WHILE(input stream is not empty), Repeat from Step3 to Step10
Step3:      token = getNextToken()
Step4:      IF(token instanceof operand)
Step5:              push(token);
Step6:      ELSE IF (token instance of operator)
Step7:              op2 = pop()
Step8:              op1 = pop()
Step9:              result = calc(token, op1, op2);
Step10:             push(result)
            end ELSE
Step11: pop(result)
```

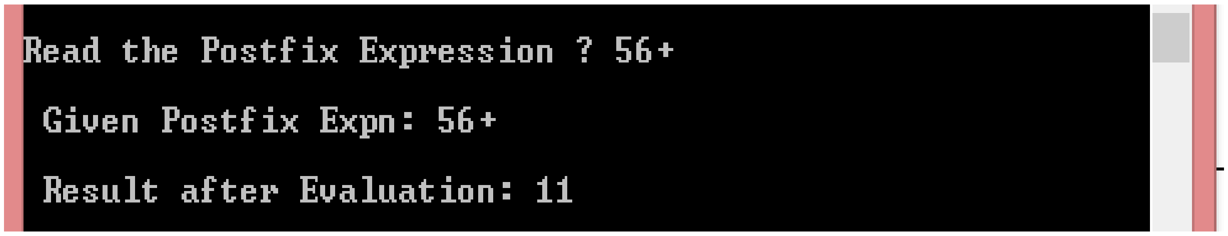
3.3 Coding using C Language

```c
#define SIZE 50                              /* Size of Stack */
#include<stdio.h>
#include <ctype.h>
int s[SIZE];
int top= -1;                                 /* Global declarations */
void push(int elem)                          /* Function FOR PUSH operation */
{
        s[++top]=elem;
}

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

main()
{
        char pofx[50],ch;
        int i=0,op1,op2;
18.        printf("\n\nRead the Postfix Expression ? ");
19.        scanf("%s",pofx);
20.        while( (ch=pofx[i++]) != '\0')
21.        {
22.                if(isdigit(ch))
23.                        push(ch-'0');                    /* Push the operand */
24.                else
25.                {       /*pop two  operands */
26.                        op2=pop();
27.                        op1=pop();
28.                        switch(ch)
29.                        {
30.                                case '+': push(op1+op2);
31.                                           break;
32.                                case '-':  push(op1-op2);
33.                                           break;
34.                                case '*': push(op1*op2);
35.                                           break;
36.                                case '/':  push(op1/op2);
37.                                           break;
38.                        }
39.                  }
40.           }
41.           printf("\n Given Postfix Expn: %s\n",pofx);
42.           printf("\n Result after Evaluation: %d\n",s[top]);
43. }
```

Explanation of the program

- Lines 6 to 9 define the function "push"
- Lines 10 to 13 define the function "pop"
- Lines 18 and 19 read the postfix expression
- Lines 20 to 40 evaluate postfix expression
- Lines 36-37 print the result

Expected Result



```
Read the Postfix Expression ? 56+

 Given Postfix Expn: 56+

 Result after Evaluation: 11
```

Program 7: Queue

1 Problem Statement

When jobs are entered in system they are stored in the job queue and whenever the processor is free the jobs are moved to ready queue for execution. Consider a college is having a printer. Design and develop a C program to implement job scheduling in the printer.
Hint: Store the jobs based on their job id in queue.

2 Student Learning Outcomes

After successful execution of this program, the student shall able to

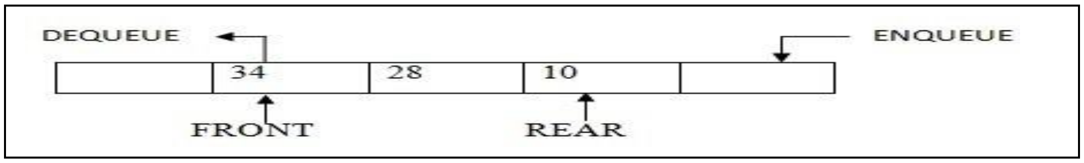
- implement simple queue
- apply various operation performed on queue in problem solving

3 Design and development of the Program

3.1 Theory

QUEUE: Queue is a data structure where elements are inserted from one end and deleted from the other end. The end from elements are inserted is called rear end and the end from which elements are deleted is called front end. Since the first element inserted is the first element to be deleted, queue is also called First In First Out data structure.
The various operations that can be implemented in queue are:

1. Enqueue (Insert at rear)
2. Dequeue (Delete from front)
3. Display



3.2 Algorithm

Input: The option whether the element is to be inserted , deleted or displayed
Variables used: q-> array for queue; front, rear-> variables to show state of queue; MAX-> maximum size of queue; n->option for operation; item->item to be enqueued or dequeued
Output: The Output of the operation performed on queue
Algorithm INSERT_REAR

```
Step 1: IF  rear=MAX-1                    where MAX=Size of the Queue
Step 2:     write queue overflow
Step 3: END IF
Step 4: ELSE
Step 5:     rear=rear+1
Step 6:     q[rear]=item                  where item=item to be inserted
Step 7: END ELSE
```

Algorithm DELETE_FRONT

```
Step 1: IF(front>rear)
Step 2:     write queue underflow
Step 3: END IF
Step 4: ELSE
Step 5:   write  q[front]
Step 6:   front=front+1
Step 7: END ELSE
```

Algorithm DISPLAY

