Programming in C++ and Data Structures Lab Manual

II- B.Tech I- Sem (R-19)

RAJEEV GANDHI MEMORIAL

COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

2019-2023

(ESTD-1995)

VISION OF THE DEPARTMENT

- To empower students with cutting edge technologies in computer science and engineering
- To train the students as entrepreneurs in computer science and engineering to address the needs of the society
- To develop smart applications to disseminate information to rural people

MISSION OF THE DEPARTMENT

- To become the best computer science and engineering department in the region offering undergraduate, post graduate and research programs in collaboration with industry
- To incubate, apply and spread innovative ideas by collaborating with relevant industries and R & D labs through focused research groups.
- To provide exposure to the students in the latest tools and technologies to develop smart applications for the society

R G M COLLEGE OF ENGINEERING AND TECHNOLOGY

AUTONOMOUS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

II B.Tech. I-Sem (CSE) P C
3 1.5

(A0595193) PROGRAMMING IN C++ AND DATA STRUCTURES LAB

COURSE OBJECTIVES:

- ❖ To make the student learn an object oriented way of solving problems.
- ❖ Learn how to implement some useful data structures.
- ❖ Understand the effect of data structures on an algorithm's complexity.
- ❖ To develop skills to design and analyze simple linear data structures
- ❖ To Strengthen the ability to identify and apply the suitable data structure for the given real world problem
- ❖ To Gain knowledge in practical applications of data structures

COURSE OUTCOMES:

- ❖ Basic ability to analyse algorithms and to determine algorithm correctness and time efficiency class.
- ❖ Design, write, execute, and debug programs in C++.
- ❖ At the end of this lab session, the student will
- ❖ Be able to design and analyze the time and space efficiency of the data structure
- ❖ Be capable to identity the appropriate data structure for given problem
- ❖ Have practical knowledge on the application of data structures

MAPPING OF COs & POs

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	2	-	-	-	-	-	-	1	-	-	-	1	-	-	-
CO2	2	1	-	-	-	1	-	-	-	-	-	-	-	-	-
CO3	2	-	1	-	-	1	-	-	-	-	-	-	-	-	-
CO4	2	-	-	1	-	1	-	-	-	-	-	-	-	-	-
CO5	2	-	-	1	-	-	-	-	-	-	-	-	-	-	-
CO6	2	-	-	-	-	-	-	-	-	-	-	1	-	-	-

WEEK 1:

Write C++ program that convert the given expression from Infix to prefix using templates.

WEEK 2:

Write a C++ program to evaluate the given Postfix expression.

WEEK 3:

Write C++ program that implement all the operations on Circular Queue with array representation with templates.

WEEK 4:

Write C++ program that implement all the operations on DE Queue with array representation with templates.

WEEK 5:

Write C++ programs to implement the following using an array representation with templates.

a) Min Heap b) Max Heap

WEEK 6:

Write C++ programs to implement the following using an array representation with templates.

a) Ascending Priority Queue b) Descending Priority Queue

WEEK 7:

Write C++ program to implement Doubly Linked List for all operations along with templates.

WEEK 8:

Write a C++ program to implement the following operations on Binary Tree

a) Insert b) Delete c) Search d) Display

WEEK 9:

Write a C++ program to implement the following operations on Binary Search Tree

a) Insert b) Delete c) Search d) Display

WEEK 10:

Write a C++ program to implement the following collision resolution techniques using templates.

a) Linear Probing b) Quadratic Probing c) Double Hashing

TEXT BOOKS:

1. Object Oriented Programming Through C++, E. Balaguruswamy 6 Edition, 2013.
2. Data Structures using C++, Oxford, Varsha H. Patil.
3. Classic Data Structures, Debasis Samanta, PHI Learning Pvt Ltd, 2nd edition.
4. Data Structures and Algorithms in C++, Third Edition, 2006 Adam Drozdek, Thomson.
5. Data Structures using C++, D.S. Malik, Thomson

1. Write a C++ program to print multiplication table of a given number.

Description:

For example, for a number 2 with 3 rows, the output should be:

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
```

At the time of execution, the program should print the following messages one by one on the console as:

```
Enter an integer number :
Enter number of rows :
```

For example, if the user gives the input as:

```
Enter an integer number : 5
Enter number of rows : 4
```

then the program should print the result as:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
```

```cpp
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    int num,i,upto;
    cout << "Enter number u want table : ";
    cin >> num;
    cout << "Enter upto how many numbers u want : ";
    cin >> upto;
    cout << "Multiplication table of "<<num<<" is : "<<endl;
    for(i=1;i<=upto;i++)
        cout << num << " * " <<i<<" = "<<num*i<<endl;
    return 0;
}
```

Output: -

Test Case-1

```
Enter number u want table : 8
Enter upto how many numbers u want : 10
Multiplication table of 8 is :
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
```

Test Case-2

```
Enter number u want table : 9
Enter upto how many numbers u want : 10
Multiplication table of 9is :
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
```

2. Write a CPP program to find prime numbers up to n values (n must be user defined).

Description:

At the time of execution, the program should print the message on the console as:

Enter the value of N to find prime numbers up to:
For example, if the user gives the input as:
Enter the value of N to find prime numbers up to: 20
then the program should print the result as:
2 3 5 7 11 13 17 19

Source Code: -

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    int N, i, j, isPrime;
    cout << "Enter the value of N to find prime numbers upto : ";
    cin >> N;
    for(i = 2; i <= N; i++)
    {
        isPrime = 0;
        for(j = 2; j <= i/2; j++)
        {
            if(i % j == 0)
            {
                isPrime = 1;
                break;
            }
        }
    if(isPrime==0 && N!= 1)
        cout << i << " ";
    }
    return 0;
}
```

Output: -

Test Case -1	Test Case -2
Enter the value of N to find prime numbers upto : 30 **2 3 5 7 11 13 17 19 23 29**	Enter the value of N to find prime numbers upto : 50 **2 3 5 7 11 13 17 19 23 29 31 37 41 43 47**

3.Write a C++ program to perform arithmetic operations on two numbers. (Use switch case).

<u>Description:</u>

At the time of execution, the program should print the message on the console as:

Enter two integer values :

For example, if the user gives the input as:

Enter two integer values : 12 10

Next, the program should print the message on the console as:

Enter an arithmetic operator :

For example, if the user gives the input as:

Enter an arithmetic operator : +

then the program should print the result as:

12 + 10 = 22

For example, if the input given as

Enter two integer values : 12 0

Enter an arithmetic operator : /

then the output should be

Division is not possible! Divide by zero error

For example, if the input given as

Enter two integer values : 5 0

Enter an arithmetic operator : %

then the output should be

Modulo division is not possible! Divide by zero error

Source Code

```
#include<iostream>
#include<iomanip>
using namespace std;
int main()
{
    int a,b;
    char ch;
    cout<< "Enter a and b values : ";
    cin >> a >> b;
    cout << "CatLog To Select Certain Operations "<<endl;
    cout << "+ -> Addition"<<endl;
    cout << "- -> Subtraction"<<endl;
    cout << "* -> Multiplication"<<endl;
    cout << "/ -> Division"<<endl;
    cout << "% -> Modulo Division"<<endl;
    cout << "Enter any above sign to perform certain operations : ";
    cin >> ch ;
    switch(ch)
    {
         case '+' :
                    cout << "Addition of "<< a <<"and "<< b <<" is : "<<(a+b)<<endl;
                    break;
         case '-' :
                    cout << "Substraction of "<< a <<"and "<< b <<" is : "<<(a-b)<<endl;
                    break;
         case '*' :
                    cout << "Multiplication of "<< a <<"and "<< b <<" is :"<<(a*b)<<endl;
                    break;
         case '/' :
                    cout << "Division of "<< a <<"and "<< b <<" is : "<<(a/b)<<endl;
                    break;
         case '%' :
                    cout << "Modulo of "<< a <<"and "<< b <<" is : "<<(a%b)<<endl;
                    break;
         default : cout<< "Enter the operations seen in catlog"<<endl;
    }
    return 0;
}
```

Output: -

Test Case-1	Test Case-2
Enter a and b values : 3 9 CatLog To Select Certain Operations + -> Addition - -> Subtraction * -> Multiplication / -> Division % -> Modulo Division Enter any above sign to perform certain operations + Addition of 3and 9 is : 12	Enter a and b values : 3 6 CatLog To Select Certain Operations + -> Addition - -> Subtraction * -> Multiplication / -> Division % -> Modulo Division Enter any above sign to perform certain operations + Addition of 3and 6 is : 9

4. Write a C++ program to swap two numbers using:

a. Call -by-value

b. Call -by-reference

Description:

Call -by-value

During execution, the program should print the following message on the console as:

```
Enter two integer values :
```

For example, if the user gives the following input as:

```
Enter two integer values : 22 44
```

Then the program should print the result as follows:

```
Before swapping in main : a = 22 b = 44
After swapping in swap : x = 44 y = 22
After swapping in main : a = 22 b = 44
```

Call -by-reference

At the time of execution, the program should print the message on the console as:

```
Enter two integer values :
```

For example, if the user gives the input as:

```
Enter two integer values : 12 13
```

then the program should print the result as:

```
Before swapping in main : a = 12 b = 13
After swapping in swap : *p = 13 *q = 12
After swapping in main : a = 13 b = 12
```

Source Code:

```
#include<iostream>
using namespace std;
void swap_value(int ,int);
void swap_reference(int&,int&);
int main()
{
    int a,b;
    cout << "Enter a and b values : ";
    cin >> a >> b;
    cout<< "Swap two numbers by call by value "<<endl;
    swap_value(a,b);
    cout<< "Swap two numbers by call by reference"<<endl;
    swap_reference(a,b);
    return 0;
}
```

```
void swap_value(int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
    cout <<"After swapping by call by value : "<<endl;
    cout << "a = "<<a<<endl;
    cout << "b = "<<b<<endl;
}
void swap_reference(int &a, int &b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
    cout <<"After swapping by call by value : "<<endl;
    cout << "a = "<<a<<endl;
    cout << "b = "<<b<<endl;
}
```

Output: -

Test Case-1	Test Case-2
Enter a and b values : 10 20 **Swap two numbers by call by value** After swapping by call by value : a = 20 b = 10 **Swap two numbers by call by reference** After swapping by call by value : a = 20 b = 10	Enter a and b values : 50 60 **Swap two numbers by call by value** After swapping by call by value : a = 60 b = 50 **Swap two numbers by call by reference** After swapping by call by value : a = 60 b = 50

5. Write a C++ program to find factorial of a given number. (using recursive and non-recursive functions)

Description:
At the time of execution, the program should print the message on the console as:

Enter an integer :

For example, if the user gives the input as:

Enter an integer : 5

then the program should print the result as:

Factorial of given number 5 = 120

Source Code: -

```
#include<iostream>
#include<iomanip>
using namespace std;
int fact(int );
int fact_recur(int);
int main()
{
    int num,a,b;
    cout<< "Enter a number to find it's factorial: ";
    cin >> num;
    cout << "Factorial without using recursive function"<<endl;
    a=fact(num);
    cout<< "Factorial of given number "<<num<<" is "<<a<<endl;
    cout << "Factorial with using recursive function"<<endl;
    b=fact_recur(num);
    cout<< "Factorial of given number "<<num<<" is "<<b<<endl;
    return 0;
}
int fact(int num)
{
    int i;
    int     fac=1;
    for(i=1;i<=num;i++)
        fac=fac*i;
    return fac;
}
```

```
int fact_recur(int num)
{
    if(num==1)
        return 1;
    else
        return (num*fact_recur(num-1));
}
```

Output : -

Test Case-1	Test Case-2
Enter a number to find it's factorial: 10 Factorial without using recursive function Factorial of given number 10 is 3628800 Factorial with using recursive function Factorial of given number 10 is 3628800	Enter a number to find it's factorial: 6 Factorial without using recursive function Factorial of given number 6 is 720 Factorial with using recursive function Factorial of given number 6 is 720

6. Write a C++ program to find addition of two given numbers using inline function.

Description:
At the time of execution, the program should print the message on the console as:

Enter a and b values :

For example, if the user gives the input as:

Enter a and b values : 2 3

then the program should print the result as:

Addition of 2 and 3 = 5

Source Code: -

```
#include<iostream>
#include<iomanip>
using namespace std;
int add(int,int);
int main()
{
    int a,b,c;
    cout << "Enter a and b values : ";
    cin >> a >> b;
    c=add(a,b);
    cout << "Addition of "<<a<<" and "<<b<<" = "<<c<<endl;
    return 0;
}
inline int add(int a,int b)
{
    return(a+b);
}
```

Output: -

Test Case-1	Test Case-2
Enter a and b values: 3 6 Addition of 3 and 6 = 9	Enter a and b values: 10 56 Addition of 10 and 56 = 66

7. Write a C++ program to find Max and Min of two given numbers using inline functions.

Description:

At the time of execution, the program should print the message on the console as:

Enter a and b values :

For example, if the user gives the input as:

Enter a and b values : 2 3

then the program should print the result as:

Maximum Value of the two numbers is: 3

Minimum Value of the two numbers is: 2

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
int max_min(int,int);
int main()
{
    int c,d,a;
    cout<< "Enter a and b values : ";
    cin >> c >> d ;
    a=max_min(c,d);
    if(a==c)
    {
        cout << "Maximum Value of the two numbers is : "<<c<<endl;
    }
    if (a==d)
    {
        cout << "Maximum Value of the two numbers is : "<<d<<endl;
    }
    if(a!=c)
    {
        cout << "Minimum Value of the two numbers is : "<<c<<endl;
    }
    if (a!=d)
    {
        cout << "Minimum Value of the two numbers is : "<<d<<endl;
    }
    return 0;
}
inline int max_min(int a,int b)
{
    if(a>b)
        return a;
    else
        return b;
}
```

Output:

Test Case-1	Test Case-2
Enter a and b values: 10 6 Maximum Value of the two numbers is: 10 Minimum Value of the two numbers is: 6	Enter a and b values: 3 2 Maximum Value of the two numbers is: 3 Minimum Value of the two numbers is: 2

8. Write all the following programs using Object-Oriented implementation (using classes and objects).

a. Write a C++ program to find sum and average of N numbers.

b. Write a C++ program to print squares of numbers and cubes of numbers between 1 and N.

c. Write a C++ program to print Fibonacci series up to N.

