OOPC Assignment : 2

20DCS103 - Rushik Rathod.

Page No.
Date / /

1. Destructor :

Destructors are usually used to de-allocate memory and do other cleanup for a class object and its class members when the object is destroyed.

→ A destructor is called for a class object when that object passes out of scope or is explicity deleted.

→ Example :-

```
#include <iostream>
using namespace std;
class Simple
{ public :
    Simple()          // constructor is made
    { cout << " Constructor is called " << endl ; }.
    ~ Simple()        // Destructor is made
    { cout << " Destructor is called " << endl ; }.
     void display()
     { cout << " Member function is called " << endl ; }.
};
int main()
{
   Simple s1;         // object created.
   s1.display();
   return 0;
}.
```

→ Output : Constructor is called
Member function is called
Destructor is called

2. i) Default Constructor :

→ A constructor which does not have any parameters or arguments, then it is called default constructor.

→ If programmer does not create any default constructor then compiler automatically creates one default constructor.

→ Example :-

```
#include <iostream>
using namespace std;
class Simple
{
  public :
     Simple()
     { cout << " Default constructor is called." <<endl;
     }
     void display()
     { cout<< " Member function is called." <<endl;}
};
int main()
{
  Simple S1 ;     // Object created
  s.display() ;
  return 0 ;
}
```

Page No.
Date / /

→ Output : Default constructor is called.
Member function is called.

ii) Parameterized constructor :

→ If a constructor is accepting the parameters then it is called a parameterized constructor.

→ When a programmer created a parameterized constructor then one needs to pass the initial values as arguments to the constructor when object is declared.

→ Example :-

```
#include <iostream>
using namespace std;
class Complex
{   int a,b;
    public:
        Complex ( int x , int y ) // parameterized constructor
        {   a = x ;
            b = y ;
        }
        void display ()
        { cout << " a = " << a << endl ;
          cout << " b = " << b << endl ;      }
};
int main()
{   Complex C1 ( 10 , 20 ) ;
```

```
    C1.display()
    return 0;
}
```

→ Output: a = 10
b = 20.

3. ** Operators which can NOT be overloaded:

sizeof operator → sizeof
Membership operator → . (dot)
Pointer to member op.. → *
Scope resolution op.. → ::
Conditional operator → ?:

** Operators which can be overloaded by member function of a class but can NOT be overloaded by friend function.

Assignment operator =
Function call operator ()
Subscripting operator []
Class member access operator →

20IDCS103 - Rushik Rathod

4. Operator Function :

When we need to define an additional task to an operator, then we must specify what it means in relation to the class to which the operator is applied.

→ Syntax :

keyword

```
returntype classname :: operator op(argumentlist)
{
    function body
}
```

→ Arguments required :-

	Unary	Binary
Using member function	0	1
Using friend function	1	2

5. Operator overloading :-

When we provide special meaning to an operator then it is known as operator overloading.

→ C++ provides this facility to users.

→ Operator Overloading provides a flexible option for the creation of new definitions for the C++ operators.

Page No.
Date / /

→ Example :- Overloading Unary operator.

```
#include <iostream>
using namespace std;
class Complex
{ private :
      int a, b ;
  public :
      Complex ( int x , int y )
      {
          a = x ;
          b = y ;          }
      void display()
      { cout << " a = " << a << " \n b = " << b << endl
      }
      void operator -()
      { a = -a ;
          b = -b ;          }
} ;
int main()
{
    Complex c1 ( 3, -4 ) ;
    c1. display ;
    - c1 ;
    c1. display ;
    return 0 ;
}
```

→ Output :-

a = 3
b = -4
a = -3
b = 4

6. Constructor in derived class, when parameterized constructor is in base class :-

→ When any base class contains a parameterized constructor then it is compulsory for the derived class to have a constructor and pass the arguments to the base class constructor.

→ Example :-

```
#include <iostream>
using namespace std;
class Base
{   int a;
 public:
    Base(int x)
    {   a = x;  }
        cout << "a = " << a << endl;  }.
};

class Derived : public Base
{   int b;
  public:
```

```
    Derived (int y , int x ) : Base(x)
    {   b = y ;
        cout << "b = " << b << endl;  }
};

int main()
{
    Derived d(10, 20);
    return 0;
}
```

→ Output :- a = 20
b = 10

7. Differentiate :-

i) Function overloading	i) Function overriding
→ When we declare same functions with different arguments in the same class, then function overloading happens.	→ When we declare same functions in both the Base class and in the derived class, then function overriding happens.
→ In function overloading one can have any number of overloaded functions.	→ In function overriding one can only override functions once in the child class.