```
Step 1: IF(front>rear)
Step 2:    write queue empty
Step 3: END IF
Step 4: ELSE
Step 5:    FOR i= rear to front-1
Step 6:        write q[i]
Step 7:    END FOR
Step 8: END ELSE
```

3.3 Coding using C Language

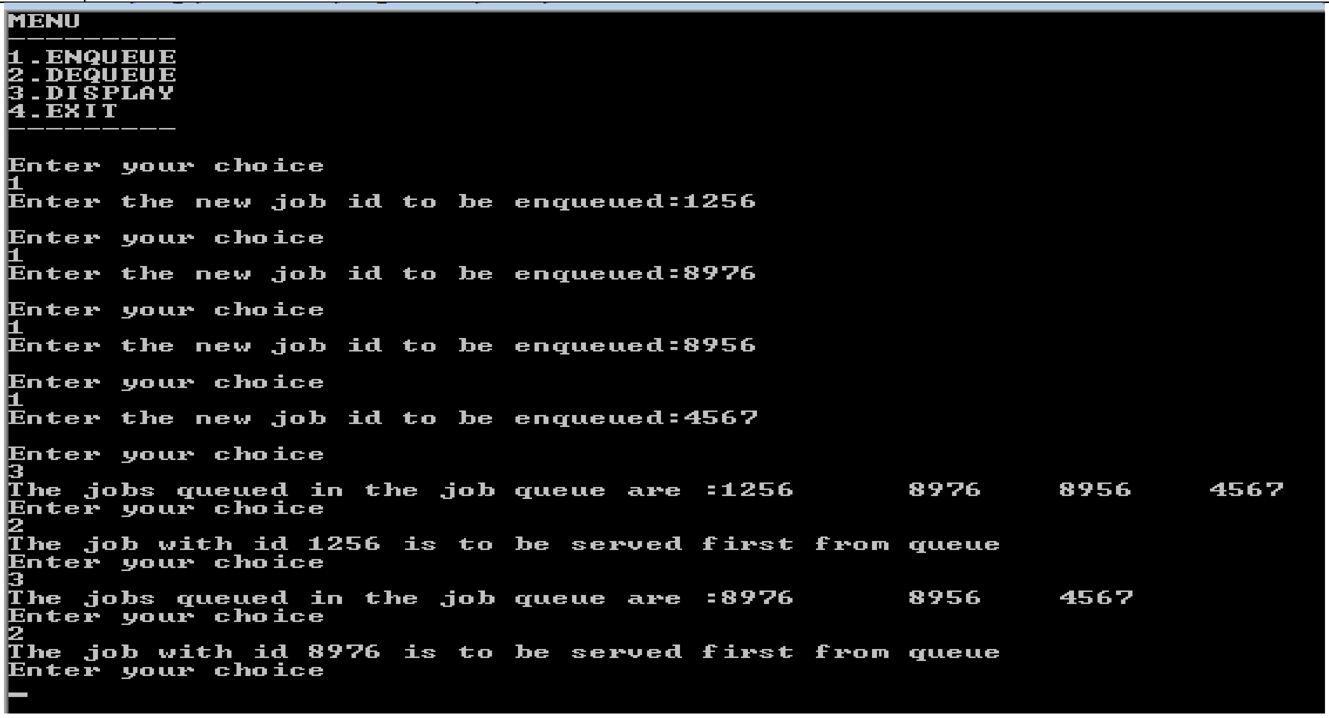
```
#include<stdio.h>
#define MAX 10
void insert(int item);
void delete();
void display();
int jobqueue[MAX],front= -1,rear= -1;
main()
{
        int n,job_id;
        printf("MENU\n------\n1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.EXIT\n ----- \n");
        while(1)
         {
                printf("\nEnter your choice\n");
                scanf("%d",&n);
                switch(n)
                {
                        /* case to insert a job into queue*/
                        case 1: printf("Enter the new job id to be enqueued:");
                                scanf("%d",&job_id);
                                insert(job_id);                 /*Function call to insert*/
                                break;
                        /*case to delete an element from the queue*/
                case 2: delete();
                        break;
                /*case to display the jobs waiting in queue*/
                case 3: display();
                        break;
                /*case to exit from loop*/
                case 4:
                        break;
                default:
                        printf("Invalid choice\n");
                        break;
        }
    }
}
void insert(int item)
{
   if(rear<MAX-1)             /*Queue is not full*/
   {
       if(rear==-1)                         /*Queue is empty*/
       {
            front=0;                     /*increase front by 1*/
            rear=0;                      /*increase rear by 1*/
            jobqueue[rear]=item;             /*insert job at rear*/
       }
       else
            jobqueue[++rear]=item;         /*increase rear by 1 and insert item*/
   }
   else
       printf("Job Queue Overflow\n");        /*Queue full*/
}/*End of insert()*/

void delete()
{
   if(front>=0)                         /*Queue is not empty*/
   {
        /*dequeue element from front*/
       printf("The job with id %d is to be served first from queue",jobqueue[front]);
       if(front==rear)            /*only one element in queue*/
       {
            front= -1;          /*no element left to be deleted*/
            rear= -1;           /*no element in queue*/
       }
       else
            front++;           /*increase front by 1*/
   }
   else
      printf("Job Queue Underflow! There are no jobs in queue\n");      /*Queue empty*/
```

```
64. }/*End of delete()*/

65. void display()
66. {
67.     if((front= = -1)&&(rear= = -1))          /*Queue empty*/
68.         printf("The queue is empty\n");
69.     else
70.     {
71.         int i;
72.         printf("The jobs queued in the job queue are :");
73.         for(i=front;i<=rear;i++)              /*traverse elements from front to rear*/
74.             printf("%d\t",jobqueue[i]);    /*print elements from queue*/
75.     }
76. }/*End of display()*/
```

Explanation of the program

- Lines 3 to 5 declare functions
- Line 6 declares the queue
- Lines 17 to 20 enqueue job with particular id by calling push function
- Lines 21 to 22 delete(process) job with particular id by calling pop function
- Lines 23 to 24 display the job queue
- Lines 33 to 48 define the function "insert"
- Lines 49-64 define the function "delete"
- Lines 65-76 define the function "display"

Expected Result