Description:

At the time of execution, the program should print the message on the console as:

```
Enter the number u want: 10
```

then the program should print the result as:

```
Sum of first 10 numbers is = 55
Average of the 10 numbers is = 5
```

At the time of execution, the program should print the message on the console as:

```
Enter the upto which number u want for squares and cubes: 10
```

then the program should print the result as:

```
Squares of first 10 are
1 4 9 16 25 36 49 64 81 100
```

At the time of execution, the program should print the message on the console as:

```
Cubes of first 10 are
```

then the program should print the result as:

```
1 8 27 64 125 216 343 512 729 1000
```

At the time of execution, the program should print the message on the console as:

```
Fibonacci series
```

then the program should print the result as:

```
Enter the number of terms: 6
Fibonacci Series:  0 1 1 2 3 5
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class sum_avg
{
    public:
        int n,total=0;
        float avg;
};
class squares_cubes
{
    public:
        int end;
};
class fibo
{
    public:
        int n,t1=0,t2=1,nextTerm=0;
};
int main()
{
    // Sum and average
    sum_avg s1;
    int i;
    cout << "Enter the number u want : ";
    cin >> s1.n;
    for(i=1;i<=s1.n;i++)
        s1.total=+s1.total+i;
    s1.avg=s1.total/s1.n;
    cout << "Sum of first "<<s1.n<<" numbers is = "<<s1.total<<endl;
    cout << "Average of the "<< s1.n << " numbers is = " << s1.avg<<endl;
    // Squares and cubes
    squares_cubes sq;
    int n;
    cout <<endl<< "Enter the upto which number u want for sauares and cubes : ";
    cin >> sq.end;
    cout << "Squares of first " << sq.end<<" are "<<endl;
    for (i=1;i<=sq.end;i++)
        cout << i*i<<" ";
    cout << endl;
    cout << "Cubes of first " << sq.end<<" are "<<endl;
    for(i=1;i<=sq.end;i++)
        cout << i*i*i <<" ";
    cout <<endl;
    // Fibonacci series upto N
    fibo f;
    cout << "Fibonacci series"<<endl;
    cout << "Enter the number of terms: ";
    cin >> f.n;
    cout << "Fibonacci Series: ";
    for (int i = 1; i <= f.n; i++)
    {
        if(i == 1)
        {
            cout << " " << f.t1<<" ";
            continue;
        }
        if(i == 2)
        {
            cout << f.t2 << " ";
            continue;
        }
        f.nextTerm = f.t1 + f.t2;
        f.t1 = f.t2;
        f.t2 = f.nextTerm;
        cout << f.nextTerm << " ";
    }
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter the number u want: 10 Sum of first 10 numbers is = 55 Average of the 10 numbers is = 5 Enter the upto which number u want for squares and cubes: 10 Squares of first 10 are 1 4 9 16 25 36 49 64 81 100 Cubes of first 10 are 1 8 27 64 125 216 343 512 729 1000 Fibonacci series Enter the number of terms: 6 Fibonacci Series: 0 1 1 2 3 5	Enter the number u want: 100 Sum of first 100 numbers is = 5050 Average of the 100 numbers is = 50 Enter the upto which number u want for squares and cubes: 20 Squares of first 20 are 1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361 400 Cubes of first 20 are 1 8 27 64 125 216 343 512 729 1000 1331 1728 2197 2744 3375 4096 4913 5832 6859 8000 Fibonacci series Enter the number of terms: 10 Fibonacci Series: 0 1 1 2 3 5 8 13 21 34

9. Write a C++ program to illustrate use of default arguments.

Description:

At the time of execution, the program should print the message on the console as:

Enter a and b values: 2 3

then the program should print the result as:

Sum of 2 and 3 is = 5

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
int sum(int ,int);
int sum(int ,int ,int);
int sum(int ,int ,int ,int);
int main()
{
    int c,a,b;
    cout << "Enter a and b values : ";
    cin >> a >> b;
    c=sum(a,b);
    cout << "Sum of "<<a<<" and "<<b<<" is = "<<c<<endl;
    return 0;
}
int sum(int a,int b)
{
    return (a+b);
}
int sum(int a,int b,int z=0)
{
    return (a+b+z);
}
int sum(int a,int b,int z=0,int w=0)
{
    return (a+b+z+w);
}
```

Output:

Test Case-1	Test Case-2

10. Write a C++ program to illustrate Static data members & Static member functions.

Description:

At the time of execution, the program should print the message on the console as:

```
With static the data after incremrnt : 1

With static the data after incremrnt : 2

Without static the data after increment : 1

Without static the data after increment : 1
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class member
{
    public:
        static int a;// Static Data Member
        int b=1;
        static int increment() // Static Data Member Function
        {
             static int count =0;// Static Data Member
             count++;
            return (count);
        }
};
int main ()
{
    member c;
    cout << "With static the data after incremrnt : "<<c.increment()<<endl;
    cout << "With static the data after incremrnt : "<<c.increment()<<endl;
    cout << "Without static the data after increment : "<<c.increment()<<endl;
    cout << "Without static the data after increment : "<<c.increment()<<endl;
    return 0;
}
```

Output:

Test Case-1	Test Case-2
With static the data after incremrnt : 1 With static the data after incremrnt : 2	Without static the data after increment : 1 Without static the data after increment : 1

11. Write a C++ program to find sum of two times (using objects as arguments concept).

Description:

At the time of execution, the program should print the message on the console as:

```
Enter Time  hours :

Enter Time  minutes :
```

then the program should print the result as:

```
Enter Time  hours: 3

Enter Time  minutes: 55

Enter Time  hours: 4

Enter Time  minutes: 29

Total hours is: 8

Total minutes is  24
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class Time
{
    public:
        int hours, min;
        void read ( )
        {
            cout << "Enter Time  hours : ";
            cin >> hours;
            cout << "Enter Time  minutes : ";
            cin >> min;
        }

};
void sum(Time t1, Time t2)
{
    Time t33;
    t33.hours=t1.hours+t2.hours;
    t33.min=t1.min+t2.min;
    if(t33.min>=60)
    {
        t33.hours=t33.hours+1;
        t33.min=t33.min-60;
    }
    cout<<"Total hours is : "<<t33.hours<<endl;
    cout<< "Total minutes is : "<<t33.min<<endl;
}
int main()
{
    Time t1;
    t1.read();
    Time t2;
    t2.read();
    sum(t1,t2);
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter Time hours: 3 Enter Time minutes: 55 Enter Time hours: 4 Enter Time minutes: 29 Total hours is: 8 Total minutes is 24	Enter Time hours : 4 Enter Time minutes : 52 Enter Time hours : 6 Enter Time minutes : 40 Total hours is : 11 Total minutes is : 32

12. Write a C++ program to illustrate use of friend functions.

Description:

At the time of execution, the program should print the message on the console as:

```
Enter the real part :

Enter the imaginary part :
```

then the program should print the result as:

```
Enter the real part : 3

Enter the imaginary part : 6

Given  Number is : 3 + 6i

Enter the real part : 2

Enter the imaginary part : 4

Given  Number is : 2 + 4i

Given  Number is : 5 + 10i
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class Complex
{
    int rp,ip;
    friend Complex sumData(Complex c1,Complex c2);
    public:
        void getdata()
        {
            cout << "Enter the real part : ";
            cin >> rp;
            cout << "Enter the imaginary part : ";
            cin >> ip;
        }
        void PrintData()
        {
            cout << "Given  Number is : "<<rp<< " + "<<ip<<"i"<<endl;
        }
};
Complex sumData(Complex c1,Complex c2)
{
    Complex c3;
    c3.rp=c1.rp+c2.rp;
    c3.ip=c1.ip+c2.ip;
    return c3;
}
int main()
{
    Complex c1,c2,c3;
    c1.getdata();
    c1.PrintData();

    c2.getdata();
    c2.PrintData();

    c3=sumData(c1,c2);
    c3.PrintData();
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter the real part : 3 Enter the imaginary part : 6 Given Number is : 3 + 6i Enter the real part : 2 Enter the imaginary part : 4 Given Number is : 2 + 4i Given Number is : 5 + 10i	Enter the real part : 1 Enter the imaginary part : 5 Given Number is : 1 + 5i Enter the real part : 9 Enter the imaginary part : 12 Given Number is : 9 + 12i Given Number is : 10 + 17i

13. Write a C++ program to illustrate use of constructor and destructor.

Description:

At the time of execution, the program should print the message on the console as:

```
Inside the main function
Creating 1st object
This is the time constructor is called for object number 1
Entering into the block
Creating two more objects
This is the time constructor is called for object number 2
This is the time constructor is called for object number 3
Exiting this block
This is the tome when destructor is called for object number 3
This is the tome when destructor is called for object number 2
Back to main ()
This is the tome when destructor is called for object number 1
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
int count=0;
class number
{
    public:
        number()
        {
            count++;
            cout << "This is the time constructor is called for object number "<<    coun
t <<endl;
        }
        ~number()
        {
            cout << "This is the tome when destructor is calles for object number "<<coun
t<<endl;
            count--;
        }
};
```

```
int main()
{
      cout << "Inside the main function"<<endl;
      cout << "Creating 1st object"<<endl;
      number n1;
      {
          cout << "Entering into the block"<<endl;
          cout << "Creating two more objects"<<endl;
          number n2,n3;
          cout << "Exiting this block"<<endl;
      }
      cout << "Back to main ()"<<endl;
    return 0;
}
```

Output:

Test Case-1

```
Inside the main function
Creating 1st object
This is the time constructor is called for object number 1
Entering into the block
Creating two more objects
This is the time constructor is called for object number 2
This is the time constructor is called for object number 3
Exiting this block
This is the tome when destructor is called for object number 3
This is the tome when destructor is called for object number 2
Back to main ()
This is the tome when destructor is called for object number 1
```

14. Write a C++ program to perform various arithmetic operations on two complex numbers using friend functions.

Description:

During execution, the program should print the following messages one by one on the console as:

Enter the real and imaginary parts of the first complex number :

Enter the real and imaginary parts of the second complex number :

For example, if the user gives the input as:

Enter the real and imaginary parts of the first complex number : 2 3

Enter the real and imaginary parts of the second complex number : 4 5

Then the program should print the result as:

The First complex number is : 2 + i3

TheSecond complex number is : 4 + i5

Sum of the two given complex numbers is : 6 + i8

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class Calculator
{
    float a,b;
    friend  void addcomplex(Calculator,Calculator);
    friend  void subcomplex(Calculator,Calculator);
    friend  void mulcomplex(Calculator,Calculator);
    friend  void divcomplex(Calculator,Calculator);
    public:
        void getdata()
        {
            cout << "Enter a value : ";
            cin >> a;
            cout << "Enter b value : ";
            cin >> b;
        }
        void PrintData()
        {
            cout << "Complex number is : "<< a << " + i "<<b<<endl;
        }
};
void addcomplex(Calculator c1,Calculator c2)
```

```
{
    Calculator c3;
    c3.a=c1.a+c2.a;
    c3.b=c1.b+c2.b;
    cout << "Resultant Complex number for addition  is "<<c3.a<<" + i "<<c3.b<<endl;
}
void subcomplex(Calculator c1,Calculator c2)
{
    Calculator c3;
    c3.a=c1.a-c2.a;
    c3.b=c1.b-c2.b;
    cout << "Resultant Complex number for subtraction is "<<c3.a<<" + i"<<c3.b<<endl;
}

void mulcomplex(Calculator c1,Calculator c2)
{
    Calculator c3;
    c3.a=(c1.a*c2.a)-c1.b*c2.b;
    c3.b=c2.a*c1.b+c1.a*c2.b;
    cout << "Resultant Complex number for mutiplication is "<<c3.a<<" + i"<<c3.b<<end
l;
}
void divcomplex(Calculator c1,Calculator c2)
{
    Calculator c3;
    double a,b,c;
    if(c2.b<0)
    {
        c3.a=c1.a*c2.a-c1.b*c2.b;
        c3.b=(c2.a*c1.b-c1.a*c2.b);
        a=c2.a*c2.a+c2.b*c2.b;
        b=(c1.a*c2.a+c1.b*c2.b)/a;
        c=(c2.a*c1.b-c1.a*c2.b)/a;
        cout << "Resultant complex number for division is : "<<"("<<b<<" + i "<<c<<")
"<<endl;
    }
    else
    {
        c3.a=c1.a*c2.a+c1.b*c2.b;
        c3.b=(c2.a*c1.b-c1.a*c2.b);
        a=c2.a*c2.a+c2.b*c2.b;
        b=(c1.a*c2.a+c1.b*c2.b)/a;
        c=(c2.a*c1.b-c1.a*c2.b)/a;
        cout << "Resultant complex number for division is : "<<"("<<b<<" + i "<<c<<")
"<<endl;
```

```
    }

}
int main()
{
    Calculator c1,c2,c3;

    c1.getdata();
    c1.PrintData();
    c2.getdata();
    c2.PrintData();
    addcomplex(c1,c2);
    subcomplex(c1,c2);
    mulcomplex(c1,c2);
    divcomplex(c1,c2);
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter a value : 3 Enter b value : 6 Complex number is : 3 + i 6 Enter a value : 2 Enter b value : 5 Complex number is : 2 + i 5 Resultant Complex number for addition is 5 + i 11 Resultant Complex number for subtraction is 1 + i1 Resultant Complex number for mutiplication is -24 + i27 Resultant complex number for division is : (1.24138 + i -0.103448)	Enter a value : 8 Enter b value : 6 Complex number is : 8 + i 6 Enter a value : 2 Enter b value : 4 Complex number is : 2 + i 4 Resultant Complex number for addition is 10 + i 10 Resultant Complex number for subtraction is 6 + i2 Resultant Complex number for mutiplication is -8 + i44 Resultant complex number for division is : (2 + i -1)

15.Write a C++ program to find sum of N numbers using DMA concept. (Write a C++program to compute sum of N elements (using an array). Array should be allocated using new operator prior to the reading of values from the user, and de-allocate the array at the end of the program).

Description:

During execution, the program should print the following messages one by one on the console as:

```
Enter aray size : 5
Enter array elements
Enter the value of arr[0] = 1
Enter the value of arr[1] = 2
Enter the value of arr[2] = 3
Enter the value of arr[3] = 2
Enter the value of arr[4] = 3
```

For example, if the user gives the input as:

```
Elements present in the array are :
Value at arr[0] = 1
Value at arr[1] = 2
Value at arr[2] = 3
Value at arr[3] = 2
Value at arr[4] = 3
Sum of 5 elements of the array are = 11
```

Source Code:

```
#include<iostream>3
#include<iomanip>
using namespace std;
int main()
{
    int n,i,sum=0;
    cout << "Enter aray size : ";
    cin >> n;
    int *arr = new int(n);
    cout << "Enter array elements "<<endl;
    for(i=0;i<n;i++)
    {
        cout << "Enter the value of arr["<<i<<"] = ";
        cin >> arr[i];
    }
    cout << "Elemnts present in the array are : "<<endl;
    for(i=0;i<n;i++)
    {
        cout << "Value at arr["<<i<<"] = "<<arr[i]<<endl;
        sum=sum+arr[i];
    }
    cout << "Sum of "<<n<<" elements of the array are = "<<sum<<endl;
    free(arr);
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter aray size : 5	Enter aray size : 5
Enter array elements	Enter array elements
Enter the value of arr[0] = 1	Enter the value of arr[0] = 1
Enter the value of arr[1] = 2	Enter the value of arr[1] = 2
Enter the value of arr[2] = 3	Enter the value of arr[2] = 5
Enter the value of arr[3] = 2	Enter the value of arr[3] = 6
Enter the value of arr[4] = 3	Enter the value of arr[4] = 4
Elemnts present in the array are :	Elemnts present in the array are :
Value at arr[0] = 1	Value at arr[0] = 1
Value at arr[1] = 2	Value at arr[1] = 2
Value at arr[2] = 3	Value at arr[2] = 5
Value at arr[3] = 2	Value at arr[3] = 6
Value at arr[4] = 3	Value at arr[4] = 4
Sum of 5 elements of the array are = 11	Sum of 5 elements of the array are = 18

16. Write a C++ program which implements single level inheritance

Description:

During execution, the program should print the following messages one by one on the console as:

```
Enter Unique number of the student : 120
Enter the name of the student : Aravind
Enter height of the student : 157
Enter Weight of the student : 58
```

For example, if the user gives the input as:

```
Name of the Student is : Aravind
Roll Number of the student is : 120
Height of the student is : 157
Weight of the student is : 58
```

Source Code:

```cpp
#include<iostream>
#include<iomanip>
using namespace std;
class Student
{
    int num;
    string name;
    public:
        void getdata()
        {
            cout << "Enter Unique number of the student : ";
            cin >> num;
            cout << "Enter the name of the student : ";
            cin >> name;
        }
        void putdata()
        {
            cout << "Name of the Student is : "<< name <<endl;
            cout << "Roll Number of the student is : "<<num << endl;
        }
};
class inher : public Student
{
    float height,weight;
    public :
        void getinher()
        {
            cout << "Enter height of the student : ";
            cin >> height;
            cout << "Enter Weight of the student : ";
            cin >> weight;
        }
        void putinher()
        {
            cout << "Height of the student is : "<<height<<endl;
            cout << "Weight of the student is : "<<weight<<endl;
        }
};



int main()
{
    class inher p;
    p.getdata();
    p.getinher();
    p.putdata();
    p.putinher();

    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter Unique number of the student : 120	Enter Unique number of the student : 63
Enter the name of the student : Aravind	Enter the name of the student : Rahul
Enter height of the student : 157	Enter height of the student : 168
Enter Weight of the student : 58	Enter Weight of the student : 69
Name of the Student is : Aravind	Name of the Student is : Rahul
Roll Number of the student is : 120	Roll Number of the student is : 63
Height of the student is : 157	Height of the student is : 168
Weight of the student is : 58	Weight of the student is : 69

17.Write a C++ program which implements multi-level inheritance.