20DCS103 - Rushik Rathod

Page No.
Date

→ Overloading happens at the compile time. therefore it is known as compile time polymorphism.

ii) Opening a file with a constructor function

→ When we use a file name to initialize the file stream object, it is called opening a file using constructor

→ Example :

```
#include <bits/stdc++.h>
using namespace std;
int main()
{ char name[20]
  int no;
  ofstream outf("D:\\file\\
        simple");
  cout << "Enter number
  and name :");
  cin >> no >> name;
  outf << no << "\n";
  outf << name;
  outf.close();
  return 0;
}
```

→ Overriding happens at run time therefore it is knowns as run time polymorphism.

ii) Opening a file with open() function.

→ When we use open() to open a file, we may create a single stream object and use it to open each file in turn.

→ Example :

```
# include <bits/stdc++.h>
using namespace std;
int main()
{
  ofstream outf;
  outf.open("DATA1");
  - - - - - - - - -
  outf.close();
  outf.open("Data2");
  - - - - - - -
  outf.close();
  return 0;
}
```

Page No.
Date / /

20DCS103 - Rushik Rathod

iii) Virtual functions

→ A virtual function is a member function of base class which can be redefined by derived class.

→ Example :-

```
#include <iostream>
using namespace std;
class Base
{ public:
  virtual void put()
  { cout << "put baseclass"
        << endl; }
  void display()
  { cout << "display base
      class" << endl; }
};
class Derived : public Base
{ public;
  void put()
  { cout << "put derived
      class" << endl; }
  void display()
  { cout << "display derived
      class" << endl; }
};
```

iii) Virtual Base class

→ When two subclasses access a single copy of parent class, then the parent class is known as virtual base class.

→ Example :-

```
#include <iostream>
using namespace std;
class Super
{ protected;
    int a;
  public:
    void getdata(int x)
    { a = x; }
};
class Sub1 : public Super
{ protected:
    int b;
  public:
    void getb.(int y)
    { b = y; }
};
class sub2 : public Super
{ protected:
    int c;
  public:
```

2019CS103 - Rushik. Ruthod

```
int main()
{
  Base *bptr;
  Derived d;
  bptr = d;
  bptr -> put();
  bptr -> display();
  return 0;
}
```

→ Output :

put derived class
display base class.

→ Hence, from this example we can easily understand the use of virtual functions.

```
void getc(int z)
{ c=z; }
};
class Sub12 : public sub1, public sub2
{ public:
    void display()
    {
      cout<<"a = "<<a
          <<endl;
      cout<<"b = "<<b
          <<endl;
      cout<<"c = "<<c
          <<endl;}
};
int main()
{
    Sub12 s;
    s.geta(1);
    s.getb(2);
    s.getc(3);
    s.display;
    return 0;
}
```

→ Output : a = 1
b = 2
c = 3

→ This is the way how virtual base class works.

iv) Virtual Base class

→ When two sub classes access a single copy of parent class, then the parent class is known as virtual base class.

→ Syntax :

```
class Super
{ // body  };
class Sub1 : public Super
{ // body  };
class Sub2 : public Super
{ // body  };
class Sub12 : public sub1,
              public Sub2.
{
   // body  };
int main()
{ ...
  ... }.
```

→ Here, class super is called a virtual base class.

iv) Abstract class.

→ When a class has atleast one pure virtual function, then it is called an abstract class.

→ Syntax :

```
class Super
{
   virtual void get()=0;
}.
int main()
{
   ...
   ...
}.
```

→ Here, class super is called an abstract class which contains one pure virtual function.

20DCS103 - Rushik Rathod

8. Type conversion: class type to basic type

→ When we want to convert a particular value of a class to another data type, then we need to do type conversions as shown below:

→ Example:

```
#include <iostream>
using namespace std;
class Complex
{ int a;
 public:
    void getdata(int x)
    {   a = x;
        cout << " a = " << a << endl;   }
    operator int()
    {   return a;   }
};
int main()
{
   Complex c1;
   c1.getdata(3);
   int temp;
   temp = c1; // assigning c1 to temp
   cout << " temp = " << temp << endl;
   return 0;
}
```

→ Output :

a = 3
temp = 3

→ Three rules for defining an overloaded casting operator function :

1) It must be a class member.
2) It must not specify a return type.
3) It must not have any arguments.

9. Type conversion : basic type to class type

→ When we need to assign a particular value to an object of a class, then we need to use type conversion from basic type to class type. as shown below:

→ To convert basic type to class type, we need to use constructor in the program.