```
MENU
---------
1.ENQUEUE
2.DEQUEUE
3.DISPLAY
4.EXIT
---------

Enter your choice
1
Enter the new job id to be enqueued:1256

Enter your choice
1
Enter the new job id to be enqueued:8976

Enter your choice
1
Enter the new job id to be enqueued:8956

Enter your choice
1
Enter the new job id to be enqueued:4567

Enter your choice
3
The jobs queued in the job queue are :1256       8976     8956     4567
Enter your choice
2
The job with id 1256 is to be served first from queue
Enter your choice
3
The jobs queued in the job queue are :8976       8956     4567
Enter your choice
2
The job with id 8976 is to be served first from queue
Enter your choice
_
```

Program 8: Circular Queue

1 Problem Statement

In a machine gun bullets are always inserted in circular queue. Design and develop a C program to process bullets in machine gun.
Hint: Bullet insertion and firing (deletion) can be implemented using circular queue. Bullets are given with specific id.

2 Student Learning Outcomes

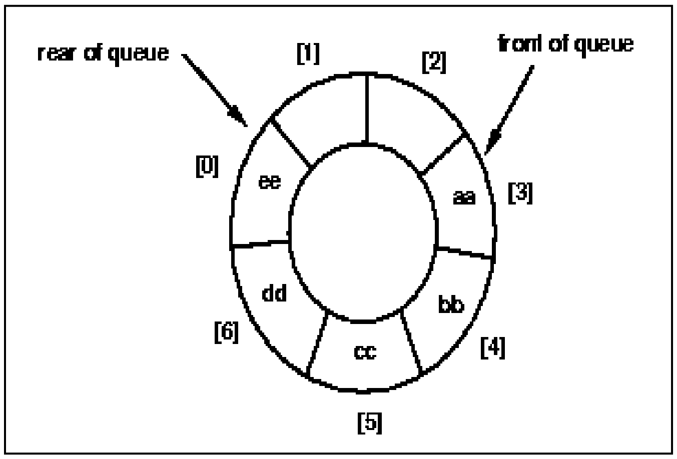
After successful execution of this program, the student shall able to

- Define circular queue
- Implement the operations performed on circular queue and apply the same for problem solving.

3 Design and development of the Program

3.1 Theory

In case of ordinary queue, rear insertion is not possible even if space is available at the front end. This is because before inserting an element in ordinary queue, it is tested whether rear is greater than the maximum size of the queue. If so, a message is printed stating queue size full and cannot be inserted. This disadvantage can be overcome using circular queue. **Circular queue** is a linear data structure. It follows First In First Out principle. In circular queue the last node is connected back to the first node to make a circle. Elements are added at the rear end and the elements are deleted at front end of the queue. Both the front and the rear pointers points to the beginning of the array. It is also called as "Ring buffer".



3.2 Algorithm

Input: Option whether the element is to be inserted, deleted or displayed
Variables used: cq->circular queue; front,rear->basic variables to show state of circular queue; MAXSIZE-> maximum size of Circular queue; num, item->item to be inserted or deleted
Output: Implementation of circular queue

Algorithm: Insert

```
Step 1: IF(front=(rear+1)%QUEUE_SIZE)
Step 2:     write queue overflow
Step 3: END IF
Step 4: ELSE
Step 5:     IF(front== -1)
Step 6:             front=rear=0;
Step 7:     END IF
Step 8: END ELSE
Step 9: ELSE
Step 10:    rear=(rear+1)%MAXSIZE;          where MAXSIZE=Size of the queue
Step 11:    cq[rear]=item;
Step 12: END ELSE
```

Algorithm: Delete

```
Step 1: IF(front == -1)
Step 2:     write CIRCULAR QUEUE IS UNDERFLOW
Step 3: END IF
Step 4: ELSE
Step 5:     item=cq[front];
Step 6:     cq[front]=0;
Step 7: END ELSE
Step 8: IF(front==rear)
Step 9:     front=rear=-1;
Step 10:END IF
Step 11:ELSE
Step 12:    front = (front+1)%MAXSIZE;
Step 13:    write DELETED ELEMENT FROM QUEUE IS :a
Step 14:END ELSE
```

3.3	**Coding using C Language**

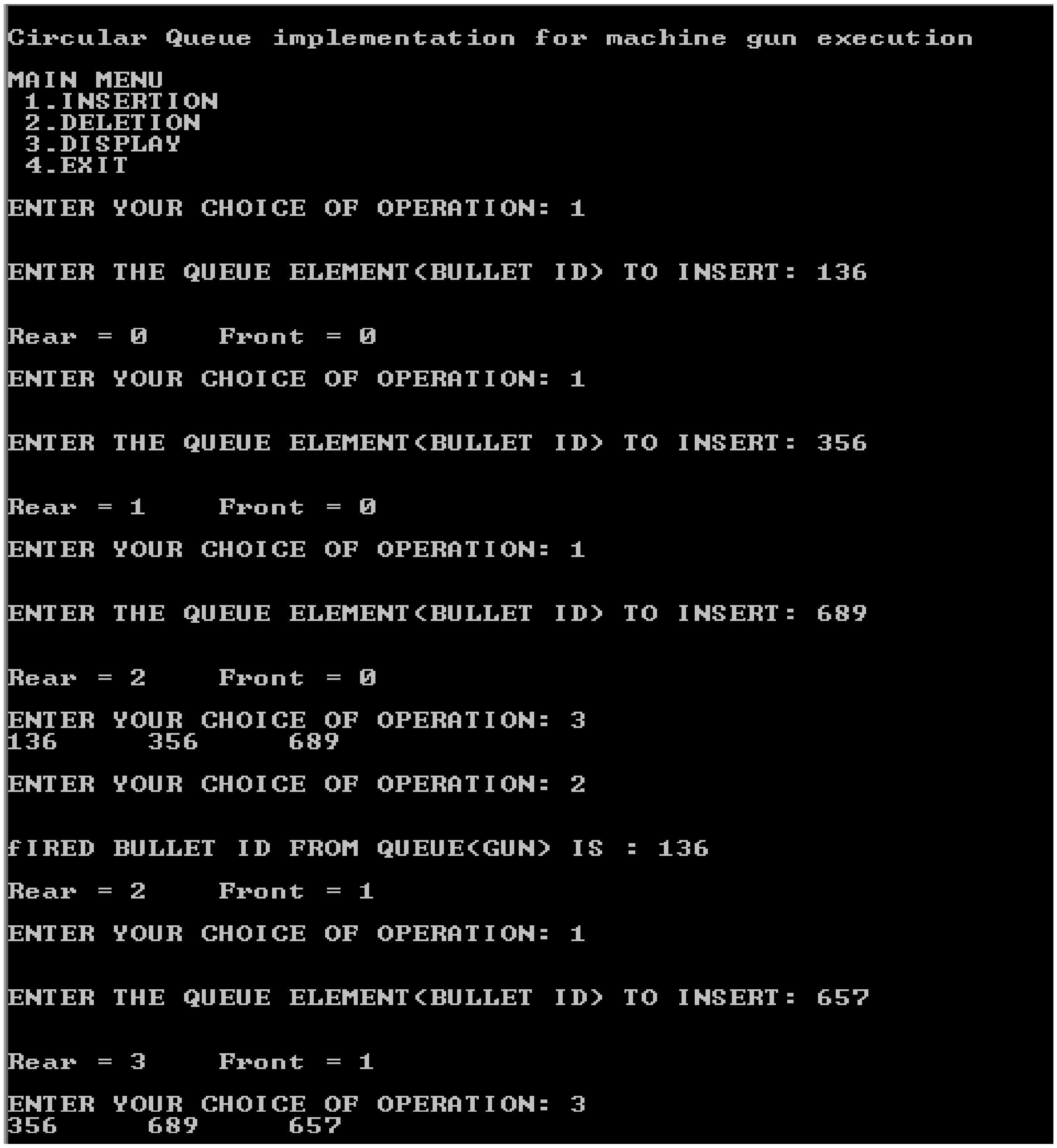
```
#include <stdio.h>
#include<stdlib.h>
#define MAXSIZE 5
int cq[MAXSIZE];              /*initialize circular queue*/
int front=-1,rear=-1;
void add(int);
void del();
void display();
int main()
{