Description:

During execution, the program should print the following messages one by one on the console as:

```
Enter Unique number of the student : 56

Enter the name of the student : Rahul

Name of the Student is : Rahul

Roll Number of the student is : 56

Enter student marks of subject 1 : 98

Enter student marks of subject 2 : 97

Enter student marks of subject 3 : 96
```

For example, if the user gives the input as:

```
Marks of Subject -1 is = 98

Marks of Subject -2 is = 97
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class Student
{
    int num;
    string name;
    public:
        void getdata()
        {
            cout << "Enter Unique number of the student : ";
            cin >> num;
            cout << "Enter the name of the student : ";
            cin >> name;
        }
        void putdata()
        {
            cout << "Name of the Student is : "<< name <<endl;
            cout << "Roll Number of the student is : "<<num << endl;
        }
};
class marks : public Student
{
    protected :
        float m1, m2,m3;
    public :
        void getmarks()
        {
            cout << "Enter student marks of subject 1 : ";
            cin >> m1;
            cout << "Enter student marks of subject 2 : ";
            cin >> m2;
            cout << "Enter student marks of subject 3 : ";
            cin >> m3;
        }
        void putmarks()
        {
            cout << "Marks of Subject -1 is = "<<m1<<endl;
            cout << "Marks of Subject -2 is = "<<m2<<endl;
            cout << "Marks of Subject -3 is = "<<m3<<endl;
        }
};
class result : public marks
{
    int total;
    float avg;
    public:
    void show()
    {
        total=m1+m2+m3;
        avg=total/3.0;
        cout << "Total   = "<<total<<endl;
        cout << "Average = "<<avg<<endl;
    }
};
int main()
{
    result s1;
    s1.getdata();
    s1.putdata();
    s1.getmarks();
    s1.putmarks();
    s1.show();
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter Unique number of the student : 56 Enter the name of the student : Rahul Name of the Student is : Rahul Roll Number of the student is : 56 Enter student marks of subject 1 : 98 Enter student marks of subject 2 : 97 Enter student marks of subject 3 : 96 Marks of Subject -1 is = 98 Marks of Subject -2 is = 97 Marks of Subject -3 is = 96 Total = 291 Average = 97	Enter Unique number of the student : 123 Enter the name of the student : Aravind Name of the Student is : Aravind Roll Number of the student is : 123 Enter student marks of subject 1 : 90 Enter student marks of subject 2 : 95 Enter student marks of subject 3 : 96 Marks of Subject -1 is = 90 Marks of Subject -2 is = 95 Marks of Subject -3 is = 96 Total = 281 Average = 93.6667

18.Write a C++ program which implements hierarchical inheritance.

Description:

During execution, the program should print the following messages one by one on the console as:

```
1.Savings accout
2.Curent account
Enter any one option : 2
Enter acc_no of the user : 546731
Enter the name of the user : Rahul
Account number of the user is = 546731
```

For example, if the user gives the input as:

```
Name of the user is : Rahul
Enter the balance amount in the users bank : 1800
The balance amount in your account is more than 1000 so need not to worry
```

Source Code:

```cpp
#include<iostream>
#include<iomanip>
using namespace std;
class Bank
{
    int acc_no;
    string name;
    public :
        void getdata()
        {
            cout << "Enter acc_no of the user : ";
            cin >> acc_no;
            cout << "Enter the name of the user : ";
            cin >> name;
        }
        void putdata()
        {
            cout << "Account number of the user is = "<<acc_no<<endl;
            cout << "Name of the user is : "<<name<<endl;
        }
};
class Savings : public Bank
{
    float balance;
    public :
        void getbal()
        {
            cout <<"Enter the balance amount in the users bank : ";
            cin >> balance;
            if (balance < 500)
                cout  << "For an savings account there should be a minimum balanc
e of 500 rupees"<<endl;
            else
                cout << "The balance amount in your account is more than 500 so n
eed not to worry"<<endl;
        }
};
class Current : public Bank
{
    float balance;
    public :
        void getbal()
        {
            cout <<"Enter the balance amount in the users bank : ";
            cin >> balance;
            if (balance < 1000)
                cout  << "For an Current account there should be a minimum balanc
e of 1000 rupees"<<endl;
            else
                cout << "The balance amount in your account is more than 1000 so
need not to worry"<<endl;
        }
};
int main()
{
    int option;
    cout << "1.Savings accout"<<endl;
    cout << "2.Curent account"<<endl;
    cout << "Enter any one option : ";
    cin >> option;
    if (option==1)
    {
        Savings s;
        s.getdata();
        s.putdata();
        s.getbal();
    }
    else if(option==2)
    {
        Current c;
        c.getdata();
        c.putdata();
        c.getbal();
    }
    else
    {
        cout << "Please enter the option 1 or 2 only"<<endl;
    }

    return 0;
}
```

Output:

Test Case-1	Test Case-2
1.Savings accout 2.Curent account Enter any one option : 2 Enter acc_no of the user : 546731 Enter the name of the user : Rahul Account number of the user is = 546731 Name of the user is : Rahul Enter the balance amount in the users bank : 1800 The balance amount in your account is more than 1000 so need not to worry	1.Savings accout 2.Curent account Enter any one option : 1 Enter acc_no of the user : 123546 Enter the name of the user : Aravind Account number of the user is = 123546 Name of the user is : Aravind Enter the balance amount in the users bank : 600 The balance amount in your account is more than 500 so need not to worry

19. Write a C++ program which implements multiple-inheritance.

Description:

During execution, the program should print the following messages one by one on the console as:

```
Enter the id of the student : 47
Enter the name of the student : Rahul
Enter marks of maths : 96
Enter marks of pyhsics : 95
Enter marks of chemistry : 96
Enter student marks in sports  : 96
```

For example, if the user gives the input as:

```
Id of the student is : 47
Name of the student is  : Rahul
Marks of Maths = 96
Marks of Physics = 95
Marks of Chemistry = 96
Sports marks of the candidate is : 96
Total Marks = 287
Average Marks = 95.6667
Average marks + Spmarks = 191.667
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class student
{
   public:
        int id;
        string name;
       void getdata()
        {
            cout << "Enter the id of the student : ";
            cin >> id;
            cout << "Enter the name of the student : ";
            cin >> name;
        }
       void putdataid()
        {
            cout << "Id of the student is : "<<id<<endl;
            cout << "Name of the student is  : "<<name<<endl;
        }
};
class marks
{
    public:
        int m,p,c;
        void getmarks()
        {
            cout << "Enter marks of maths : ";
            cin >>m;
            cout << "Enter marks of pyhsics : ";
            cin >>p;
            cout << "Enter marks of chemistry : ";
            cin >>c;
        }
        void putmarks()
        {
            cout << "Marks of Maths = "<<m<<endl;
            cout << "Marks of Physics = "<<p<<endl;
            cout << "Marks of Chemistry = "<<c<<endl;
        }
};
class sports
{
    public:
        int spmarks;
   // public:
        void getspmarks()
        {
            cout <<"Enter student marks in sports  : ";
            cin >> spmarks;
        }
        void putspmarks()
        {
            cout << "Sports marks of the candidate is : "<<spmarks<<endl;
        }
};
class result : public marks,public sports,public student
{
    public :
        int total;
        float average;
    //public :
        void show ()
        {
            total = m+p+c;
            average = total/3.0;
            cout << "Total Marks = "<<total<<endl;
            cout << "Average Marks = "<<average<<endl;
            cout << "Average marks + Spmarks = "<<average+spmarks<<endl;
        }
};
int main()
{
    result r1;
    r1.getdata();
    r1.getmarks();
    r1.getspmarks();
    r1.putdataid();
    r1.putmarks();
    r1.putspmarks();
    r1.show();
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter the id of the student : 47	Enter the id of the student : 49
Enter the name of the student : Rahul	Enter the name of the student : 96
Enter marks of maths : 96	Enter marks of maths : 66
Enter marks of pyhsics : 95	Enter marks of pyhsics : 96
Enter marks of chemistry : 96	Enter marks of chemistry : 98
Enter student marks in sports : 96	Enter student marks in sports : 99
Id of the student is : 47	Id of the student is : 49
Name of the student is : Rahul	Name of the student is : 96
Marks of Maths = 96	Marks of Maths = 66
Marks of Physics = 95	Marks of Physics = 96
Marks of Chemistry = 96	Marks of Chemistry = 98
Sports marks of the candidate is : 96	Sports marks of the candidate is : 99
Total Marks = 287	Total Marks = 260
Average Marks = 95.6667	Average Marks = 86.6667
Average marks + Spmarks = 191.667	Average marks + Spmarks = 185.667

20. Write a C++ program to implement hybrid inheritance using virtual base classes.

Description:

During execution, the program should print the following messages one by one on the console as:

```
Enter the id of the student : 47
Enter the name of the student : Rahul
Enter marks of maths : 96
Enter marks of pyhsics : 95
Enter marks of chemistry : 96
Enter student marks in sports  : 96
```

For example, if the user gives the input as:

```
Id of the student is : 47
Name of the student is  : Rahul
Marks of Maths = 96
Marks of Physics = 95
Marks of Chemistry = 96
Sports marks of the candidate is : 96
Total Marks = 287
Average Marks = 95.6667
Average marks + Spmarks = 191.667
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class student
{
    int id;
    string name;
    public :
        void getdata()
        {
            cout << "Enter the id of the student : ";
            cin >> id;
            cout << "Enter the name of the student : ";
            cin >> name;
        }
        void putdataid()
        {
            cout << "Id of the student is : "<<id<<endl;
            cout << "Name of the student is  : "<<name<<endl;
        }
};
class marks : virtual public student
{
    protected:
        int m,p,c;
    public:
        void getmarks()
        {
            cout << "Enter marks of maths : ";
            cin >>m;
            cout << "Enter marks of pyhsics : ";
            cin >>p;
            cout << "Enter marks of chemistry : ";
            cin >>c;
        }
        void putmarks()
        {
            cout << "Marks of Maths = "<<m<<endl;
            cout << "Marks of Physics = "<<p<<endl;
            cout << "Marks of Chemistry = "<<c<<endl;
        }
};
class sports  : public virtual student
{
    protected:
        int spmarks;
    public:
        void getspmarks()
        {
            cout <<"Enter student marks in sports  : ";
            cin >> spmarks;
        }
        void putspmarks()
            cout << "Sports marks of the candidate is : "<<spmarks<<endl;
};
class result : public marks,public sports
{
    protected :
        int total;
        float average;
    public :
        void show ()
        {
            total = m+p+c;
            average = total/3.0;
            cout << "Total Marks = "<<total<<endl;
            cout << "Average Marks = "<<average<<endl;
            cout << "Average marks + Spmarks = "<<average+spmarks<<endl;
        }
};
int main()
{
    result r1;
    r1.getdata();
    r1.getmarks();
    r1.getspmarks();
    r1.putdataid();
    r1.putmarks();
    r1.putspmarks();
    r1.show();
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter the id of the student : 47	Enter the id of the student : 49
Enter the name of the student : Rahul	Enter the name of the student : 96
Enter marks of maths : 96	Enter marks of maths : 66
Enter marks of pyhsics : 95	Enter marks of pyhsics : 96
Enter marks of chemistry : 96	Enter marks of chemistry : 98
Enter student marks in sports : 96	Enter student marks in sports : 99
Id of the student is : 47	Id of the student is : 49
Name of the student is : Rahul	Name of the student is : 96
Marks of Maths = 96	Marks of Maths = 66
Marks of Physics = 95	Marks of Physics = 96
Marks of Chemistry = 96	Marks of Chemistry = 98
Sports marks of the candidate is : 96	Sports marks of the candidate is : 99
Total Marks = 287	Total Marks = 260
Average Marks = 95.6667	Average Marks = 86.6667
Average marks + Spmarks = 191.667	Average marks + Spmarks = 185.667

21. Write a C++ program to illustrate use of function overloading. (Write a C++ program to find areas of different shapes (square, rectangle and triangle)).

Description:

During execution, the program should print the following messages one by one on the console as:

```
        Menu
  1. Area of the square
  2. Area of the Rectangle
  3. Area of the Triangle
  Enter the value from the menu
        Menu
        1. Area of the square
        2. Area of the Rectangle
        3. Area of the Triangle
        Enter the value from the menu : 3
        Enter length of side if Triangle : 34
```

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class Area
{
    public:
        double length,width,height;
};
void area(double length)
{
    cout << "Area of square is = "<<length*length<<endl;
}
void area(double length,double width)
{
    cout << "Area of rectangle is = "<<length*width<<endl;
}
void area(double a,double length,double height)
{
    cout << "Area of Triangle is = "<<a*(length*height)<<endl;
}
```

```
int main()
{
    char ch;
    Area a;
    cout << "Menu "<<endl;
    cout << "1. Area of the square"<<endl;
    cout << "2. Area of the Rectangle"<<endl;
    cout << "3. Area of the Triangle"<<endl;
    cout << "Enter the value from the menu : ";
    cin >> ch;
    switch (ch)
    {
        case '1':
                cout << "Enter the length of the Square : ";
                cin >> a.length;
                area(a.length);
                break;
        case '2':
                cout << "Enter the length of the Rectangle : ";
                cin >> a.length;
                cout << "Enter the width of the Rectangle : ";
                cin >> a.width;
                area(a.length,a.width);
                break;

        case '3':
                cout << "Enter length of side if Triangle : ";
                cin >> a.length;
                cout << "Enter the height of the Triangle : ";
                cin >> a.height;
                area((0.5),a.length,a.height);
                break;

        default:
                cout << "Please enter the value given in menu"<<endl;
    }
    return 0;
}
```

Output:

Test Case-1	Test Case-2	Test Case-03
Menu	Menu	Menu
1. Area of the square	1. Area of the square	1. Area of the square
2. Area of the Rectangle	2. Area of the Rectangle	2. Area of the Rectangle
3. Area of the Triangle	3. Area of the Triangle	3. Area of the Triangle
Enter the value from the menu : 3	Enter the value from the menu : 2	Enter the value from the menu : 2
Enter length of side if Triangle : 34	Enter the length of the Rectangle : 5	Enter the length of the Rectangle : 3
Enter the height of the Triangle : 4	Enter the width of the Rectangle : 6	Enter the width of the Rectangle : 5
Area of Triangle is = 68	Area of rectangle is = 30	Area of rectangle is = 15

22. Write a C++ program to illustrate use of operator overloading. (Write a C++ program to perform various arithmetic operations on two complex numbers).