→ Example :-

```
#include <iostream>
using namespace std;
class Complex
{   int a;
```

20DCS103 - Rushik. Rathod

```
public :
    Complex()    // default constructor
    { }
    Complex ( int x)
    {    a = x ;
        // cout << ++a = --a << a << endl;
    }
    void display()
    {    cout << " a = " << a << endl ; }
    void getdata( int t1 )
    {    a = t1 ;                          }
};
int main()
{
    Complex c1 ;
    int temp = 4 ;          c1. getdata(3);
    c1 = 4 ;                c1. display();
    c1 =                    int temp = 4 ;
     c1 = temp ;
     c1. display() ;
     return 0 ;
}
```

→ Output :

a = 3
a = 4 ← From converting the value (basic type to class type)

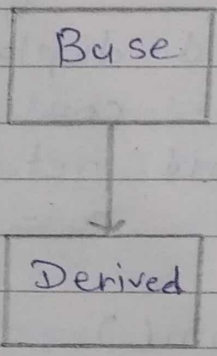
20DCS103 . Rushik. Rathod.

10. Types of Inheritance :-

i) Single Inheritance :-

A derived class with only one base class is called single Inheritance.

→ Example :




```
#include <iostream>
using namespace std;

class Base
{ protected :
   int a;
   public :
   void geta( int x)
   {  a = x ;                    }
};

class Derived : public Base
{ protected :
    int b;
   public ;
   void getb(int y)
   {
      geta(3);          // calling geta function
      b = y ;                    }
   void display()
   { cout << " a= " << a << "\nb = " << b << endl;}
};
```

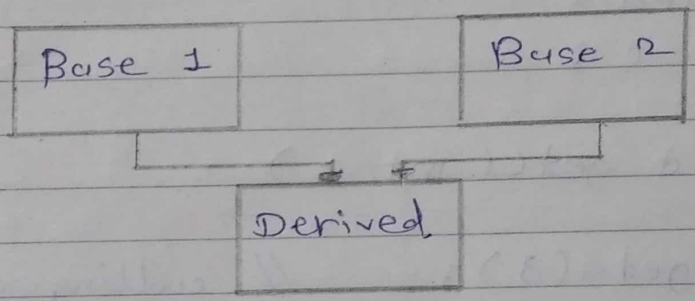
```
int main()
{
    Derived d;
    d.getb(4);
    d.display();
    return 0;
}.
```

→ Output :

```
a = 3
b = 4
```

ii) Multiple Inheritance :-

A derived class with more than one base class is called multiple inheritance.



→ Example :-

```
#include <iostream>
using namespace std;

class Base1
{ protected :
```

```
    int a;
    public:
        void geta (int x)
        {   a = x;                    }.
};

class Base2 .
{ protected:
    int b;
  public:
    void getb( int y ).
    { b = y;                          }
};

class Derived : public Base1 , public Base2
{ protected:
    int c;
  public:
    void getc( int z ).
    {
        geta(3);        // calling geta function
        getb(4);        // calling getb function
        c = z;
    }
    void display()
    { cout << " a = " << a << endl;
      cout << " b = " << b << endl;
      cout << " c = " << c << endl;   }
};
```

20DCS103 - Rushik. Rathod.

```
int main()
{
    Derived d;
    d.getc(5);
    d. display();
    return 0;
}.
```

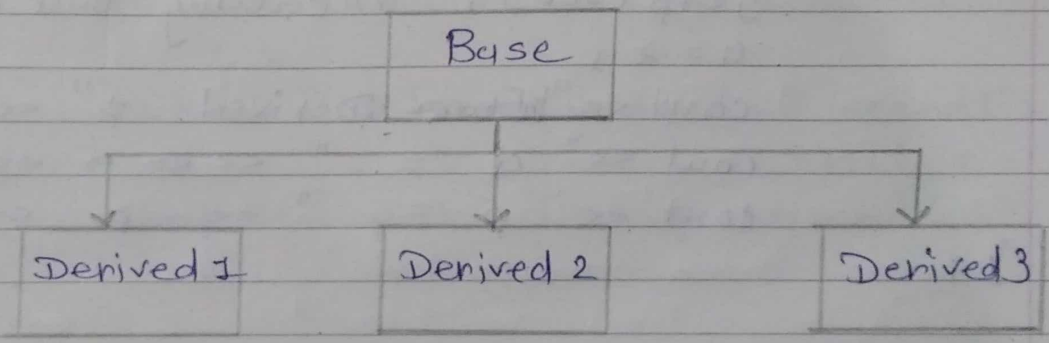
→ Output :

a	=	3
b	=	4
c	=	5

~~a=3~~
~~b=4~~
~~c=5~~.