      int ch,i,num;
      printf("\nCircular Queue implementation for machine gun execution");
printf("\n\nMAIN MENU\n 1.INSERTION\n 2.DELETION\n 3.DISPLAY\n
4.EXIT");
      while(1)
      {
            printf("\n\nENTER YOUR CHOICE OF OPERATION: ");
            scanf("%d",&ch);
            switch(ch)
            {
              case 1:
                    printf("\n\nENTER THE QUEUE ELEMENT(BULLET ID) TO INSERT:
");
                    scanf("%d",&num);           /*read element to be inserted*/
                    add(num);                   /*call function add*/
                    break;
             case 2:
                    del();                      /*call function del*/
                    break;
             case 3:
                   display();                   /*call function display*/
                   break;
             case 4:
                   exit(0);
            default: printf("\n\nInvalid Choice . ");
          }
      }
}
/*function to add an element into queue*/
void add(int item)
{
         if(front ==(rear+1)%MAXSIZE)            /*Queue is full*/
         {
              printf("\n\nCIRCULAR QUEUE(GUN)IS FULL");
```

```
        }
        else
          {
               if(front==-1)                          /*Queue empty*/
                       front=rear=0;              /*increase front and rear by 1*/
               else
             rear=(rear+1)%MAXSIZE;         /*move rear one place to the right*/
             cq[rear]=item;                                  /*insert element at rear*/
             printf("\n\nRear = %d    Front = %d ",rear,front);  /*print front and rear value */
          }
}
/*function to delete an element from the queue*/
void del()
{
         int a;
         if(front == -1)                                   /*Queue empty*/
               printf("\n\nCIRCULAR QUEUE(GUN) IS EMPLTY");
         else
         {
               a=cq[front];                         /*dequeue element from front and assign it to a*/
               cq[front]=0;                         /*assign 0 at front*/
               if(front==rear)                      /*only one element in queue*/
                     front=rear=-1;            /*move front one place to its left*/
               else
                     front = (front+1)%MAXSIZE;      /*move front one place to its right*/
               printf("\n\nfIRED BULLET ID FROM QUEUE(GUN) IS : %d ",a);  /*print deleted element*/
               printf("\n\nRear = %d     Front = %d ",rear,front);  /*print the values of front and rear*/
         }
}
void display()                        /* function to display the queue elements*/
{
        int i;
        if(front<rear)
        {
             for(i=front;i<=rear;i++)                 /*traverse queue elements from front to rear*/
                 printf("%d\t",cq[i]);       /*print elements of queue*/
        }
        else
        {
            for(i=0;i<MAXSIZE;i++)
                 printf("%d\t",cq[i]);
        }                                              /*elements of queue*/
}
```

Explanation of the program

- Line 4 and 5 declare circular queue and other variables.
- Lines 6 to 8 declare the prototype of insert, delete and display functions.
- Lines 24 to 31 call for insert, delete and display functions.
- Lines 39 to 54 define the function "Insert" to insert elements into the queue
- Lines 56 to 72 define the function "del" to delete elements from the queue
- Lines 74 to 91 define the function ""display" to display the elements of the queue

3.4	**Expected Results**