Description:

During execution, the program should print the following messages one by one on the console as:

```
Enter rp value : 2

Enter ip value : 3

Complex number is : 2 + i 3

Enter rp value : 6

Enter ip value : 9

Complex number is : 6 + i 9
```

For example, if the user gives the input as:

```
Resultant Complex number for addition  is 8 + i 12

Resultant Complex number for subtraction is -4 + i-6

Resultant Complex number for mutiplication is -15 + i36

Resultant complex number for division is : (0.333333 + i 0)
```

Source Code:

```
#include<iostream>
#include<iomanip>
#include<math.h>
using namespace std;
class Complex
{
    public:
    double rp,ip;

        void getdata()
        {
            cout << "Enter rp value : ";
            cin >> rp;
            cout << "Enter ip value : ";
            cin >> ip;
        }
        void PrintData()
        {
            cout << "Complex number is : "<< rp << " + i "<<ip<<endl;
        }
        Complex operator+(Complex);
        Complex operator-(Complex);
        Complex operator*(Complex);
        Complex operator/(Complex);

};
Complex Complex ::  operator+(Complex c2)
{
    Complex c3;
    c3.rp=rp+c2.rp;
    c3.ip=ip+c2.ip;
    return c3;
}
Complex Complex ::  operator-(Complex c2)
{
    Complex c3;
    c3.rp=rp-c2.rp;
    c3.ip=ip-c2.ip;
    return c3;
}
Complex Complex ::  operator*(Complex c2)
{
    Complex c3;
    c3.rp=(rp*c2.rp)-ip*c2.ip;
    c3.ip=c2.ip*rp+ip*c2.rp;
    return c3;
}
Complex Complex ::  operator/(Complex c2)
{
    Complex c3;
    double a,b,c;

        a=(c2.rp*c2.rp)+(c2.ip*c2.ip);
        b=(rp*c2.rp)+(ip*c2.ip);
        c=(ip*c2.rp)-(rp*c2.ip);
        c3.rp=b/a;
        c3.ip=c/a;
        return c3;
}
int main()
{
    Complex c1,c2,c3,c4,c5,c6;

    c1.getdata();
    c1.PrintData();
    c2.getdata();
    c2.PrintData();
    c3=c1+c2;
    cout << "Resultant Complex number for addition  is "<<c3.rp<<" + i "<<c3.ip<<
endl;
    c4=c1-c2;
    cout << "Resultant Complex number for subtraction is "<<c4.rp<<" + i"<<c4.ip<
<endl;
    c5=c1*c2;
    cout << "Resultant Complex number for mutiplication is "<<c5.rp<<" + i"<<c5.i
p<<endl;
    c6=c1/c2;
    cout << "Resultant complex number for division is : "<<"("<<c6.rp<<" + i "<<c
6.ip<<")"<<endl;
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter rp value : 2 Enter ip value : 3 Complex number is : 2 + i 3 Enter rp value : 6 Enter ip value : 9 Complex number is : 6 + i 9 Resultant Complex number for addition is 8 + i 12 Resultant Complex number for subtraction is -4 + i-6 Resultant Complex number for mutiplication is -15 + i36 Resultant complex number for division is : (0.333333 + i 0)	Enter rp value : 5 Enter ip value : 4 Complex number is : 5 + i 4 Enter rp value : 3 Enter ip value : 6 Complex number is : 3 + i 6 Resultant Complex number for addition is 8 + i 10 Resultant Complex number for subtraction is 2 + i-2 Resultant Complex number for mutiplication is -9 + i42 Resultant complex number for division is : (0.866667 + i -0.4)

23. Write a C++ program that illustrates unary operator overloading for ++ and -- operators.

Description:

During execution, the program should print the following messages one by one on the console as:

The value after increment ++ : 1

The value after decrement -- : 0

Source Code:

```
#include<iostream>
#include<iomanip>
using namespace std;
class uninary
{
    public:
    int a;
        uninary()
        {
            a=0;
        }
        void putdata()
        {
            cout << "Value of a is before using the operator overloading : "<<a<<endl;
        }
        void operator++()
        {
            a++;
            cout << "The value after increment ++ : "<<a<<endl;
        }
        void operator--()
        {
            a--;
            cout << "The value after decrement -- : "<<a<<endl;
        }
};

int main()
{
    uninary u1;
    u1.operator++();
    u1. operator--();
    return 0;
}
```

Output:

Test Case-01

```
The value after increment ++ : 1
The value after decrement -- : 0
```

24. Write a C++ program to illustrate use of virtual functions.

Description:

During execution, the program should print the following messages one by one on the console as:

```
Base show function
Base display function
 Base show function
Derived display function
```

Source Code:

```
#include<iostream>
using namespace std;
class Base
{
    public:
    void show()
    {
        cout<<"\n Base show function";
    }
    virtual void display()
    {
        cout<<"\n Base display function";
    }
};
class Derived:public Base
{
    public:
    void show() //function overriding
    {
        cout<<"\n Derived show function";
    }
    void display() //function overriding
    {
    cout<<"\n Derived display function";
    }
};
int main()
{
    Derived ObjDerived;
    Base ObjBase;
    Base *ptr;
    ptr=&ObjBase;
    ptr->show();
    ptr->display();
    ptr=&ObjDerived;
    ptr->show();
    ptr->display();
    return 0;
}
```

Output:

```
Test Case-1
 Base show function
 Base display function
 Base show function
 Derived display function
```

25. Write a C++ program to illustrate:

a. Function templates.

b. Class templates.

Description:

During execution, the program should print the following messages one by one on the console as:

After swapping(int values):a 5 b: 3

After swapping(float values): c: 6.7 d: 4.5

Source Code:

```
#include<iostream>
using namespace std;
template<class T>
void swap1(T &x,T &y)
{
    T temp;
    temp=x;
    x=y;
    y=temp;
}
int main()
{
    int a=3,b=5;
    swap1(a,b);
    cout<<"\n After swapping(int values):a "<<a<<"b: "<<b;
    cout<<endl;
    float c=4.5,d=6.7;
    swap1(c,d);
    cout<<"\nAfter swapping(float values): c: "<<c<<"d: "<<d;
    return 0;
}
```

Output:

Test Case-1

```
 After swapping(int values):a 5 b: 3

After swapping(float values): c: 6.7 d: 4.5
```

26. Write a C++ program to illustrate use of exception handling.

Description:

During execution, the program should print the following messages one by one on the console as:

Enter the two integer values : 2 3

Result is : 0

Source Code:

```
#include <iostream>
#include<iomanip>
using namespace std;
float division(int x, int y)
{
    if( y == 0 )
    {
        throw "Attempted to divide by zero!";
    }
    return (x/y);
}
int main ()
{
    int a,b;
    cout<<"Enter the two integer values : ";
    cin>>a>>b;
    float c = 0;
    try
    {
        c = division(a, b);
    cout <<"Result is : "<< c << endl;
    }
    catch (const char* e)
    {
        cerr << e << endl;
    }
    return 0;
}
```

Output

Test Case-1	Test Case-2
Enter the two integer values : 2 3 Result is : 0	Enter the two integer values : 10 0 Attempted to divide by zero!

27. Implement doubly linked list ADT.

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward or backward easily as compared to Single Linked List.

Following are the important terms to understand the concept of doubly linked list.

• **Link** − Each link of a linked list can store a data called an element.

• **Next** − Each link of a linked list contains a link to the next link called Next.

• **Prev** − Each link of a linked list contains a link to the previous link called Prev.

• **LinkedList** − A Linked List contains the connection link to the first link called First and to the last link called Last.

- Doubly Linked List contains a link element called first and last.
- Each link carries a data field(s) and two link fields called next and prev.
- Each link is linked with its next link using its next link.
- Each link is linked with its previous link using its previous link.
- The last link carries a link as null to mark the end of the list.

Source Code:

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
/*
 * Node Declaration
 */
using namespace std;
struct node
{
    int info;
    struct node *next;
    struct node *prev;
}*start;
```

```
/*
 Class Declaration
 */
class double_llist
{
    public:
        void create_list(int value);
        void add_begin(int value);
        void add_after(int value, int position);
        void delete_element(int value);
        void search_element(int value);
        void display_dlist();
        void count();
        void reverse();
        double_llist()
        {
            start = NULL;
        }
};

/*
 * Main: Conatins Menu
 */
int main()
{
    int choice, element, position;
    double_llist dl;
    while (1)
    {
        cout<<endl<<"----------------------------"<<endl;
        cout<<endl<<"Operations on Doubly linked list"<<endl;
        cout<<endl<<"----------------------------"<<endl;
        cout<<"1.Create Node"<<endl;
        cout<<"2.Add at begining"<<endl;
        cout<<"3.Add after position"<<endl;
        cout<<"4.Delete"<<endl;
        cout<<"5.Display"<<endl;
        cout<<"6.Count"<<endl;
        cout<<"7.Reverse"<<endl;
        cout<<"8.Quit"<<endl;
        cout<<"Enter your choice : ";
        cin>>choice;
        switch ( choice )
        {
        case 1:
            cout<<"Enter the element: ";
            cin>>element;
            dl.create_list(element);
            cout<<endl;
            break;
        case 2:
            cout<<"Enter the element: ";
            cin>>element;
            dl.add_begin(element);
            cout<<endl;
            break;
        case 3:
            cout<<"Enter the element: ";
            cin>>element;
            cout<<"Insert Element after postion: ";
            cin>>position;
            dl.add_after(element, position);
            cout<<endl;
            break;
        case 4:
            if (start == NULL)
            {
                cout<<"List empty,nothing to delete"<<endl;
                break;
            }
            cout<<"Enter the element for deletion: ";
            cin>>element;
            dl.delete_element(element);
            cout<<endl;
            break;
        case 5:
            dl.display_dlist();
            cout<<endl;
            break;
        case 6:
            dl.count();
            break;
        case 7:
            if (start == NULL)
            {
                cout<<"List empty,nothing to reverse"<<endl;
                break;
            }
            dl.reverse();
            cout<<endl;
            break;
        case 8:
            exit(1);
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
    return 0;
}

/*
 * Create Double Link List
 */
void double_llist::create_list(int value)
{
    struct node *s, *temp;
    temp = new(struct node);
    temp->info = value;
    temp->next = NULL;
    if (start == NULL)
    {
        temp->prev = NULL;
        start = temp;
    }
    else
    {
        s = start;
        while (s->next != NULL)
            s = s->next;
        s->next = temp;
        temp->prev = s;
    }
}

/*
 * Insertion at the beginning
 */
void double_llist::add_begin(int value)
{
    if (start == NULL)
    {
        cout<<"First Create the list."<<endl;
        return;
    }
    struct node *temp;
    temp = new(struct node);
    temp->prev = NULL;
    temp->info = value;
    temp->next = start;
    start->prev = temp;
    start = temp;
    cout<<"Element Inserted"<<endl;
}

/*
 * Insertion of element at a particular position
 */
void double_llist::add_after(int value, int pos)
{
    if (start == NULL)
    {
        cout<<"First Create the list."<<endl;
        return;
    }
    struct node *tmp, *q;
    int i;
    q = start;
    for (i = 0;i < pos - 1;i++)
    {
        q = q->next;
        if (q == NULL)
        {
            cout<<"There are less than ";
            cout<<pos<<" elements."<<endl;
            return;
        }
    }
    tmp = new(struct node);
    tmp->info = value;
    if (q->next == NULL)
    {
        q->next = tmp;
        tmp->next = NULL;
        tmp->prev = q;
    }
    else
    {
        tmp->next = q->next;
        tmp->next->prev = tmp;
        q->next = tmp;
        tmp->prev = q;
    }
    cout<<"Element Inserted"<<endl;
}

/*
 * Deletion of element from the list
 */
void double_llist::delete_element(int value)
{
    struct node *tmp, *q;
     /*first element deletion*/
    if (start->info == value)
    {
        tmp = start;
        start = start->next;
        start->prev = NULL;
        cout<<"Element Deleted"<<endl;
        free(tmp);
        return;
    }
    q = start;
    while (q->next->next != NULL)
    {
        /*Element deleted in between*/
        if (q->next->info == value)
        {
            tmp = q->next;
            q->next = tmp->next;
            tmp->next->prev = q;
            cout<<"Element Deleted"<<endl;
            free(tmp);
            return;
        }
        q = q->next;
    }
     /*last element deleted*/
    if (q->next->info == value)
    {
        tmp = q->next;
        free(tmp);
        q->next = NULL;
        cout<<"Element Deleted"<<endl;
        return;
    }
    cout<<"Element "<<value<<" not found"<<endl;
}

/*
 * Display elements of Doubly Link List
 */
void double_llist::display_dlist()
{
    struct node *q;
    if (start == NULL)
    {
        cout<<"List empty,nothing to display"<<endl;
        return;
    }
    q = start;
    cout<<"The Doubly Link List is :"<<endl;
    while (q != NULL)
    {
        cout<<q->info<<" <-> ";
        q = q->next;
    }
    cout<<"NULL"<<endl;
}

/*
 * Number of elements in Doubly Link List
 */
void double_llist::count()
{
    struct node *q = start;
    int cnt = 0;
    while (q != NULL)
    {
        q = q->next;
        cnt++;
    }
    cout<<"Number of elements are: "<<cnt<<endl;
}

/*
 * Reverse Doubly Link List
 */
void double_llist::reverse()
{
    struct node *p1, *p2;
    p1 = start;
    p2 = p1->next;
    p1->next = NULL;
    p1->prev = p2;
    while (p2 != NULL)
    {
        p2->prev = p2->next;
        p2->next = p1;
        p1 = p2;
        p2 = p2->prev;
    }
    start = p1;
    cout<<"List Reversed"<<endl;
}
```

Output:

Test Case-1	Test Case-2
----------------------------	----------------------------
Operations on Doubly linked list	Operations on Doubly linked list
----------------------------	----------------------------
1.Create Node	1.Create Node
2.Add at begining	2.Add at begining
3.Add after position	3.Add after position
4.Delete	4.Delete
5.Display	5.Display
6.Count	6.Count
7.Reverse	7.Reverse
8.Quit	8.Quit
Enter your choice : 1	Enter your choice : 5

```
Enter the element: 50


----------------------------

Operations on Doubly linked list

----------------------------
1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 5
The Doubly Link List is :
50 <-> NULL


----------------------------

Operations on Doubly linked list

----------------------------
1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 2
Enter the element: 6
Element Inserted


----------------------------

Operations on Doubly linked list
```

```
List empty,nothing to display


----------------------------

Operations on Doubly linked list

----------------------------
1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 2
Enter the element: 49
First Create the list.


----------------------------

Operations on Doubly linked list

----------------------------
1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 6
Number of elements are: 1


----------------------------

Operations on Doubly linked list
```

```
-----------------------------
1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 8
```

```
-----------------------------
1.Create Node
2.Add at begining
3.Add after position
4.Delete
5.Display
6.Count
7.Reverse
8.Quit
Enter your choice : 8
```

28. Write a C++ program to convert a given Infix expression to prefix expression using templates.

Any mathematical expression can be written in three different but equivalent notations namely infix, postfix and prefix.

1. Infix notation : A + B
Operators are written in-between their operands. This is the usual way we write expressions.
An expression such as A * (B - C) / D is usually taken to mean something like: "First subtract C from B, then multiply the result by A, then divide by D to give the final answer."

Infix notation needs the following information to make the order of evaluation of the operators clear.

Precedence and Associativity rules that are built into the language.

brackets ( ) which allow users to override these rules.

The normal rule of associativity says that we perform operations from left to right, so the multiplication by A is assumed to come before the division by D.