iii) Hierarchical Inheritance :-

A base class with more than one derived classes is called hierarchical inheritance



→ Example :

```
#include <iostream>
using namespace std;
```

20DCS103 - Rushik. Puthod

```
class Base
{
  protected :
      int p;
  public :
      void getp (int t)
      {   p = t;                    }
};

class Derived 1 : public Base
{
  protected :
      int a;
  public :
      void geta (int x)
      {
          getp (100);   // calling base class function
          a = x;
          cout << "In From Derived 1" << endl;
          cout << " a = " << a << endl;
          cout << " p = " << p << endl;
};

class Derived 2 : public Base
{
  protected :
      int b;
  public :
```

20DCS103 - Rushik. Rathod

```
    void getb(int y)
    {
        getp(200); // calling base class function
        b = y ;
        cout << "\n From Derived 2" << endl;
        cout << " b = " << b << endl;
        cout << " p = " << p << endl;
};

class Derived3 : public Base
{
  protected :
      int c ;
  public :
      void getc(int z)
      {
          getc(300); // calling base class function
          c = z ;
          cout << "\n From Derived 3" << endl;
          cout << " c = " << c << endl;
          cout << " p = " << p << endl;
};

int main()
{  Derived1 d1;
   d1.geta(10);
   Derived2 d2;
   d2.getb(20);
   Derived3 d3;
```

```
    d3.getc(30);
    return 0;
}
```

→ Output :-

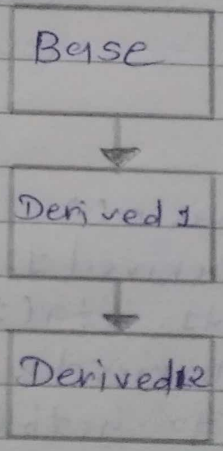
From Derived 1
a = 10
p = 100

From Derived 2
b = 20
p = 200

From Derived 3
c = 30
p = 300.

iv) Multilevel Inheritance :

The process of deriving a class from another class is called multilevel inheritance.



20DCS103 - Rushik. Ruthod

→ Example :-

```
#include <iostream>
using namespace std;

class Base
{ protected :
      int a;
  public :
      void geta (int x)
      {  a = x ;                    }.
};
class Derived1 : public Base
{ protected :
      int b;
  public :
      void getb (int y)
      { geta (10) ;  // calling function of
        b = y ;                    // Base class   }
};
class Derived2 : public Derived1.
{ protected :
      int c;
  public :
      void getc ( int z)
      { getb (20) ; // calling function of baseclass
        c = z ;
      }.
```

Page No.
Date / /

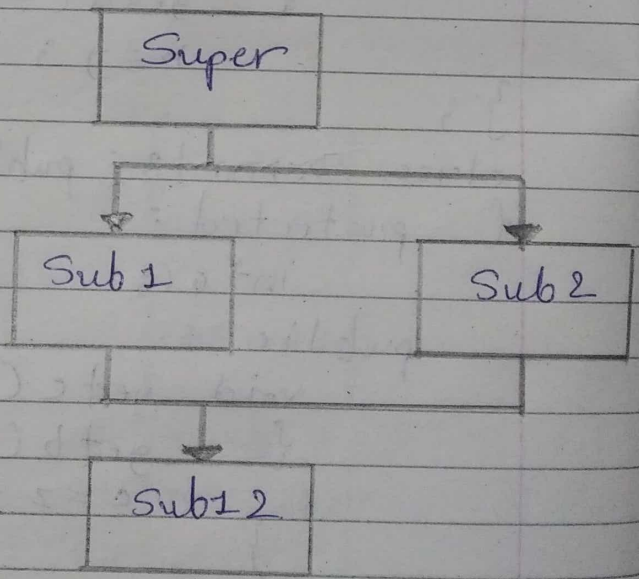
```
void display()
{  cout << " a = " << a << endl ;
   cout << " b = " << b << endl ;
   cout << " c = " << c << endl ;
}
int main()
{
   Derived2 d2;
   d2.getc(30);
   d2.display();
   return 0;
}
```

→ Output :-

a = 10
b = 20
c = 30

v). Hybrid Inheritance :

The process of applying two or more types of inheritance to design a program is called hybrid Inheritance.