```
Circular Queue implementation for machine gun execution

MAIN MENU
 1.INSERTION
 2.DELETION
 3.DISPLAY
 4.EXIT

ENTER YOUR CHOICE OF OPERATION: 1


ENTER THE QUEUE ELEMENT<BULLET ID> TO INSERT: 136


Rear = 0    Front = 0

ENTER YOUR CHOICE OF OPERATION: 1


ENTER THE QUEUE ELEMENT<BULLET ID> TO INSERT: 356


Rear = 1    Front = 0

ENTER YOUR CHOICE OF OPERATION: 1


ENTER THE QUEUE ELEMENT<BULLET ID> TO INSERT: 689


Rear = 2    Front = 0

ENTER YOUR CHOICE OF OPERATION: 3
136     356     689

ENTER YOUR CHOICE OF OPERATION: 2


fIRED BULLET ID FROM QUEUE<GUN> IS : 136

Rear = 2    Front = 1

ENTER YOUR CHOICE OF OPERATION: 1


ENTER THE QUEUE ELEMENT<BULLET ID> TO INSERT: 657


Rear = 3    Front = 1

ENTER YOUR CHOICE OF OPERATION: 3
356     689     657
```

Program 9: Singly Linked List

1	**Problem Statement**

Consider the University "REVA" is having 100 students where, the details of each student like name, roll no and marks of 3 subjects is to be stored. Design and develop a C program to constructa singly linked list to enter records of different students in list, display the list and calculate the percentage of each student. Also count the number of students passed (scored >40 in all thesubjects).

2	**Student Learning Outcomes**

After successful execution of this program, the student shall able to

- implement single linked list.
- differentiate between array and linked list.
- implement various operations in single linked list.
- create and manage databases using linked list.

3	**Design and development of the Program**
3.1	**Theory**

A linked list is a data structure consisting of a group of nodes which together represent a sequence. In singly linked list each node is composed of data and a pointer (link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence.

Operations on Linked list:

- Creation of a linked list
- Traversal
- Searching an element
- Insertion of an element at different positions.
- Deletion of an element from different positions.
- Reversing a linked list

3.2	**Algorithm**

Input : choice to insert , delete or display the values of nodes
Variables used:t->node,start->pointer which points to the first node,n->data
Output: Singly linked list implemented

INSERTION AT BEGINNING

```
Step 1: temp←new node
Step 2: enter info to be inserted
Step 3: read n
Step 4:temp->info←n
Step 5: temp->link← start
Step 6: start← temp
```

INSERTION AT END

```
Step 1: p←start
Step 2:Repeat step 3 untill p->next=NULL
Step 3:     p←p->next
Step 4: t->next←NULL
Step 5: p->next←t
```

DELETION AT POSITION

```
Step 1: IF pos=1 then
Step 2:          temp←start
Step 3:          start←temp->link
Step 4: ELSE
Step 5:           p←start
Step 6:           FOR i←0 to i < pos-2 Repeat Steps 8
Step 7:                     p←p->link
                  END FOR
Step 8:           temp←p->link
Step 9:           p->link←temp->link
```

DISPLAY

```
Step 1: p←start
Step 2: WHILE(p!=NULL)
Step 3:            print p->info
Step 4:            p←p->link
```

3.3 Coding using C Language

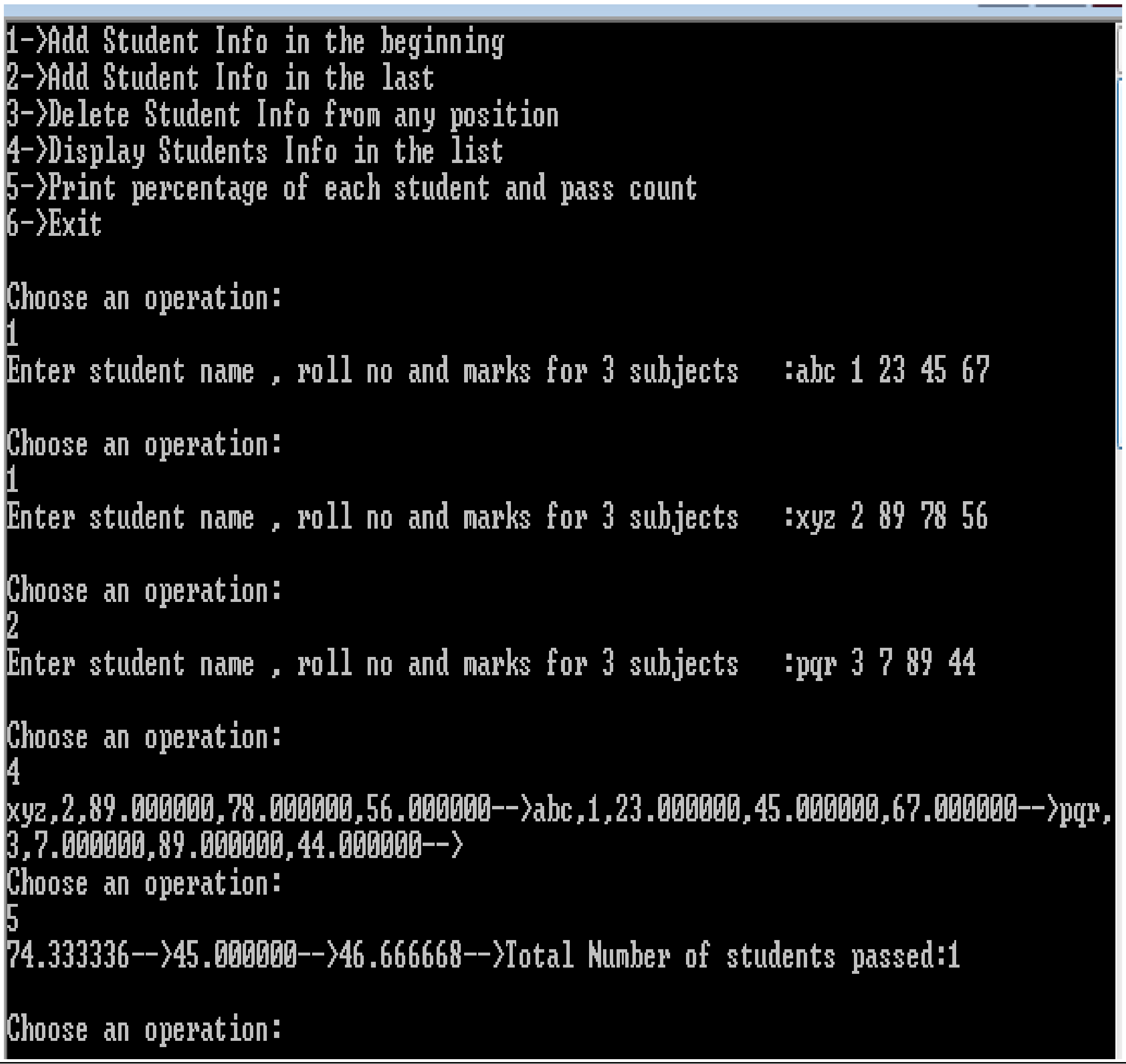
```
#include<stdio.h>
#include<stdlib.h>
struct REVA                                    /*creation of linked list*/
{
```