Similarly, the usual rules for precedence say that we perform multiplication and division before we perform addition and subtraction. In this expression the subtraction happens before multiplication because of the parenthesis. [parenthesis are used to override the rules of associativity and precedence]

2. Prefix notation (also known as "Polish notation") : + A B
Operators are written before their operands. The expression in prefix notation / * A - B C D is equivalent to A * (B - C) / D

Generally all the parenthesis are removed from the prefix expression. Even if the parenthesis are present they are unnecessary.

Operators act on the two nearest values on the right.

Although in prefix notation "operators are evaluated left-to-right", they use values to their right, and if these values themselves involve computations then this changes the order that the operators have to be evaluated in.

In the example above, although the division is the first operator on the left, it acts on the result of the multiplication, and so the multiplication has to happen before the division (and similarly the subtraction has to happen before the multiplication).

3. Postfix notation (also known as "Reverse Polish notation") : A B +
Operators are written after their operands. The expression in postfix notation A B C - * D / is equivalent to A * (B - C) / D.

The order of evaluation of operators is always left-to-right, and brackets cannot be used to change this order. Because the "-" is to the left of the "*" in the example above, the subtraction must be performed before the multiplication.

Operators act on values immediately to the left of them. For this reason any expression is converted into postfix notation for evaluation.

conversion of infix to prefix expression:

- Find the reverse of the given infix expression.
- Apply the procedure to convert the infix to postfix expression.
- And finally get the reverse of the result found in the step 2.

Understanding conversion of infix to postfix expression :

- The order of operands do not change when we convert an expression from infix notation to postfix notation. For example: given 2+4*6 as the infix expression, the postfix expression is 246*+. Observe that the operands 2,4 and 6 are in the same order
- There are no parenthesis in the postfix expression. So they are not printed to the postfix expression. For Example : given (2+4)*6 as the infix expression, the postfix expression is 24+6*. Observe that the parenthesis are removed from the postfix notations.
- The order of the operators in the postfix notation is the actual order that is used for evaluation.

Source Code:

```
#include <iostream>
#include <stack>
#include <algorithm>

using namespace std;

bool isOperator(char c)
{
    if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^') {
        return true;
    }
    else {
        return false;
    }
}

int precedence(char c)
{
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
}

string InfixToPrefix(stack<char> s, string infix)
{
    string prefix;
    reverse(infix.begin(), infix.end());

    for (int i = 0; i < infix.length(); i++) {
        if (infix[i] == '(') {
            infix[i] = ')';
        }
        else if (infix[i] == ')') {
            infix[i] = '(';
        }
    }
    for (int i = 0; i < infix.length(); i++) {
        if ((infix[i] >= 'a' && infix[i] <= 'z') || (infix[i] >= 'A' && infix[i]
<= 'Z')) {
            prefix += infix[i];
        }
        else if (infix[i] == '(') {
            s.push(infix[i]);
        }
        else if (infix[i] == ')') {
            while ((s.top() != '(') && (!s.empty())) {
                prefix += s.top();
                s.pop();
            }

            if (s.top() == '(') {
                s.pop();
            }
        }
        else if (isOperator(infix[i])) {
            if (s.empty()) {
                s.push(infix[i]);
            }
            else {
                if (precedence(infix[i]) > precedence(s.top())) {
                    s.push(infix[i]);
                }
                else if ((precedence(infix[i]) == precedence(s.top()))
                    && (infix[i] == '^')) {
                    while ((precedence(infix[i]) == precedence(s.top()))
                        && (infix[i] == '^')) {
                        prefix += s.top();
                        s.pop();
                    }
                    s.push(infix[i]);
                }
                else if (precedence(infix[i]) == precedence(s.top())) {
                    s.push(infix[i]);
                }
                else {
                    while ((!s.empty()) && (precedence(infix[i]) < precedence(s.t
op()))) {
                        prefix += s.top();
                        s.pop();
                    }
                    s.push(infix[i]);
                }
            }
        }
    }

    while (!s.empty()) {
        prefix += s.top();
        s.pop();
    }

    reverse(prefix.begin(), prefix.end());
    return prefix;
}

int main()
{
    string infix, prefix;
    cout << "Enter a Infix Expression :" << endl;
    cin >> infix;
    stack<char> stack;
    cout << "INFIX EXPRESSION: " << infix << endl;
    prefix = InfixToPrefix(stack, infix);
    cout << endl << "PREFIX EXPRESSION: " << prefix;

    return 0;
}
```

Output :

```
Test Case-1
Enter a Infix Expression :
ABC+-DE*/
INFIX EXPRESSION: ABC+-DE*/

PREFIX EXPRESSION: -+ABC/*DE
```

29. Write a C++ program to evaluate the given postfix expression.

The postfix notation of the expression is the best notation for the evaluation for the machine as the expression itself determines the order of execution of operators (As the operators are placed in postfix notation based on the operator precedence rules).

In other words, during evaluation the operators need to be performed in the order they appear.

The **algorithm** for evaluating the given postfix expression is as follows:

```
Step-1: Create a stack.

Step-2: For each character c in the postfix expression
        if c is an operand
                push c into stack
        if c is an operator
                pop an element from stack and assign it to A
                pop one more element from stack and assign it to B
                If the stack becomes empty before popping two elements the print
        "Invalid postfix expression." and STOP.
                calculate result = B c A , where c is the operator.
                push result back to stack.

Step-3: After reaching the end of postfix expression, pop and print the element from
the stack as the result.

Step-4: If stack becomes empty before reaching the end of expression then print
"Invalid postfix expression." and also the stack should have one element after reaching
the end of expression else print "Invalid postfix expression."
```

Source Code:

```
#include <iostream>
#include <cmath>
using namespace std;
template <class T>
class Evalpostfix
{
    T *postfix;
    float *stack;
    int max,top;
    public:
        Evalpostfix(int size=10)
        {
            postfix=new T[size];
            stack=new float[size];
            max=size;
            top=-1;
        }
       void read()
        {
            cout<<"Enter the postfix expression : ";
            cin>>postfix;
        }
        void push(float ele)
        {
            stack[++top]=ele;
        }
        float pop()
        {
            return stack[top--];
        }
        void evaluation()
        {
            int val1,val2;
            float result;
            char ch;
            for(int i=0;postfix[i]!='\0';i++)
            {
                ch=postfix[i];
                if(isdigit(ch))
                {
                    push(ch-48);
                }
                else if(ch==' ')
                    continue;
                else
                {
                    val2=pop();
                    val1=pop();
                    switch(ch)
                    {
                        case '+' : result=val1+val2;
                            break;
                        case '-' : result=val1-val2;
                            break;
                        case '*' : result=val1*val2;
                            break;
                        case '/' : result=val1/val2;
                            break;
                        case '%' : result=val1%val2;
                            break;
                        case '^' : result= pow(val1,val2);
                            break;
                        default :
                                exit(0);
                    }
                    push(result);
                }
            }
            cout<<"Result of the postfix expresssion is : "<<pop();
        }
};
int main()
{
    Evalpostfix<char> obj(10);
    obj.read();
    obj.evaluation();
    return 0;
}
```

Output:

Test Case-1	Test Case-2
Enter the postfix expression : 934*8+4/- Result of the postfix expresssion is : 4	Enter the postfix expression : 623+-382/+*2^3+ Result of the postfix expresssion is : 52

30. Implement Circular queue ADT using Arrays.

Circular Queue:

Circular queue avoids the wastage of space in a regular queue implementation using arrays. Programming in C++ and Data Structures

How Circular Queue Works

Circular Queue works by the process of circular increment i.e. when we try to increment any variable and we reach the end of queue, we start from the beginning of queue by modulo division with the queue size. i.e.

if REAR + 1 == 5 (overflow!), REAR = (REAR + 1)%5 = 0 (start of queue)

Queue operations work as follows:

• Two pointers called *FRONT* and *REAR* are used to keep track of the first and last elements in the queue.

• When initializing the queue, we set the value of *FRONT* and *REAR* to -1.

• On enqueing an element, we circularly increase the value of *REAR* index and place the new element in the position pointed to by *REAR*.

• On dequeueing an element, we return the value pointed to by *FRONT* and circularly increase the *FRONT* index.

• Before enqueing, we check if queue is already full.

• Before dequeuing, we check if queue is already empty.

• When enqueing the first element, we set the value of *FRONT* to 0.

• When dequeing the last element, we reset the values of *FRONT* and *REAR* to -1.

Source Code:

```
#include <iostream>
using namespace std;
template <class T>
class CircularQueueADT
{
    T *arr;
    int front,rear;
    int max,len;
    public:
```

```
CircularQueueADT(int size = 10)
{
    arr = new T[size];
    front = rear = 0;
    max = size;
    len=0;
}
bool isEmpty()
{
    return front==rear;
}
bool isFull()
{
    return front == (rear+1)%max;
}
void Enqueue(T ele)
{
    if(isFull())
    {
        cout<<"Queue is Full\n";
    }
    else
    {
        arr[rear]=ele;
        rear = (rear+1)%max;
        len++;
        cout<<"Successfully inserted\n";
    }
}
T Dequeue()
{
    T ele=-1;
    if(isEmpty())
    {
        cout<<"Queue is Empty\n";
    }
    else
    {
        ele = arr[front];
        front = (front+1)%max;
        len--;
    }
    return ele;
}
void display()
```

```
        {
            if(isEmpty())
            {
                cout<<"Queue is empty\n";
            }
            else
            {
                cout<<"Elements in the Queue are : ";
                if (front <rear)
                {
                    for(int i = front;i<rear;i++)
                    {
                    cout<<arr[i]<<" ";
                    }
                    cout<<endl;
                }
                else
                {
                    for(int i = front;i<max-1;i++)
                    {
                        cout<<arr[i]<<" ";
                     }
                    for(int i = 0;i<rear;i++)
                    {
                        cout<<arr[i]<<" ";
                    }
                    cout<<endl;
                }
            }
        }
        void size()
        {
            if(isEmpty())
            {
                cout<<"Circular Queue is Empty\n";
            }
            else
            {
                cout<<"Size of the Circular Queue is : "<<len<<endl;
            }
        }
        ~CircularQueueADT()
        {
            delete arr;
        }
```

```
};
int main()
{
    CircularQueueADT <int> obj(10);
    int ele,ch;
    cout<<"\n1.Enqueue\n2.Dequeue\n3.display\n4.size\n5.exit\n";
    while(1)
    {
        cout<<"Enter the choice : ";
        cin>>ch;
        switch(ch)
        {
            case 1:
                cout<<"Enter the element : ";
                cin>>ele;
                obj.Enqueue(ele);
                break;
            case 2:
                cout<<"Deleted element is : "<<obj.Dequeue()<<endl;
                break;
            case 3:
                obj.display();
                break;
            case 4:
                obj.size();
                break;
            case 5:
                cout<<"Program Exited";
                exit(0);
            default :
                    cout<<"Invalid choice\n";
        }
    }
    return 0;
}
```

Output:

Test Case-1	Test Case-2
1.Enqueue 2.Dequeue 3.display 4.size 5.exit Enter the choice : 1	1.Enqueue 2.Dequeue 3.display 4.size 5.exit Enter the choice : 1

Enter the element : 20 Successfully inserted Enter the choice : 1 Enter the element : 30 Successfully inserted Enter the choice : 4 Size of the Circular Queue is : 2 Enter the choice : 3 Elements in the Queue are : 20 30 Enter the choice : 2 Deleted element is : 20 Enter the choice : 30 Invalid choice Enter the choice : 5 Program Exited	Enter the element : 10 Successfully inserted Enter the choice : 1 Enter the element : 30 Successfully inserted Enter the choice : 1 Enter the element : 50 Successfully inserted Enter the choice : 2 Deleted element is : 10 Enter the choice : 2 Deleted element is : 30 Enter the choice : 4 Size of the Circular Queue is : 1 Enter the choice : 5 Program Exited

31. Implement Double Ended queue ADT using Arrays.

A Deque or Double Ended Queue is a generalized version of Queue data structure that allows insert and delete at both ends.

A Deque can be used as both queue and a stack.

Operations of Double ended queue

The operations that are performed on a double ended queue are :

- inject() - This operation inserts an element to the rear of the double ended queue.
- eject() - This operation deletes an element from the rear of the double ended queue.
- push() - This operation inserts an element to the front of the double ended queue.
- pop() - This operation deletes an element from the front of the double ended queue.

Source Code:

```
//31. Implement Double Ended queue ADT using Arrays.
#include <iostream>
using namespace std;
template <class T>
class DEQueueADT
{
    T *q;
    int front,rear,maxsize;
    public:
        DEQueueADT(int size = 10)
        {
            maxsize = size;
            q = new T[maxsize];
            front=rear=0;
        }
        bool isEmpty()
        {
            if(front == rear)
                return true;
            else
                return false;
        }
        bool isFull()
        {
            if(front == 0 && rear == maxsize)
                return true;
            else
                return false;
        }
        void enqueueEnd(T ele)
        {
            if(isFull())
                cout<<"Queue is Full\n";
            else
            {
                if(rear!=maxsize)
                {
                    q[rear++]=ele;
                }
                else
                {
                    front--;
                    for(int i = front;i<rear-1;i++)
                    {
                        q[i]=q[i+1];
                    }
                    q[rear-1]=ele;
                    rear++;
                }
                cout<<"Successfully element is inserted at end of the queue\n";
            }
        }
        void enqueueBeg(T ele)
        {
            if(isFull())
                cout<<"Queue is Full\n";
            else
            {
                if(front!=0)
                {
                    q[--front]=ele;
                }
                else
                {
                    if(front==rear)
                    {
                        q[rear++]=ele;
                    }
                    else
                    {
                        for(int i = rear ; i>front;i--)
                        {
                            q[i]=q[i-1];
                        }
                        rear++;
                        q[front]=ele;
                    }
                }
                cout<<"Successfully element is inserted at begin\n";
            }
        }
        T dequeueBeg()
        {
            T ele=-1;
            if(isEmpty())
            {
                cout<<"Queue is Empty\n";
            }
            else
            {
                ele=q[front++];
            }
            return ele;
        }
        T dequeueEnd()
        {
            T ele=-1;
            if(isEmpty())
            {
                cout<<"Queue is Empty\n";
            }
            else
            {
                ele=q[--rear];
            }
            return ele;
        }
        void display()
        {
            if(isEmpty())
            {
                cout<<"Queue is Empty\n";
            }
            else
            {
                cout<<"Elements in the Double Ended queue are : ";
                for(int i = front; i<rear;i++)
```

```
                {
                    cout<<q[i]<<" ";
                }
                cout<<endl;
            }
        }
        ~DEQueueADT()
        {
            delete q;
        }
};
int main()
{
    int n;
    cout<<"Enter the size of Double Ended queue you want : ";
    cin>>n;
    DEQueueADT<int> obj(n);
    int ele,ch;
    cout<<"1.enqueueEnd\n2.enqueueBeg\n3.dequeueBeg\n4.dequeueEnd\n5.display\n6.e
xit\n";
    while(1)
    {
        cout<<"Enter the choice : ";
        cin>>ch;
        switch(ch)
        {
            case 1 :
                    cout<<"Enter the element : ";
                    cin>>ele;
                    obj.enqueueEnd(ele);
                    break;
            case 2 :
                    cout<<"Enter the element : ";
                    cin>>ele;
                    obj.enqueueBeg(ele);
                    break;
            case 3 :
                    cout<<"Deleted element at Beginning is : "<<obj.dequeueBeg()<
<endl;
                    break;
            case 4 :
                    cout<<"Deleted element at end is : "<<obj.dequeueEnd()<<endl;
                    break;
            case 5 :
                    obj.display();
                    break;
            case 6 :
                    cout<<"Program is ended\n";
                    exit(0);
            default :
                    cout<<"INVALID CHOICE\n";
        }
    }
    return 0;
}
```

Output:

Test Case-1	Test Case-2
1.enqueueEnd 2.enqueueBeg 3.dequeueBeg 4.dequeueEnd 5.display 6.exit Enter the choice : 1 Enter the element : 10 Successfully element is inserted at end of the queue Enter the choice : 2 Enter the element : 20 Successfully element is inserted at begin Enter the choice : 5 Elements in the Double Ended queue are : 20 10 Enter the choice : 3 Deleted element at Beginning is : 20 Enter the choice : 5 Elements in the Double Ended queue are : 10 Enter the choice : 6 Program is ended	1.enqueueEnd 2.enqueueBeg 3.dequeueBeg 4.dequeueEnd 5.display 6.exit Enter the choice : 5 Queue is Empty Enter the choice : 1 Enter the element : 60 Successfully element is inserted at end of the queue Enter the choice : 2 Enter the element : 10 Successfully element is inserted at begin Enter the choice : 4 Deleted element at end is : 60 Enter the choice : 5 Elements in the Double Ended queue are : 10 Enter the choice : 6 Program is ended

32. Implement Max-Heap tree and its operations.

- Heap Sort is a comparison based sorting technique based on a data structure called Binary Heap.
- Heap Sort is similar to a selection sort where we find the maximum element and place the maximum element at the end and the same process is repeated for the remaining elements.
- Binary Heap A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.
- A complete binary tree is said to be a Binary Heap if all the elements of the tree are placed such that the value of the parent node is greater (or smaller) than that the values of its children nodes.
- If the values of the parents are always greater than the child nodes, then the binary heap is called as Max Heap.