20DCS103 - Rushik Rathod

→ Example :-

```
#include <iostream>
using namespace std;

class Super
{   protected :
        int a ;
    public :
        void geta ( int x )
        {     a = x ;                    }.
};

class Sub1 : virtual public super.
{   protected :
        int b ;
    public :
        void getb (int y )
        {  b = y ;                       }.
};

class Sub2 : Virtual public Super.
{   protected :
        int c ;
    public :
        void getc ( int z )
        {  c = z ;                       }
};
```

```
class Sub12 : public Sub1, public Sub2
{ public :
      void getdata()
      {   geta(10);
          getb(20);
          getc(30);          }
      void display()
      { cout << " a = " << a << endl;
        cout << " b = " << b << endl;
        cout << " c = " << c << endl;    }
};
int main()
{
   Sub12 s;
   s.getdata();
   s.display();
   return 0;
}
```

→ Output :-

```
a = 10
b = 20
c = 30
```

11. Virtual functions :-

→ A virtual function is a member function of a base class whose resolution with base class pointers and references occurs at run time.

20DCS103 - Rushik Rathod

Page No.
Date 1.9.21

→ The need for having a virtual function is to implement a different functionality in the derived class.

→ ~~Explain~~ → Example :-

```
#include <iostream>
#include <conio.h>
using namespace std;

class Base
{
    public:
        virtual void fun()
        {
          cout<< " Member function fun() of baseclass
                  accessed " << endl;
        }
};

class Derived : public Base
{
    public:
        void fun()
        {
          cout<<" Member function fun() of derived
                  class accessed " << endl
        }
};
```

20DCS103 - Rushik. Rathod

```
int main()
{
    Base *m1;
    Derived d1;
    m1 = &d1;
    m1 -> fun();
    return 0;
}
```

→ Output :-

Member function fun() of derived class accessed

** Pure virtual function :-

→ Pure virtual function has no body; therefore a programmer must need to add the =0 notation for declaration of the pure virtual function in the base class.

→ Syntax :

```
class classname
{ public :
        virtual void virtualfunctionname() = 0;
            // pure virtual function
};
```

Page No.
Date / /

12. Explain public, private and protected access specifier with example.

→ Access specifiers define how the methods / members of a class can be accessed.

→ There are (3) access specifiers in C++ :-

1) public : members are accessible from outside the class.
2) private : members can not be accessed or viewed by the outside class.
3) protected : members cannot be accessed from outside the class, however, they can be accessed in inherited classes and friend functions

→ Example :-

```
#include <iostream>
using namespace std;
class Complex
{ private:
      int a;
  protected:
      int b;
  public:
      int c;
};
```

20DCS103 - Rushik Rathood

```
int main()
{ Complex c1;
    c1.a = 10;    // Not allowed
    c1.b = 20;    // Not allowed.
    c1.c = 30;
    return 0;
}
```

13. Constructor in derived classes :-

→ If any base class contains a parameterized constructor then it is compulsory for the derived class to have a ~~declared~~ constructor and pass the arguments to the base class constructor.

→ When both the derived and base classes contain constructors then the base class constructor is executed first and then the derived class constructor will be executed.

→ In case of multiple inheritance, the base classes are executed in the order in which they appears in the declaration of the derived class.

→ Here is the table which demonstrates the correct order of execution :-

Page No.
Date / /

Method of Inheritance	Order of Execution.
class B : public A { } ;	A() ; Base constructor B() ; Derived constructor
class A : public B, public C { } ;	B() ; base [first] C() ; base [Second] A() ; derived
class : public B, virtual public C { } ;	C() ; virtual base B() ; ordinary base A() ; derived

14. Polymorphism :-

The polymorphism is the ability to use an operator or function in different ways.

→ Polymorphism gives different meanings or functions to the operator or functions.

* Early Binding :-

→ The compile time polymorphism is implemented with templates.
→ It includes function overloading and operator overloading.
→ Early binding is the binding that an object is bound to its function call at compile time.

20DCS103 - Rushik. Rathod.

→ The main advantage of early binding is efficiency and disadvantage is the lack of flexibility.

* Late Binding :-

→ The run-time polymorphism is implemented with inheritance and virtual functions.
→ Dynamic binding is the binding that appropriate function is done at run time.
→ Dynamic binding requires the use of pointers to objects.
→ The main advantage of dynamic binding has better flexibility than static binding.

** Virtual functions in run-time polymorphism:

→ The vital reason for having a virtual function is to implement a different functionality in the derived class.

→ Example :-

```
#include <iostream>
using namespace std;

class Base
{ protected:
        int a;
  public:
```

20DCS103 - Rushik. Rathod.

```
// geta(int x)
// {
// virtual void geta(int x)
// Base(){ }. // default constructor
   Base(int x)  // parameterized constructor
   { a = 10; }.
   virtual void display()
     { cout<<" From base a = "<<a<<endl;}
};

class Derived : public Base
{ public:
     void display()
     { cout<<" From derived a = "<<a<<endl;}
};

int main()
{
   Base *B1;
   Derived D1;
    B1 = &D1;
    B1 -> display();
    return 0;
}
```

→ Output :-

From derived a = 10

20DCS103 - Rushik Rathod

15. Containership :-

→ Containership is a special type of aggregation that implies ownership. It is also called composition.