```
            char name[20];
             int rollno;
            float sub1,sub2,sub3;
            struct REVA *link;
}*start=NULL;
/*Function definition*/
struct REVA *addatbeg(struct REVA *start);
struct REVA *addatend(struct REVA *start);
struct REVA *delatpos(struct REVA *start,int pos);
void display(struct REVA *start);
void percentandcount(struct REVA *start);

main()
{
            int choice,pos;
            printf("1→Add Student Info in the beginning\n");
            printf("2→ Add Student Info in the last\n");
            printf("3→Delete Student Info from any position\n");
            printf("4→Display Students Info in the list\n");
            printf("5→Print percentage of each student and pass count\n");
            printf("6→Exit\n");
            while(1)
            {
                  printf("\nChoose an operation:\n");
                  scanf("%d",&choice);

                 switch(choice)
               {
                    case 1: start=addatbeg(start);
                            break;
                    case 2: start=addatend(start);
                             break;
                    case 3: printf("Enter the position\t:");
                            scanf("%d",&pos);
                            start=delatpos(start,pos);
                            break;
                    case 4: display(start);
                             break;
                    case 5: percentandcount(start);
                            break;
                    case 6: exit(0);
            }
    }
}/*End of main()*/
/*Function to insert at beginning*/
struct REVA *addatbeg(struct REVA *start)
{
    struct REVA *temp;
    temp=(struct REVA *)malloc(sizeof(struct REVA));
    printf("Enter student name , roll no and marks for 3 subjects\t:");
    scanf("%s%d%f%f%f",temp->name,&temp->rollno,&temp->sub1,&temp->sub2,&temp->sub3);
    temp->link=start;
    start=temp;
    return start;
} /*End of function*/
/*Function to insert at last*/
struct REVA *addatend(struct REVA *start)
{
    struct REVA *temp;
    temp=(struct REVA *)malloc(sizeof(struct REVA));
    printf("Enter student name , roll no and marks for 3 subjects\t:");
    scanf("%s%d%f%f%f",temp->name,&temp->rollno,&temp->sub1,&temp->sub2,
    &temp->sub3);
    struct REVA *p=start;
    while(p->link!=NULL)
    {
        p=p->link;
    }
    temp->link=NULL;
    p->link=temp;
    return start;
}/* End of function*/
  /*Function to delete from a position*/
struct REVA *delatpos(struct REVA *start,int pos)
{
    int i;
    struct REVA *temp;
    if(pos==1)
    {
       temp=start;
       start=temp->link;
    }
    else
    {
        struct REVA *p=start;
        for(i=0;i<pos-2;i++)
        {
         p=p->link;
        }
        temp=p->link;
        p->link=temp->link;
    }
    free(temp);
    return start;
}/* End of function*/
/*Function to display the list*/
void display(struct REVA *start)
{
    struct REVA *p=start;
    while(p!=NULL)
    {
       printf("%s,%d,%f,%f,%f-->",p->name,p->rollno,p->sub1,p->sub2,p->sub3);
       p=p->link;
    }
}/* End of function*/
/*Function to display percentage and pass count*/
void percentandcount(struct REVA *start)
{
    struct REVA *p=start;
    float per;
    int count=0;
    while(p!=NULL)
    {
       per=(p->sub1+p->sub2+p->sub3)/3;
       printf("%f-->",per);
       if(p->sub1>40&&p->sub2>40&&p->sub3>40)
          count++;
       p=p->link;
    }
    printf("Total Number of students passed:%d\n",count);
}/*End of function*/
```

Explanation of the program

- Lines 3 to 9 declare structure for node
- Lines 10 to 14 declare all functions
- Lines 30 to 40 call functions
- Lines 46 to 55 define the function "addatbeg"
- Lines 56 to 70 define the function "addatend"
- Lines 71 to 92 define the function "delatpos"
- Lines 93 to 101 define function "display"
- Lines 102 to 116 define the function "percentandcount"

3.4	**Expected Results**



```
1->Add Student Info in the beginning
2->Add Student Info in the last
3->Delete Student Info from any position
4->Display Students Info in the list
5->Print percentage of each student and pass count
6->Exit

Choose an operation:
1
Enter student name , roll no and marks for 3 subjects   :abc 1 23 45 67

Choose an operation:
1
Enter student name , roll no and marks for 3 subjects   :xyz 2 89 78 56

Choose an operation:
2
Enter student name , roll no and marks for 3 subjects   :pqr 3 7 89 44

Choose an operation:
4
xyz,2,89.000000,78.000000,56.000000-->abc,1,23.000000,45.000000,67.000000-->pqr,
3,7.000000,89.000000,44.000000-->
Choose an operation:
5
74.333336-->45.000000-->46.666668-->Total Number of students passed:1

Choose an operation:
```

Program 10: Doubly Linked List

1 Problem Statement

Consider the company "abc" is having 100 employees where, the details of each employee like name and id is to be stored. Design and develop a C program to construct a doubly linked list to enter records of different employees in list, display the list and print the names of all employeeswith particular employee id.

2 Student Learning Outcomes

After successful execution of this program, the student shall able to

- implement doubly linked list.
- differentiate between array and linked list.
- implement various operations in doubly linked list.
- create and manage databases using linked list.

3 Design and development of the Program

3.1 Theory

In a doubly linked list, each node contains, besides the next-node link, a second link field pointing to the 'previous' node in the sequence. The two links may be called 'forward' and 'backward', or 'next' and 'prev'(previous).

Operations on Linked list:

- Creation of a linked list
- Traversal
- Searching an element
- Insertion/Deletion of an element at different positions.
- Reversing a linked list

3.2 Algorithm

Input : choice to insert , delete or display the values of nodes

Variables used:temp->node,start->pointer which points to the first node,n->data

Output: Doubly linked list implemented

ADDING A NEW VALUE AS A NEW NODE TEMP:

```
Step 1: temp←new node
Step 2: enter info to be inserted
Step 3: read n
Step 4: temp->info←n
Step 5:temp->next←NULL
Step 6: temp->prev←NULL
```

INSERTION AT BEGIN

```
Step 1: temp->prev ←NULL
Step 2: temp->next←start
Step 3: start←temp
```

INSERTION AT END

```
Step 1: p←start
Step 2:Repeat step 3 untill p->next=NULL
Step 3:     p←p->next
Step 4: temp->next←NULL
Step 5: temp->prev←p
Step 6: p->next←temp
```

DISPLAY

```
Step 1: p←start
Step 2: WHILE p=NULL Repeat Step 3 and 4
Step 3:       print p->info
Step 4:       p←p->next
```

SEARCH

```
Step 1: p←start
Step 2: pos←0
Step 3: WHILE p=NULL Repeat Step 3 and 4
Step 4:      pos←pos+1
Step 5:      IF p->info = search key
Step 6:           print "pos"
Step 7:      p←p->next
```

3.3 Coding using C Language

```
#include<stdio.h>
#include<stdlib.h>
/*structure definition*/
struct abc
{
        char name[20];
         int empid;
        struct abc *prev;
        struct abc *next;
}*start=NULL;
/*Function Declaration*/
struct abc *addatbeg(struct abc *start);
struct abc *addatend(struct abc *start);
```

```
void display(struct abc *start);
void search(struct abc *start,int key);
main()
{
        int choice,pos,key;
        printf("1->Add Employee Info in the beginning\n");
        printf("2->Add Employee Info in the last\n");
        printf("3->Display Etudents Info in the list\n");
        printf("4->Search for an Employee\n");
        printf("5->Exit\n");
        while(1)
        {
        printf("\nChoose an operation:\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: start=addatbeg(start);
                break;
            case 2: start=addatend(start);
                break;
            case 3: display(start);
                break;
            case 4: printf("Enter the employee id to be searched\t:");
                scanf("%d",&key);
                search(start,key);
                break;
            case 5: exit(0);
        }

  }
  getch();
}/*End of main()*/
/* Function to add element in beginning */
struct abc *addatbeg(struct abc *start)
{
     struct abc *temp;
     temp=(struct abc *)malloc(sizeof(struct abc));
     printf("Enter employee name and id\t:");
     scanf("%s%d",temp->name,&temp->empid);
     temp->prev=NULL;
     temp->next=start;
```

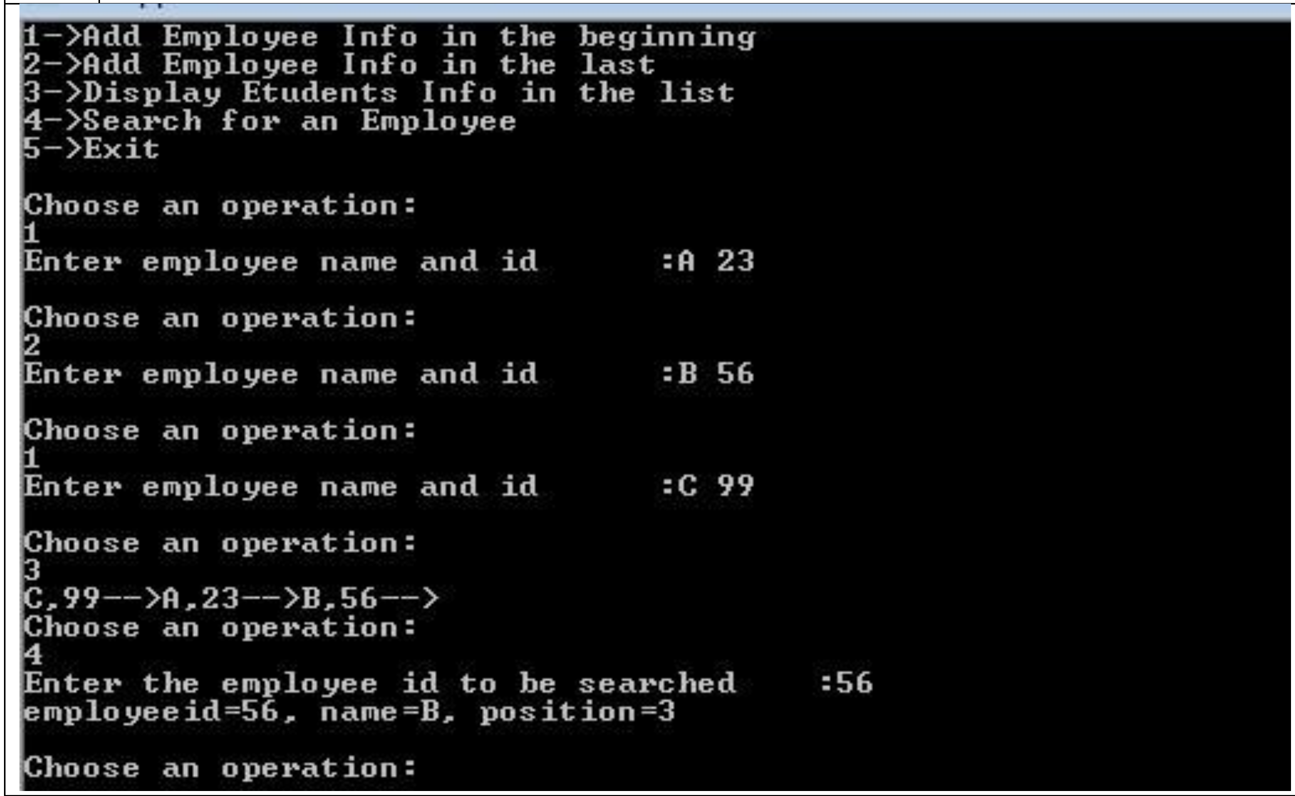
```
        start=temp;
        return start;
}/*End of addatbeg()*/
/* Function to add element at end */
struct abc *addatend(struct abc *start)
{
        struct abc *temp;
        temp=(struct abc *)malloc(sizeof(struct abc));    /*node creation*/
        printf("Enter employee name and id\t:");
        scanf("%s%d",temp->name,&temp->empid);
        struct abc *p=start;
        while(p->next!=NULL)
        {
            p=p->next;
        }
        temp->next=NULL;
        temp->prev=p;
        p->next=temp;
        return start;
}/*End of addatend()*/
/* Function to display elements (employees information) */
void display(struct abc *start)
{
        struct abc *p=start;
        while(p!=NULL)
        {
           printf("%s,%d-->",p->name,p->empid);
           p=p->next;
        }
}/*End of display()*/