Source Code:

```
#include<iostream>
using namespace std;
template<class T>
class maxheap
{
    T *heap;
    int maxsize,count;
    public:
        maxheap(int size=10)
        {
            maxsize=size;
            heap=new T[maxsize];
            count=0;
        }
        void push(T ele)
        {
            count++;
            int pos=count;
            if(pos<=maxsize)
            {
                while(heap[pos/2]<ele&&pos>1)
                {
                    heap[pos]=heap[pos/2];
                    pos=pos/2;
                }
            heap[pos]=ele;
            }
            else
                cout<<"\n heap is full";
        }
        T pop()
        {
            T x=0;
            if(count!=0)
            {
                x=heap[1];
                int i=1;
                int j=2*i;
                while((heap[count]<heap[j]||heap[count]<heap[j+1])&&j<count)
                {
                    if(heap[j]>heap[j+1])
                    {
                        heap[i]=heap[j];i=j;
                    }
                    else
                    {
                        heap[i]=heap[j+1];i=j+1;
                    }
                    j=2*i;
                }
                heap[i]=heap[count];
                count--;
            }
            else cout<<"\nHeap is empty";
                return x;
        }
        void display()
        {
            cout<<"\n The heap is:";
            for(int i=1;i<=count;i++)
                cout<<" "<<heap[i];
        }
};
int main()
{
    maxheap<int> h;
    int n;
    int ch;
    while(1)
    {
        cout<<"\n 1.Insert 2.Delete 3.Exit\n Enter Ur choice:";
        cin>>ch;
        switch(ch)
        {
            case 1:
                    cout<<"\n Enter an element:";
                    cin>>n;
                    h.push(n);
                    break;
            case 2:
                    cout<<"\n The deleted element is : "<<h.pop();
                    break;
            case 3:
                    return 0;
            default:
                    cout<<"\n Wrong option \n try again .....";
        }
        h.display();
    }
    return 0;
}
```

Output:

Test Case-1	Test Case-2
1.Insert 2.Delete 3.Exit	1.Insert 2.Delete 3.Exit
Enter Ur choice:1	Enter Ur choice:1
Enter an element:1	Enter an element:2
The heap is: 1	The heap is: 2
1.Insert 2.Delete 3.Exit	1.Insert 2.Delete 3.Exit
Enter Ur choice:5	Enter Ur choice:1
Wrong option	Enter an element:6
try again	
The heap is: 1	The heap is: 6 2
1.Insert 2.Delete 3.Exit	1.Insert 2.Delete 3.Exit
Enter Ur choice:1	Enter Ur choice:2
Enter an element:6	The deleted element is : 6
	The heap is: 2
The heap is: 6 1	1.Insert 2.Delete 3.Exit
1.Insert 2.Delete 3.Exit	Enter Ur choice:3
Enter Ur choice:2	

The deleted element is : 6 The heap is: 1 1.Insert 2.Delete 3.Exit Enter Ur choice:3	

33. Implement Min-Heap tree and its operations.

- Heap Sort is a comparison based sorting technique based on a data structure called Binary Heap.
- Heap Sort is similar to a selection sort where we find the maximum element and place the maximum element at the end and the same process is repeated for the remaining elements.
- Binary Heap A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.
- A complete binary tree is said to be a Binary Heap if all the elements of the tree are placed such that the value of the parent node is greater (or smaller) than that the values of its children nodes.
- If the values of the parents are always smaller than the child nodes, then the binary heap is called as Min Heap.

Source Code:

```
#include<iostream>

using namespace std;
template<class T>
class minheap
{
    T *heap;
    int maxsize,count;
    public:
        minheap(int size=10)
        {
            maxsize=size;
            heap=new T[maxsize];
            count=0;
        }
        void push(T ele)
        {
            count++;
            int pos=count;
            if(pos<=maxsize)
            {
                while(heap[pos/2]>ele&&pos>1)
                {
                    heap[pos]=heap[pos/2];
                    pos=pos/2;
                }
                heap[pos]=ele;
            }
            else
                cout<<"\n heap is full";
        }
        T pop()
        {
            T x=0;
            if(count!=0)
            {
                x=heap[1];
                int i=1;
                int j=2*i;
                while((heap[count]>heap[j]||heap[count]>heap[j+1])&&j<count)
                {
                    if(heap[j]<heap[j+1])
                    {
                        heap[i]=heap[j];i=j;
                    }
                    else
                    {
                        heap[i]=heap[j+1];i=j+1;
                    }
                    j=2*i;
                }
                heap[i]=heap[count];
                count--;
            }
            else
                cout<<"\nHeap is empty";
            return x;
        }
        void display()
        {
            cout<<"\n The heap is:";
            for(int i=1;i<=count;i++)
                cout<<" "<<heap[i];
        }
};
int main()
{
    minheap<int> h;
    int n;
    int ch;
    while(1)
    {
        cout<<"\n 1.Insert 2.Delete 3.Exit\n Enter Ur choice:";
        cin>>ch;
        switch(ch)
        {
            case 1:
                    cout<<"\n Enter an element:";
                    cin>>n;
                    h.push(n);
                    break;

            case 2:
                    cout<<"\n The deleted element is : "<<h.pop();
                    break;
            case 3:
                    return 0;
            default:
                    cout<<"\n Wrong option \n try again .....";
        }
        h.display();
    }
    return 0;
}
```

Output:

Test Case-1

```
1.Insert 2.Delete 3.Exit
 Enter Ur choice:1

 Enter an element:25

 The heap is: 25
 1.Insert 2.Delete 3.Exit
 Enter Ur choice:1

 Enter an element:36

 The heap is: 36 25
 1.Insert 2.Delete 3.Exit
```

```
Enter Ur choice:1

Enter an element:65

The heap is: 65 25 36
1.Insert 2.Delete 3.Exit
Enter Ur choice:3
```

34. Implement Ascending Priority Queue.

A **priority queue** is an extension of a normal queue.

An ascending **priority queue** has the following properties:

- Every element that is present in the queue has a priority associated with it.
- An element with low priority is dequeued before an element with high priority, no matter where it is present in the queue.
- If two elements have the same priority, the element that came first into the queue is dequeued first. (i.e We follow FIFO principle)

The queue is created so that the highest priority element is always at the beginning of the queue.
The queue is arranged in ascending order of elements based on their priority.
This is done so that we can remove the lowest priority element in O(1) time.
When inserting an element, the queue is traversed to find a proper position and then the element is inserted so that the overall order of the priority queue is maintained.

Source Code:

```
#include<iostream>
using namespace std;
template<class T>
class priority
{
    T *q;
    int maxsize,count;
    public:
        priority(int size=10)
        {
            q=new T[size];
            maxsize=size;
            count=0;
        }
        void push(T ele)
        {
            if(count<maxsize)
            {
                int i=count;
                count++;
                while(q[i-1]<ele&&i>0)
                {
                    q[i]=q[i-1];
                    i--;
                }
                q[i]=ele;
            }
            else
                cout<<"\n The queue is full";
        }
        T pop()
        {
            T x=0;
            if(count!=0)
            {
                x=q[0];
                for(int i=1;i<count;i++)
                q[i-1]=q[i];
                count--;
            }
            else
                cout<<"\nQ is empty";
        return x;
        }
        void display()
        {
            cout<<"\n Q is : ";
            for(int i=0;i<count;i++)
                cout<<" "<<q[i];
        }
};
int main()
{
    priority<int> p;
    int n;
    int ch;
    while(1)
    {
    cout<<"\n 1.Insert 2.Delete 3.Exit\n Enter Ur choice:";
    cin>>ch;
    switch(ch)
    {
        case 1:
                cout<<"\n Enter an element:";
                cin>>n;
                p.push(n);
                break;

            case 2:
                cout<<"\n The deleted element is : "<<p.pop();
                break;
            case 3:
                return 0;
            default:
                cout<<"\n Wrong option \n try again .....";
        }
        p.display();
        }
        return 0;
}
```

Output:

Test Case-1	Test Case-2
1.Insert 2.Delete 3.Exit	1.Insert 2.Delete 3.Exit
Enter Ur choice: 1	Enter Ur choice: 1
Enter an element:2	Enter an element:2
Q is : 2	Q is : 5
1.Insert 2.Delete 3.Exit	1.Insert 2.Delete 3.Exit
Enter Ur choice:1	Enter Ur choice:1
Enter an element:5	Enter an element:1
Q is : 5 2	Q is : 1 5
1.Insert 2.Delete 3.Exit	1.Insert 2.Delete 3.Exit
Enter Ur choice:2	Enter Ur choice:2
The deleted element is : 5	The deleted element is : 1
Q is : 2	Q is : 5
1.Insert 2.Delete 3.Exit	1.Insert 2.Delete 3.Exit
Enter Ur choice:3	Enter Ur choice:3

35. Implement Descending Priority Queue.

A **priority queue** is an extension of a normal queue.

A **Descending priority queue** has the following properties:

- Every element that is present in the queue has a priority associated with it.
- An element with high priority is dequeued before an element with low priority, no matter where it is present in the queue.
- If two elements have the same priority, the element that came first into the queue is dequeued first. (i.e We follow FIFO principle)

The queue is created so that the highest priority element is always at the beginning of the queue. The queue is arranged in descending order of elements based on their priority.
This is done so that we can remove the highest priority element in O(1) time.
When inserting an element, the queue is traversed to find a proper position and then the element is inserted so that the overall order of the priority queue is maintained.

Source Code:

```
#include<iostream>
using namespace std;
template<class T>
class priority
{
    T *q;
    int maxsize,count;
    public:
        priority(int size=10)
        {
            q=new T[size];
            maxsize=size;
            count=0;
        }
        void push(T ele)
        {
            if(count<maxsize)
            {
                int i=count;
                count++;
                while(q[i-1]>ele&&i>0)
                {
                    q[i]=q[i-1];
                    i--;
                }
                q[i]=ele;
            }
            else
                cout<<"\n The queue is full";
        }
        T pop()
        {
            T x=0;
            if(count!=0)
            {
                x=q[0];
                for(int i=1;i<count;i++)
                q[i-1]=q[i];
                count--;
            }
            else
                cout<<"\nQ is empty";
            return x;
        }
        void display()
        {
            cout<<"\n Q is : ";
            for(int i=0;i<count;i++)
                cout<<" "<<q[i];
        }
};
int main()
{
    priority<int> p;
    int n;
    int ch;
    while(1)
    {
        cout<<"\n 1.Insert 2.Delete 3.Exit\n Enter Ur choice:";
        cin>>ch;
        switch(ch)
        {
            case 1:
                cout<<"\n Enter an element:";
                cin>>n;
                p.push(n);
                break;

            case 2:
                cout<<"\n The deleted element is : "<<p.pop();
                break;
            case 3:
                return 0;
            default:
                cout<<"\n Wrong option \n try again .....";
        }
        p.display();
    }
    return 0;
}
```

Output:

Test Case-1

```
1.Insert 2.Delete 3.Exit
 Enter Ur choice:1

 Enter an element:25

 Q is :  25
 1.Insert 2.Delete 3.Exit
 Enter Ur choice:2

 The deleted element is : 25
 Q is :
 1.Insert 2.Delete 3.Exit
 Enter Ur choice:1

 Enter an element:32

 Q is :  32
 1.Insert 2.Delete 3.Exit
 Enter Ur choice:1
 Enter an element:63

 Q is :  32 63
 1.Insert 2.Delete 3.Exit
 Enter Ur choice:1

 Enter an element:52

 Q is :  32 52 63
 1.Insert 2.Delete 3.Exit
 Enter Ur choice:3
```

36. Write a C++ program to implement the following operations on Binary Tree

- **Insert**
- **Delete**
- **Search**
- **Display**

In general a tree node can contain any number of children. Binary Tree is a special kind of tree in which every node can have a maximum of 2 child nodes. One child is called as left child and the other is called as right child.

In other words, every node in a binary tree can have either 0 or 1 or 2 children.

Types of binary trees

There are different types of binary trees. They are :

1. Full binary tree
2. Complete binary tree
3. Perfect binary tree
4. Extended binary tree
5. Balanced binary tree

Full binary tree

A binary tree is said to be a full binary tree if every node in the binary tree has either 0 or 2 nodes. In other word a binary tree is said to be a full binary tree if all nodes expect leaf nodes has two children. Full binary tree is also called as strict binary tree or proper binary tree or 2- tree. Image below shows an example of full binary tree.

Complete binary tree

A binary tree is complete binary tree if all levels are completely filled except possibly the last level and the last level has all keys as left as possible.

Perfect binary tree

In a perfect binary tree all the nodes must have exactly two children and at every level of perfect binary tree there must be 2level number of nodes. For example at level 2 there must be 22 = 4 nodes and at level 3 there must be 23 = 8 nodes.

In other words, A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called perfect binary tree.

A binary tree can be represented in two methods. They are :

1. Array representation.
2. Linked list representation.

1. Array representation

In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree. The root node of the tree is always put into index 0. For any parent placed at location K, the left child would be present at 2*K+1 location and the right child would be present at 2*K+2 location.

2. Linked list representation

In this representation, we use list with one type of node which consists of three fields namely left reference field, data field and right reference field.

In this representation, every node's data field stores the actual value of tree node. If that node has left child, then left reference field stores the address of that left child node otherwise that field stores NULL. If that node has right child then right reference field stores the address of right child node otherwise that field stores NULL.

Traversing is a way of visiting or displaying all the data elements of the data structure.

Generally the linear data structures like Arrays, Stacks, Queues, Lists have only one way of traversing.

Displaying (or) visiting order of all the nodes in a binary tree is called as "Binary Tree Traversal".