→ It allows any class to access the members of another class by making the object of that class.
For example, class A could contain an object of class B as a member.

** How containership is differ from inheritance :-

→ The main difference between inheritance and containership is that inheritance allows using properties and methods of an existing class in the new class while, containership is another name for composition that describes the ownership between the associated objects.

→ Fyrthermore, inheritance provides code reusability while containership allows representing the association.

→ In conclusion, inheritance and containership are two relationships, where inheritance is the methodology of creating a new class using the properties and methods of an

existing class and containership is a type of aggregation that allows a class to contain an object of a different class as a member data.

16. this pointer :-

→ this pointer represents an object that invokes a member function.

→ Example :-

```
#include <iostream>
using namespace std;

class Complex
{ int a;
  public:
    void getdata(int x)
    {
      this -> a = x;                    }
    void max();
    void putdata()
    { cout << " a = " << endl;          }
};
void Complex :: max(Complex s2)
{
  if(a > s2.a)
    return a* this;
```

```
    else
        return s2;
}
int main()
{
    Complex c1, c2, c3;
    c1.getdata(10);
    c2.getdata(20);
    c1.putdata();
    c2.putdata();
    c3 = c1.max(c2);
    c3.putdata();      // which displays the
    return 0;          // maximum number.
}
```

→ Output :-

```
a = 10      <-- Value of object c1
a = 20      <-- Value of object c2
a = 20      <-- Value of object c3. which is longer.
```

→ Here, the class Complex has data members such as getdata, putdata to get and print the data respectively.

→ max function is declared in the scope of class Complex and defined outside the class.

20DCS103 - Rushik. Rathod.

→ The max function is called by c1 and passed arguments c2. Therefore, a refers to c1 and s2.a refers to c2.

→ The function returns object s2 if s2.a is greater otherwise returns 'a'. So that inside the function we have to write return * this instead of return c1.

17. Explain the following functions:-

(i) Open()

→ The function open() is used to open multiple files that use the same stream object.

→ For example: we may want to process a set of files sequentially. In such cases, we may create a single stream object and use it to open each file in turn.

Mode.	Description
ios::app	opens a text file for appending. (to add text at the end)
ios::ate	opens a file for output and move the read/write control to the end of the file.
ios::in	opens a text file for reading.

ios :: out opens a text file for writing.
ios :: trunc truncates the content before opening a file, if file exists.

→ Example :-

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ofstream file;
    file.open("example.txt");
    return 0;
}
```

→ In the above example, we have opened the file 'example.txt' to write on it. 'example.txt' file must be creating in working directory.

(ii) eof()

→ The eof() method of ios class in c++ is used to check if the stream is has raised any EOF [End of File] error.

→ It means that this function will check if the

Page No.
Date / /

stream has its eofbit set.

→ This method does not accept any parameter.

→ This method returns true if the stream has eofbit set, else false.

→ Example :-

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    stringstream ss;
    bool isEOF = ss.eof();
    cout << " is stream eof : " << isEOF << endl;
    return 0;
}
```

→ Output :-

```
is stream eof : 0.
```

(iii) seekg()

→ seekg() is a function in the iostream library that allows us to seek an arbitrary position in a file.

20DCS103 - Rushik Rathod.

→ It is mainly used to set the position of the next character to be extracted from the input stream from a given file in c++ file handling.

→ Example :-

```
#include<fstream>
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
    fstream File("d.txt", ios::in |ios::out);
    File << "charusat";
    File.seekg(9, ios::beg);
    char F[9];
    File.read(F, 5);
    F[5] = 0;
    cout << F << endl;
    File.close();
}
```

(iv) close()

→ A file which is opened while reading or writing in file handling must be closed after performing an action on it.

→ The close() function is used to close the file currently associated with the object.