/* Function to search an employee with particular id */
void search(struct abc *start,int key)
{
        struct abc *p=start;
        int pos=0;
        while(p!=NULL)
        {
           pos++;
           if(p->empid==key)
           {
```

```
89.             printf("employeeid=%d, name=%s, position=%d\n",p->empid,p->name,pos);
90.             break;
91.           }
92.           else
93.              p=p->next;
94.       }
95.       if(pos==0)
96.       printf("employee with id %d is not present in the list\n",p->empid);
97. }/*End of search()*/
```

Explanation of the program

- Lines 3 to 9 declare structure for node
- Lines 10 to 14 declaration of all functions
- Lines 30 to 38 call functions
- Lines 39 to 40 read search key
- Lines 50 to 60 define the function "addatbeg"
- Lines 61 to 76 define the function "addatend"
- Lines 101 to 109 define function "display"
- Lines 110 to 127 define the function "search"

3.4 Expected Results



```
1->Add Employee Info in the beginning
2->Add Employee Info in the last
3->Display Etudents Info in the list
4->Search for an Employee
5->Exit

Choose an operation:
1
Enter employee name and id      :A 23

Choose an operation:
2
Enter employee name and id      :B 56

Choose an operation:
1
Enter employee name and id      :C 99

Choose an operation:
3
C,99-->A,23-->B,56-->
Choose an operation:
4
Enter the employee id to be searched    :56
employeeid=56, name=B, position=3

Choose an operation:
```

List of Lab Assignments

- Design and develop a C program to create an array to store the information of employees like id, name and salary and use different searching techniques to return the id and name of all employees with given salary using Linear Search.

- Design and develop a C program to print the Pascal Triangle as given below using array.

```
        1
      1   1
    1  2   1
  1   3  3    1
```

- Design and develop a C program to store and retrieve customer database in a bank.
  Note: Customer database can include Customer id, Name, Account Number etc.

- Design and Develop a C program to convert a decimal number to binary using a stack.

- Design and develop a c program to convert an infix expression into prefix.

- Design a C program to print all the prime factors of a number in descending order using stack. For example the prime divisors of 450 are 5,5,3,3,2

- Design and implement a program to create a stack using queue.

- Construct a circular queue to create a employee database like name, id and perform all the operations – insert, delete, display.

- Construct a single linked list to store the employee database like employee name, id and salary and search for an employee with a particular salary.

- Construct a doubly linked list to store student database and print all the elements in reverse order.

- Design and develop a C program to implement Ackermann's function A(m, n) using recursive technique:

1) If m=0, A(m, n)=n+1
2) If m>0 and n=0, A(m, n)=A(m-1, 1)
3) If m>0 and n>0, A(m, n)=A(m-1, A(m, n-1))

- Design and develop a C program to find the minimum and maximum element in the binary search tree

- Construct a C program to search for a particular element in graph.

- Implement the above program using selection sort.

List of Viva Questions

- What are the I/O functions required to read and print the array elements?
- What will happen if you assign a value to an array element whose subscript exceeds the size of array?
- If you pass an array as an argument to a function, what actually gets passed?
- Which searching technique is more efficient when given array is sorted and why?
- Brief the logic of linear search.
- Brief the logic of binary search.
- What are the application of arrays?
- What are the disadvantage of arrays?
- What is meant by a “polynomial” and “degree of polynomial” ?
- Can array be passed as an argument to a function?
- What are the similarities and differences between structures and union?
- Can a structure be defined within another structure?
- Write a function to initialize a structure variable?
- What are the applications of stack ?
- Write functions for push, pop and peek operation for stack?
- Explain Tower of Hanoi and how is it related to STACK?
- Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix notation.
- Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent postfix notation.
- Convert the postfix expression 2 3 8 * + 4 48 4 2 + / 6 * + - into prefix expression.
- Convert the postfix expression 2 3 8 * + 4 48 4 2 + / 6 * + - into infix expression
- Evaluate 2 3 8 * + 4 48 4 2 + / 6 * + - postfix expression.
- Explain the algorithm to evaluate postfix expression.
- What are the applications of queue?
- What are the values of FRONT and REAR in different cases of Queue?
- Explain the logic behind insert and delete function.
- What are the applications of a queue and circular queue?
- What is a circular queue and doubly ended queue?
- Explain basic conditions for enqueue and dequeue operations in circular queue?
- What are the differences between array and linked list implementation of a circular queue?
- Is a Linked List linear or Non-linear data structure?
- What are the advantages and disadvantages of a linked list?
- Write a function to count the number of occurrences of an element in single linked list
- How do you insert a record between two nodes in double-linked list?

- Is a Linked List linear or Non-linear data structure?
- What are the advantages and disadvantages of a linked list?
- Write a function to count the number of occurrences of an element in single linked list
- How do you insert a record between two nodes in double-linked list?

- Give some real life example where linked list is used.
- What is the data structure used to implement the recursive technique?
- What do you mean by recursive definition?
- What are the advantages of recursion over iteration?
- Name some applications of recursion?
- List out the few applications of binary tree?
- Which of the data structures Array, Linked list, Stack and Queue is efficient in constructing a tree?
- Construct a binary tree from inorder 35,26,93,21,68 and preorder 68,21,93,26,35.
- Write a function to search a key in a Binary Search Tree.
- Write preorder, postorder, inorder and level order traversal of a given tree.
- What is sorting?
- Explain the logic of bubble sort?
- What is the time complexity for bubble sort?
- How many sorting algorithms are present?