Trees can be **traversed** in different ways

While visiting all the nodes of the binary tree, we follow a particular order in which we visit the nodes, it depends on the traversal method we follow. There are three traversal methods for binary tree. They are :

1. In- order traversal.
2. Pre - order traversal.
3. Post - order traversal.

Source Code:

```
#include<iostream>
using namespace std;
template<class T>
class Node
{
    public:
        T data;
        Node<T> *left,*right;
};
template<class T>
class bintree
{
    Node<T> *root;
    public:
        bool search(T ele)
        {
            bool flag=false;
            Search1(root,ele,flag);
            return flag;
        }
```

```
void Search1(Node<T> *rt,T ele,bool &exist)
{
    if(rt!=NULL)
    {
        if(rt->data==ele)
        {
            exist=true;
            return;
        }
    Search1(rt->left,ele,exist);
    Search1(rt->right,ele,exist);
    }
}
void insert(T ele,T parent)
{
    if(root==NULL)
    {
        Node<T> *temp=new Node<T>;
        temp->data=ele;
        temp->left=temp->right=NULL;
        root=temp;
        return;
    }
    else
    {
        bool h=false;
        Search(root,parent,h);
        if(h)
        {
            ins(root,ele,parent);
        }
        else
            cout<<"\nParent not found";
    }
}
void ins(Node<T> *rt,T ele,T parent)
{
    if(rt!=NULL)
    {
        if(parent!=(rt)->data)
        {
            ins(&(rt)->left,ele,parent);
            ins(&(rt)->right,ele,parent);
        }
        else
```

```
            {
                if((rt)->left==NULL||(rt)->right==NULL)
                {
                    Node<T> *temp;
                    temp->data=ele;
                    temp->left=temp->right=NULL;
                if((rt)->left==NULL)
                {
                    (rt)->left=temp;
                }
                else
                {
                zzif((rt)->right==NULL)
                (rt)->right=temp;
            }
        }
        else
        cout<<"\n ****Insertion not possible*****";
        return;
        }
        }
    }
        void deletion(T ele)
        {
        bool h;
        Search(root,ele,h);
        if(h)
        {
        del(root,ele);
        }
        else
        cout<<"\n ****Element not found*****";
    }
    void del(Node<T> *rt,T ele)
    {
        if((rt)!=NULL)
        {
            if((rt)->data!=ele)
            {
                del(&(rt)->left,ele);
                del(&(rt)->right,ele);
            }
            else
            {
            if((rt)->left!=NULL&&(rt)->right!=NULL)
```

```
                {
                    Node<T> *temp=(rt)->right,*pt=NULL;
                    while(temp->left)
                    {
                        pt=temp;
                        temp=temp->left;
                    }
                    if(pt!=NULL)
                    {
                        pt->left=temp->right;
                        temp->left=(rt)->left;
                        temp->right=(rt)->right;
                        (rt)=temp;
                    }
                else
                {
                    (rt)->right->left=(rt)->left;
                    (rt)=(rt)->right;
                }
            }
            else
            if((rt)->left==NULL&&(rt)->right!=NULL)
            (rt)=(rt)->right;
            else
            if((rt)->left!=NULL&&(rt)->right==NULL)
            (rt)=(rt)->left;
            else (rt)=NULL;
            }
            }
        }
        void display()
        {
            cout<<"\n The binary tree is:";
            cout<<"\nInorder :";
            inorder(root);
            cout<<"\nPreorder :";
            preorder(root);
        }
        void inorder(Node<T> *rt)
        {
            if(rt!=NULL)
            {
                inorder(rt->left);
                cout<<" "<<rt->data;
                inorder(rt->right);
            }
        }
        void preorder(Node<T> *rt)
        {
            if(rt!=NULL)
            {
                cout<<" "<<rt->data;
                preorder(rt->left);
                preorder(rt->right);
            }
        }
};
int main()
{
    bintree<int> b;
    int ch,ele,pt;
    while(1)
    {
        cout<<"\n 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit";
        cout<<"\n Enter ur choice:";
        cin>>ch;
        switch(ch)
        {
            case 1:
                    cout<<"\n Enter element value:";
                    cin>>ele;
                    cout<<"\n Enter Parent element:";
                    cin>>pt;
                    b.insert(ele,pt);
                    break;
            case 2:
                    cout<<"\n Enter element value to delete:";
                    cin>>ele;
                    b.deletion(ele);
                    break;
            case 3:
                    cout<<endl;
                    b.display();
                    break;
            case 4:
                    cout<<"\n Enter element value to search:";
                    cin>>ele;
                    if(b.search(ele))
                        cout<<"\n Element found";
                    else
```

```
                        cout<<"\n Element not found";
                    break;
            case 5:
                    return 0;
            default:
                    cout<<" \n Invalid choice";
        }
        b.display();
        cout<<endl;
    }
    return 0;
}
```

Output:

Test Case-1	Test Case-2
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:2 Enter element value to delete:12 ****Element not found***** The binary tree is: Inorder : Preorder : 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:1 Enter element value:44 Enter Parent element:1 The binary tree is: Inorder : 44 Preorder : 44 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:2 Enter element value to delete:55 ****Element not found***** The binary tree is: Inorder : 44 Preorder : 44 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:2 Enter element value to delete:44 The binary tree is: Inorder : Preorder : 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:1	1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:2 Enter element value to delete:12 ****Element not found***** The binary tree is: Inorder : Preorder : 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:1 Enter element value:44 Enter Parent element:1 The binary tree is: Inorder : 44 Preorder : 44 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:2 Enter element value to delete:55 ****Element not found***** The binary tree is: Inorder : 44 Preorder : 44 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:2 Enter element value to delete:44 The binary tree is: Inorder : Preorder : 1.Insertion 2.Deletion 3.Display 4.Search 5.Exit Enter ur choice:1

```
Enter element value:78
Enter Parent element:2
The binary tree is:
Inorder : 78
Preorder : 78
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:1
Enter element value:89
Enter Parent element:78
The binary tree is:
Inorder : 89 78
Preorder : 78 89
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:1
Enter element value:90
Enter Parent element:78
The binary tree is:
Inorder : 89 78 90
Preorder : 78 89 90
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:1
Enter element value:34
Enter Parent element:90
The binary tree is:
Inorder : 89 78 34 90
Preorder : 78 89 90 34
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:1
Enter element value:35
Enter Parent element:90
The binary tree is:
Inorder : 89 78 34 90 35
Preorder : 78 89 90 34 35
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:4
Enter element value to search:23
Element not found
The binary tree is:
Inorder : 89 78 34 90 35
Preorder : 78 89 90 34 35
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:4
Enter element value to search:35
Element found
The binary tree is:
Inorder : 89 78 34 90 35
Preorder : 78 89 90 34 35
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:3
The binary tree is:
Inorder : 89 78 34 90 35
```

```
Enter element value:78
Enter Parent element:2
The binary tree is:
Inorder : 78
Preorder : 78
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:1
Enter element value:89
Enter Parent element:78
The binary tree is:
Inorder : 89 78
Preorder : 78 89
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:1
Enter element value:90
Enter Parent element:78
The binary tree is:
Inorder : 89 78 90
Preorder : 78 89 90
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:1
Enter element value:34
Enter Parent element:90
The binary tree is:
Inorder : 89 78 34 90
Preorder : 78 89 90 34
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:1
Enter element value:35
Enter Parent element:90
The binary tree is:
Inorder : 89 78 34 90 35
Preorder : 78 89 90 34 35
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:4
Enter element value to search:23
Element not found
The binary tree is:
Inorder : 89 78 34 90 35
Preorder : 78 89 90 34 35
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:4
Enter element value to search:35
Element found
The binary tree is:
Inorder : 89 78 34 90 35
Preorder : 78 89 90 34 35
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:3
The binary tree is:
Inorder : 89 78 34 90 35
```

Preorder : 78 89 90 34 35	Preorder : 78 89 90 34 35
The binary tree is:	The binary tree is:
Inorder : 89 78 34 90 35	Inorder : 89 78 34 90 35
Preorder : 78 89 90 34 35	Preorder : 78 89 90 34 35
1.Insertion 2.Deletion 3.Display 4.Search 5.Exit	1.Insertion 2.Deletion 3.Display 4.Search 5.Exit
Enter ur choice:5	Enter ur choice:5

37. Write a C++ program to implement the following operations on Binary Search Tree

- **Insert**
- **Delete**
- **Search**
- **Display**

In a binary tree, every node can have maximum of two children but there is no restriction in the order of nodes based on their values. In binary tree, the elements are arranged as they arrive to the tree, from top to bottom and left to right.

In Binary Search Tree (BST), the worst case timing complexity of all the the three operations is O(log n) , where 'n' is the number of nodes present in the tree.

A Binary Search Tree, is a binary tree data structure which has the following properties:

- The left sub tree of a node contains only nodes with keys lesser than the node's key.
- The right sub tree of a node contains only nodes with keys greater than the node's key.
- The left and right sub tree each must also be a binary search tree.
- There must be no duplicate nodes.

Source Code:

```
# include <iostream>
using namespace std;
struct node
{
    int info;
    struct node *left;
    struct node *right;
}*root;
class BST
{
    public:
        void find(int, node **, node **);
        void insert(node *, node *);
        void del(int);
        void case_a(node *,node *);
        void case_b(node *,node *);
        void case_c(node *,node *);
        void preorder(node *);
        void inorder(node *);
        void postorder(node *);
        void display(node *, int);
        BST()
        {
            root = NULL;
        }
};
int main()
{
    int choice, num;
    BST bst;
    node *temp;
    while (1)
    {
        cout<<"-----------------"<<endl;
        cout<<"Operations on BST"<<endl;
        cout<<"-----------------"<<endl;
        cout<<"1.Insert Element "<<endl;
        cout<<"2.Delete Element "<<endl;
        cout<<"3.Inorder Traversal"<<endl;
        cout<<"4.Preorder Traversal"<<endl;
        cout<<"5.Postorder Traversal"<<endl;
        cout<<"6.Display"<<endl;
        cout<<"7.Quit"<<endl;
        cout<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            temp = new node;
            cout<<"Enter the number to be inserted : ";
            cin>>temp->info;
            bst.insert(root, temp);
        case 2:
            if (root == NULL)
            {
                cout<<"Tree is empty, nothing to delete"<<endl;
                continue;
            }
            cout<<"Enter the number to be deleted : ";
            cin>>num;
            bst.del(num);
            break;
        case 3:
            cout<<"Inorder Traversal of BST:"<<endl;
            bst.inorder(root);
            cout<<endl;
            break;
        case 4:
            cout<<"Preorder Traversal of BST:"<<endl;
            bst.preorder(root);
            cout<<endl;
            break;
        case 5:
            cout<<"Postorder Traversal of BST:"<<endl;
            bst.postorder(root);
            cout<<endl;
            break;
        case 6:
            cout<<"Display BST:"<<endl;
            bst.display(root,1);
            cout<<endl;
            break;
        case 7:
            exit(1);
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
}
void BST::find(int item, node **par, node **loc)
{
    node *ptr, *ptrsave;
    if (root == NULL)
    {
        *loc = NULL;
        *par = NULL;
        return;
    }
    if (item == root->info)
    {
        *loc = root;
        *par = NULL;
        return;
    }
    if (item < root->info)
        ptr = root->left;
    else
        ptr = root->right;
    ptrsave = root;
    while (ptr != NULL)
    {
        if (item == ptr->info)
        {
            *loc = ptr;
            *par = ptrsave;
            return;
        }
        ptrsave = ptr;
        if (item < ptr->info)
            ptr = ptr->left;
else
   ptr = ptr->right;
    }
    *loc = NULL;
    *par = ptrsave;
}
void BST::insert(node *tree, node *newnode)
{
    if (root == NULL)
    {
        root = new node;
        root->info = newnode->info;
        root->left = NULL;
        root->right = NULL;
        cout<<"Root Node is Added"<<endl;
        return;
    }
    if (tree->info == newnode->info)
    {
        cout<<"Element already in the tree"<<endl;
        return;
    }
    if (tree->info > newnode->info)
    {
        if (tree->left != NULL)
        {
            insert(tree->left, newnode);
}
else
{
            tree->left = newnode;
            (tree->left)->left = NULL;
            (tree->left)->right = NULL;
            cout<<"Node Added To Left"<<endl;
            return;
        }
    }
    else
    {
        if (tree->right != NULL)
        {
            insert(tree->right, newnode);
        }
        else
        {
            tree->right = newnode;
            (tree->right)->left = NULL;
            (tree->right)->right = NULL;
            cout<<"Node Added To Right"<<endl;
            return;
        }
    }
}
void BST::del(int item)
{
    node *parent, *location;
    if (root == NULL)
    {
        cout<<"Tree empty"<<endl;
        return;
    }
    find(item, &parent, &location);
    if (location == NULL)
    {
        cout<<"Item not present in tree"<<endl;
        return;
    }
    if (location->left == NULL && location->right == NULL)
        case_a(parent, location);
    if (location->left != NULL && location->right == NULL)
        case_b(parent, location);
    if (location->left == NULL && location->right != NULL)
        case_b(parent, location);
    if (location->left != NULL && location->right != NULL)
        case_c(parent, location);
    free(location);
}
void BST::case_a(node *par, node *loc )
```

```
{
    if (par == NULL)
    {
        root = NULL;
    }
    else
    {
        if (loc == par->left)
            par->left = NULL;
        else
            par->right = NULL;
    }
}
void BST::case_b(node *par, node *loc)
{
    node *child;
    if (loc->left != NULL)
        child = loc->left;
    else
        child = loc->right;
    if (par == NULL)
    {
        root = child;
    }
    else
    {
        if (loc == par->left)
            par->left = child;
        else
            par->right = child;
    }
}
void BST::case_c(node *par, node *loc)
{
    node *ptr, *ptrsave, *suc, *parsuc;
    ptrsave = loc;
    ptr = loc->right;
    while (ptr->left != NULL)
    {
        ptrsave = ptr;
        ptr = ptr->left;
    }
    suc = ptr;
    parsuc = ptrsave;
    if (suc->left == NULL && suc->right == NULL)
```

```
        case_a(parsuc, suc);
    else
        case_b(parsuc, suc);
    if (par == NULL)
    {
        root = suc;
    }
    else
    {
        if (loc == par->left)
            par->left = suc;
        else
            par->right = suc;
    }
    suc->left = loc->left;
    suc->right = loc->right;
}
void BST::preorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty"<<endl;
        return;
    }
    if (ptr != NULL)
    {
        cout<<ptr->info<<" ";
        preorder(ptr->left);
        preorder(ptr->right);
    }
}
void BST::inorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty"<<endl;
        return;
    }
    if (ptr != NULL)
    {
        inorder(ptr->left);
        cout<<ptr->info<<" ";
        inorder(ptr->right);
    }
}
```

```
void BST::postorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty"<<endl;
        return;
    }
    if (ptr != NULL)
    {
        postorder(ptr->left);
        postorder(ptr->right);
        cout<<ptr->info<<" ";
    }
}
void BST::display(node *ptr, int level)
{
    int i;
    if (ptr != NULL)
    {
        display(ptr->right, level+1);
        cout<<endl;
        if (ptr == root)
            cout<<"Root->: ";
        else
        {
            for (i = 0;i < level;i++)
                cout<<" ";
}
        cout<<ptr->info;
        display(ptr->left, level+1);
    }
}
```

Output:

```
Test Case-1
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
```

```
7.Quit
Enter your choice : 6
Display BST:

-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 85
Root Node is Added
Enter the number to be deleted : 0
Item not present in tree
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 11
Wrong choice
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 25
Node Added To Left
Enter the number to be deleted : 0
Item not present in tree
```

```
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 3
Inorder Traversal of BST:
25 85
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 4
Preorder Traversal of BST:
85 25
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 5
Postorder Traversal of BST:
25 85
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
```

```
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 7
```

38. Implement Quadratic Probing.

Hashing is an approach in which the timing complexity of insert, delete and search operations is not dependent on the number of elements present in the data structure. In fact, all the operation like insert, delete and search operation have constant time complexity i.e O(1) when hashing approach is used.