Page No.
Date / /

→ Example :-

```
#include <fstream>
#include <iostream>
using namespace std;

int main()
{
    ofstream myfile;
    myfile.open ("employee.txt");
    if (myfile.is_open())
    {
        cout<<"File is open"<<endl;
        myfile.close();
        cout<<"File is closed"<<endl;
    }
    else
        cout<<"Error in file opening."<<endl;
    return 0;
}
```

→ Output :-

```
File is open
File is closed
```

(v) put()

→ The put() function is used to write a single

20DCS103 - Rushik Rathod

character into file

→ Example :-

```
#include <fstream>
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    ofstream ofstream_ob;
    ofstream_ob.open("File2.txt", ios::out);
    char arr[100] = "Hello World.";
    int length = strlen(arr);
    char ch;

    for(int i=0; i<length; i++)
    {
        ch = arr[i];
        ofstream_ob.put(ch);
    }
    ofstream_ob.close();
    return 0;
}
```

→ Output :-

Hello World.

Page No.
Date / /

(vi) fail()

→ The fail() method is used to check if the stream is has raised any error.

→ This function will check if this stream has failbit set.

→ Example :-

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    stringstream ss;
    bool isfail = ss.fail();
    cout << "is stream fail: " << isfail << endl;
    return 0;
}
```

→ Output :-

```
is stream fail: 0
```

→ This method does not accept any parameters and returns true if stream has failbit set, else false.

(vii) bad()

→ The bad() method is used to check if the stream is has raised any bad error.

→ This method will check if this stream has its badbit set.

→ Example :-

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    stringstream ss;
    bool isbad = ss.bad();
    cout << "is stream bad : " << isbad << endl;
    return 0;
}
```

→ Output :

```
is stream bad : 0
```

→ This method does not accept any parameters and it returns true if the stream has badbit set, else false.

Page No.
Date / /

(viii) good()

→ The good() method is used to check if the stream is good enough to work.

→ It means that this function will check if this stream has raised any error or not.

→ Example :-

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    stringstream ss;
    bool isgood = ss.good();
    cout << " is stream good : " << isgood <<
                                    endl;

    return 0;
}
```

→ Output :-

```
is stream good : 1
```

→ This method does not accept any parameters and returns true if the stream is good, else false.

(ix) seekp

→ The seekp(pos) method of ostream in C++ is used to set the position of the pointer in the output sequence with the specified position.

→ This method takes the new position to be set and returns this ostream instance with the position set to specified new position.

→ Example :-

```
#include <bits/stdc++.h>
using namespace std;

class Student
{
    int rno;
    char name[30];
public:
    void getdata()
    {
        name = "Rushik";
        rno = 103;
    }
    void putdata()
    { cout << rno << endl << name << endl;
    }
```

Page No.
Date / /

```
    void DisplayRecordAtPosition(int);
};

void Student::DisplayRecordAtPosition(int n)
{
    ofstream ofs;
    ofs.open("he.dat", ios::out || ios::binary);

    cout << "size of record : " << sizeof(*this)
                                << endl;
    ofs.seekp((n-1) * sizeof(student));
    ofs.write((char*)this, sizeof(student));
    ofs.close();
    ifstream ifs;
    ifs.open("he.dat", ios::in || ios::binary);
    ifs.seekg((n-1)* sizeof(student));
    ifs.read((char*)this, sizeof(student));
    putdata();
    ifs.close();
}
int main()
{   Student s;
    int pos = 1;
    s.getdata();
    cout<<"record no " << pos <<" (position
            int file " << pos-1 << ")\n";
    s.DisplayRecordAtPosition(pos);
    return 0;
}
```

→ Output :-

size of record : 24
record no 1 (position in file 0)
name : Rushik

(X) tellg()

→ The tellg() function is used with input streams and returns the current "get" position of the pointer in the stream.

→ It has no parameters and returns a value of the member type pos_type, which is an integer data type representing the current position of the get stream pointer.

→ Example :-

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string str = "Rushik";
    istringstream in(str);
    string word;
    in >> word;
    cout << "After reading the word \" " << word
```

Page No.
Date / /

<< "1" tellg() returns" << in.tellg()
<< '\n';

→ Output :-

After reading the word "Rushik" tellg() returns -1

18 tellp() function :-

→ The tellp() is used with output streams, and returns the current "put" position of the pointer in the stream.

→ It has no parameters and return a value of the member type pos_type, which is an integer data type representing the current position of the put stream pointer.

→ Example :-

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    fstream file;
    file.open("myfile.txt", ios::out);
    file << "Hello";
```

```
    cout << " the current position of pointer is : "
         << file.tellp() << endl ;
    file.close()
}.
```

→ Output :-

the current position of pointer is : -1

19. classes for file stream operations :-

→ The I/O system of c++ contains a set of classes that defines the file handling methods.

→ filebuf : Its purpose is to set the file buffers to read and write. Contains Openprot constant used in the open() of file stream classes. Also contains close() and open() as members.

→ fstreambase : Provides operations common to file streams. Serves as a base for fstream, ifstream and ofstream class. Contains open() and close() functions.

→ ifstream : Provides input operations. Contains open() with default input mode. Inherits the functions get(), getline(), read(), seekg(), tellg() functions from istream.

→ ofstream : Provides output operations. Contains open() with default output mode. Inherits put(), seekp(), tellp() and write() functions from ostream.

→ fstream : Provides support for simultaneous input and output operations. Contains open with default input mode. Inherits all the functions from istream and ostream classes through iostream.

20. while(fin) & while(fin.eof()!=0)

→. Let us assume that example.txt file exists.