In the hashing approach we will use a table called **Hash Table** to store the data. The size of the hash table is less than the range of the actual data. So the data values are inserted into the hash table based on the hash key value. Hash key value is the index of the hash table at which the actual data is stored. The Hash key value provides the mapping of the actual data and the index of the hash table.

Hashing function is a function which takes the key i.e the actual data to be inserted as input and gives the hash key value as output which is the index of the hash table at which the key is stored. Hashing function is always designed in such a way that the output of the hashing function is always within the range of the index of the hash table.

Choosing a good hashing function, h(k), is essential. h(k) should distribute the data as uniformly as possible to the indices of the hash table. h(k) is a function that maps universe of keys U to some index between 0 to S where S is the size of the hash table. When inserting, searching, or deleting a key k, h(k) is calculated and the h(k)th slot is checked to add, look for, or remove the key. Any hashing function h(k) should have the following properties

- Satisfy (approximately) the assumption of simple uniform hashing: each key is equally likely to hash to any of the S slots. The hash function shouldn’t bias towards particular slots.
- Quick to be calculated, should ideally have O(1) run-time.
- Should be deterministic i.e. h(k) should always return the same value for a given k.

Collision

If all keys hash to different slots of hash table, then the hash table operations are as fast as computing the hash function and changing or inspecting the value of an array element, which is O(1) runtime. However, this is not always possible. If the number of possible keys is greater than the number of slots in the hash table (which is the case most of the times), then there must be some keys that hash into the same slot, in other words a collision

During collision, a newly inserted key maps to an already occupied slot in hash table and must be handled using some collision handling technique.

There are mainly two methods **to handle collision**:

- **Open Addressing** (Closed Hashing) with linear probing or quadratic probing or double hashing.
- **Separate Chaining** (Open Hashing).

Quadratic probing

Quadratic probing is similar to linear probing and the only difference is the interval between successive probes or entry slots. When the slot at a hashed index for a key record is already occupied, we start traversing until you find an unoccupied slot. The interval between slots is computed by adding the successive value of an arbitrary polynomial in the original hashed index.

Let’s assume that the hashed index for a particular entry is index. The probing sequence for **quadratic probing** will be:

index = index % S

index = (index + 12) % S

index = (index + 22) % S

index = (index + 32) % S

.
.
.

and so on. Where S is the size of the hash table.

Source Code:
```
#include<iostream>
using namespace std;
int main()
{
    cout<<"\n Enter the size of the hash table:";
    int max;
    cin>>max;
    int hash[max],pos,i,n,ch,j=0,k=1;
    for(i=0;i<max;i++)
        hash[i]=0;
        while(1)
        {
            cout<<"\n 1.Insert 2.Delete 3.Display 4.Exit\n Enter Ur choice:";
            cin>>ch;
            switch(ch)
            {
                case 1:
                        cout<<"\n Enter element:";
                        cin>>n;
                        pos=n%max;
                        if(hash[pos]==0)
                            hash[pos]=n;
                        else
                        {
                            i=(pos+(j*j))%max;
                            j++;
                            while(hash[i]!=0&&j!=max)
                            {
                                i=(pos+(j*j))%max;
                                j++;
                            }
                            if(j==max)
                                cout<<"\n Hash table is full";
                            else
                                hash[i]=n;
                        }
                        break;
                case 2:
                        cout<<"\n Enter element:";
                        cin>>n;
                        pos=n%max;
                        j=0;
                        if(hash[pos]==n)
                            hash[pos]=0;
                        else
                        {
                            i=(pos+(j*j))%max;
                            k++;
                            while(hash[i]!=n&&k!=max)
                            {
                                i=(pos+(j*j))%max;
                                j++;
                            }
                            if(j==max)
                                cout<<"\n Element not found";
                            else
                                hash[i]=0;
                        }
                        break;
                case 3:
                        cout<<endl<<" Hash Table is : ";
                        for(i=0;i<max;i++)
                        cout<<" "<<hash[i];
                        break;
                case 4:
                        return 0;
                default:
                        cout<<"\n Wrong option \n try again .....";
            }
        }
    return 0;
}
```

Output:

```
Test Case-1
Enter the size of the hash table:10
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:23
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:33
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:43
1.Insert 2.Delete 3.Display 4.Exit
```

```
Enter Ur choice:1
Enter element:53
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
Hash Table is : 0 0 53 23 33 0 0 43 0 0
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:63
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
Hash Table is : 0 0 53 23 33 0 0 43 0 63
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:2
Enter element:53
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
Hash Table is : 0 0 0 23 33 0 0 43 0 63
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:2
Enter element:63
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
Hash Table is : 0 0 0 23 33 0 0 43 0 0
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:4
```

38. Implement Linear Probing.

Hashing is an approach in which the timing complexity of insert, delete and search operations is not dependent on the number of elements present in the data structure. In fact, all the operation like insert, delete and search operation have constant time complexity i.e O(1) when hashing approach is used.

In the hashing approach we will use a table called **Hash Table** to store the data. The size of the hash table is less than the range of the actual data. So the data values are inserted into the hash table based on the hash key value. Hash key value is the index of the hash table at which the actual data is stored. The Hash key value provides the mapping of the actual data and the index of the hash table.

Hashing function is a function which takes the key i.e the actual data to be inserted as input and gives the hash key value as output which is the index of the hash table at which the key is stored. Hashing function is always designed in such a way that the output of the hashing function is always within the range of the index of the hash table.

Choosing a good hashing function, h(k), is essential. h(k) should distribute the data as uniformly as possible to the indices of the hash table. h(k) is a function that maps universe of keys U to some index between 0 to S where S is the size of the hash table. When inserting, searching, or deleting a key k, h(k) is calculated and the h(k)th slot is checked to add, look for, or remove the key. Any hashing function h(k) should have the following properties

- Satisfy (approximately) the assumption of simple uniform hashing: each key is equally likely to hash to any of the S slots. The hash function shouldn’t bias towards particular slots.
- Quick to be calculated, should ideally have O(1) run-time.
- Should be deterministic i.e. h(k) should always return the same value for a given k.

Collision

If all keys hash to different slots of hash table, then the hash table operations are as fast as computing the hash function and changing or inspecting the value of an array element, which is

O(1) runtime. However, this is not always possible. If the number of possible keys is greater than the number of slots in the hash table (which is the case most of the times), then there must be some keys that hash into the same slot, in other words a collision

During collision, a newly inserted key maps to an already occupied slot in hash table and must be handled using some collision handling technique.

There are mainly two methods **to handle collision**:

- **Open Addressing** (Closed Hashing) with linear probing or quadratic probing or double hashing.
- **Separate Chaining** (Open Hashing).

Linear probing

In Linear probing, the interval between successive probes is fixed (usually to 1). Let's assume that the hashed index for a particular entry is index. The probing sequence for linear probing will be:
index = index % S
index = (index + 1) % S
index = (index + 2) % S
index = (index + 3) % S
...
...
and so on. Where S is the size of the hash table.

In other words, in linear probing when a collision occurs, the immediate free slot is occupied.

Source Code:
```
#include<iostream>
using namespace std;
int main()
{
    cout<<"\n Enter the size of the hash table:";
    int max;
    cin>>max;
    int hash[max],pos,i,n,ch;
    for(i=0;i<max;i++)
        hash[i]=0;
        while(1)
```

```
{
    cout<<"\n 1.Insert 2.Delete 3.Display 4.Exit\n Enter Ur choice:";
    cin>>ch;
    switch(ch)
    {
        case 1:
                cout<<"\n Enter element:";
                cin>>n;
                pos=n%max;
                if(hash[pos]==0)
                    hash[pos]=n;
                else
                {
                    i=pos;
                    pos=(pos+1)%max;
                    while(hash[pos]!=0&&i!=pos)
                        pos=(pos+1)%max;
                    if(i==pos)
                        cout<<"\n Hash table is full";
                    else
                        hash[pos]=n;
                }
        break;
        case 2:
                cout<<"\n Enter element:";
                cin>>n;
                pos=n%max;
                if(hash[pos]==n)
                    hash[pos]=0;
                else
                {
                    i=pos;
                    pos=(pos+1)%max;
                    while(hash[pos]!=n&&i!=pos)
                        pos=(pos+1)%max;
                    if(i==pos)
                        cout<<"\n Element not found";
                    else
                        hash[pos]=0;
                }
                break;
        case 3:
                cout<<endl<<"Hash table Contents :";
                for(i=0;i<max;i++)
                cout<<" "<<hash[i];
```

```
                                break;
                        case 4:
                                return 0;
                        default:
                                cout<<"\n Wrong option \n try again .....";
                }
        }
    return 0;
}
```

Output:

```
Test Case-1
Enter the size of the hash table:10
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:33
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:53
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:66
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
Hash table Contents : 0 0 0 33 53 0 66 0 0 0
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:1
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:16
Wrong option
try again .....
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:16
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
Hash table Contents : 0 1 0 33 53 0 66 16 0 0
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:2
Enter element:53
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
Hash table Contents : 0 1 0 33 0 0 66 16 0 0
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:2
Enter element:16
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
Hash table Contents : 0 1 0 33 0 0 66 0 0 0
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:2
Enter element:16
Element not found
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:4
```

40. Implement Double Hashing

In this program we used the double hashing technique to overcome collisions

- **hash function 1: n mod 20;**
- **hash function 2: R - (n mod R);**
- **where R is (23) the smallest prime greater than the size of table**

Hashing is an approach in which the timing complexity of insert, delete and search operations is not dependent on the number of elements present in the data structure. In fact, all the operation like insert, delete and search operation have constant time complexity i.e O(1) when hashing approach is used.

In the hashing approach we will use a table called **Hash Table** to store the data. The size of the hash table is less than the range of the actual data. So the data values are inserted into the hash table based on the hash key value. Hash key value is the index of the hash table at which the actual data is stored. The Hash key value provides the mapping of the actual data and the index of the hash table.

Hashing function is a function which takes the key i.e the actual data to be inserted as input and gives the hash key value as output which is the index of the hash table at which the key is stored. Hashing function is always designed in such a way that the output of the hashing function is always within the range of the index of the hash table.

Choosing a good hashing function, h(k), is essential. h(k) should distribute the data as uniformly as possible to the indices of the hash table. h(k) is a function that maps universe of keys U to some index between 0 to S where S is the size of the hash table. When inserting, searching, or deleting a key k, h(k) is calculated and the h(k)th slot is checked to add, look for, or remove the key. Any hashing function h(k) should have the following properties

- Satisfy (approximately) the assumption of simple uniform hashing: each key is equally likely to hash to any of the S slots. The hash function shouldn’t bias towards particular slots.
- Quick to be calculated, should ideally have O(1) run-time.
- Should be deterministic i.e. h(k) should always return the same value for a given k.

Collision

If all keys hash to different slots of hash table, then the hash table operations are as fast as computing the hash function and changing or inspecting the value of an array element, which is O(1) runtime. However, this is not always possible. If the number of possible keys is greater than the number of slots in the hash table (which is the case most of the times), then there must be some keys that hash into the same slot, in other words a collision

During collision, a newly inserted key maps to an already occupied slot in hash table and must be handled using some collision handling technique.

There are mainly two methods **to handle collision**:

- **Open Addressing** (Closed Hashing) with linear probing or quadratic probing or double hashing.
- **Separate Chaining** (Open Hashing).

Double hashing

Double hashing is similar to linear probing and the only difference is the interval between successive probes. Here, the interval between probes is computed by using a second hash function.

Let us say that the hashed index for a key (k) is an index that is computed by one hashing function and the slot at that index is already occupied. You must start traversing in a specific probing sequence to look for an unoccupied slot. The probing sequence will be:

index = (index + 1 * Hash2(k)) % S

index = (index + 2 * Hash2(k)) % S

index = (index + 3 * Hash2(k)) % S

.

.

. and so on

Here, Hash2() is the second hash function.

Choosing second hash function

A popular second hash function is : hash2(key) = PRIME – (key % PRIME) where PRIME is a prime smaller than the hash table size S.

A good second Hash function is:

- It must never evaluate to zero.
- Must make sure that all cells can be probed.

Source Code:

```
#include<iostream>
using namespace std;
int main()
{
    int hash[20],pos,i,n,ch,j=0;
    for(i=0;i<20;i++)
            hash[i]=0;
            while(1)
            {
                cout<<"\n 1.Insert 2.Delete 3.Display 4.Exit\n Enter Ur choice:";
                cin>>ch;
                switch(ch)
                {
                    case 1:
                            cout<<"\n Enter element:";
                            cin>>n;
                            pos=n%20;
                            if(hash[pos]==0)
                                 hash[pos]=n;
                            else
                            {
                                pos=(pos+(j*(23-(n%23))))%20;
                                j++;
                                while(hash[pos]!=0&&j!=20)
                                {
                                pos=(pos+(j*(23-(n%23))))%20;
                                j++;
                                }
                                if(j==20) cout<<"\n Hash table is full";
                                else
                                hash[pos]=n;
```

```
                        }
                            j=0;
                        break;
                case 2:
                        cout<<"\n Enter element:";
                        cin>>n;
                        pos=n%20;
                        if(hash[pos]==n)
                            hash[pos]=0;
                        else
                        {
                            pos=(pos+(j*(23-(n%23))))%20;
                            j++;
                            while(hash[pos]!=n&&j!=20)
                            {
                                pos=(pos+(j*(23-(n%23))))%20;
                                j++;
                            }
                            if(j==20)
                                cout<<"\n Element not found";
                            else
                                hash[pos]=0;
                            }
                            j=1;
                        break;
                case 3:
                        cout<<endl;
                        for(i=0;i<20;i++)
                        cout<<" "<<hash[i];
                        break;
                case 4:
                        return 0;
                default:
                        cout<<"\n Wrong option \n try again .....";
            }
        }
    return 0;
}
```

Output:

Test Case-1

```
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:24
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:34
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:1
Enter element:44
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
0 0 0 0 24 0 44 0 0 0 0 0 0 0 34 0 0 0 0 0
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:2
Enter element:34
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:3
0 0 0 0 24 0 44 0 0 0 0 0 0 0 0 0 0 0 0 0
1.Insert 2.Delete 3.Display 4.Exit
Enter Ur choice:4
```

Evaluation Procedure for Internal Laboratory Examinations:

1. Of the 25 marks for internal, 10 marks will be awarded for day-to-day work and 10 marks to be awarded for the Record work and 5 marks to be awarded by conducting an internal laboratory test.
2. Concerned Teachers have to do necessary corrections with explanations.
3. Concerned Lab teachers should enter marks in index page.
4. Internal exam will be conducted by two Staff members.

Dr.K. Subba Reddy

Professor & Head Dept. of CSE.

RAJEEV GANDHI MEMORIAL COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

(ESTD-1995)

Evaluation Procedure for External Laboratory Examinations:

1. For Practical subjects there is a continuous evaluation during the semester for 25 Sessional marks and 50 end examination marks.
2. The end examination shall be conducted by the teacher concerned (Internal Examiner) and another External Examiner, recommended by Head of the Department with the approval of principal.

Evaluation procedure for external lab examination:

1. Procedure for the program	---------- 20M
2. Execution of the program	---------- 15M
3. Viva voce	---------- 15M
Total	50M

Dr.K. Subba Reddy

Professor & Head Dept. of CSE.