```
ifstream in;
in.open("example.txt");
string str;
do {
    getline(in,str);
```

```
        cout << str << endl;
    } while (in)
    in.close();
```

→ In the above program if we write just in the object of ifstream in while(...), then it will ~~open~~ print the while content of file.

```
ifstream.in
in.open("example.txt");
string str;
while(in.eof() != 0)
{
    getin(in, str);
    cout << str << endl;
}.
in.close();
```

→ In the above program code if we write in.eof() != 0, then it will print only first line of the file as output.

→ The main reason for this is that eof() is a member function of ios class. and returns a non-zero value if the end-of file condition is encountered and a non-zero otherwise. Therefore, the above program statement terminates the program on reaching the end of file.

22. put() : The put() function is used to write a line or string to the output stream. It prints the passed string with a newline and returns an integer value.

get() : The get() function in c++ reads characters from string and stores them until a new line character is found or end of file occurs.

read() : The read() binary function is used to perform file input operations. i.e. to read the object stored in a file.

write() : The write() binary function is used to perform file output operation. i.e. to write the objects to a file, which is stored in the computer memory in a binary form. Only the data member of an object are written and not its member functions.

23. Command line arguments :-

→ The most important function of c++ is main() function.

20DCS103 - Rushik. Rathod.

→ It is mostly defined with a return type of int and without parameters :

```
int main()
{ /*_ _ _
    _ _ _ _ */ }
```

→ Command line arguments are given after the name of the program in command line shell of operating system.

→ To pass command line arguments, we typically define main() with two arguments : first argument is the number of command line arguments and second is list of command line arguments.

```
int main (int argc , char * arg[ ]
{ /*_ _ _
    _ _ _ */ }.
```

→ Properties of command line arguments :

1) They are passed to main() function.

2) They are parameters supplied to the program when it is invoked.

3) argv [argc] is a NULL pointer

4) argv[0] holds the name of the program.

5) argv[1] points to the first command line argument and argv[n] points last argument.

6) They are used to control program from outside instead of hard coding those values inside the code.

24. Various ios format :

1) width() : The width method is used to set the required field width. The output will be displayed in the given width.

Syntax : width(int width);

2) precision() : The precision method is used to set the number of the decimal point to a float value.

Syntax : precision(int num_of_digits);

3) fill() : The fill method is used to set a character to fill in the blank space of a field.

Syntax : fill (char ch) ;

4) setf() : The setf method is used to set various flags for formatting output.

Syntax : setf (arg1 , arg2) ;

5) unsetf() : The unsetf method is used to remove the flag setting.

Syntax : void unsetf (fmtflags mask) ;

25. get() and put() functions :-

→ Class istream and ostream define two member functions get() and put() which can handle a single character input-output operations.

→ Two types of get() function include get(char*) and get(void) prototypes which can be used to fetch a character including the blank space, tab and a newline character.

→ The get(char*) assigns the input character to its argument and the get(void) returns the input character.

→ As both functions are members of input-output

stream classes, one can invoke them using a suitable object only.

→ Example :- Illustrating get() and put() functions :-

```
#include <fstream.h>
#include <string.h>
#include <conio.h>
#include <iostream.h>
using namespace std;

int main()
{
  char str[] = " DEPSTAR ";
  char ch;
  cout << " Using get() and put() functions... "
       << endl;
  ofstream file_out("test.txt");
  for(int i=0; i<strlen(str); i++)
  {
    file_out.put(str[i]);
  }
  file_out.close();
  ifstream file_in("test.txt");
  while(file_in)
  {
    file_in.get(ch);
    cout << ch;
  }
```

```
    return 0;
}
```

→ Output :-

```
Using get() and put() functions...
/* ~~20DCS103 DEPSTAR~~ */
DEPSTAR
```

** getline() and write() functions :-

→ The getline() function reads a whole line text which ends with a newline character.

→ This function can be invoked by using the object cin as follows:

```
cin.getline(line, size)
```

→ For example:

```
char name[20];
cin.getline(name 20);
```

→ The input is "Object Oriented Programming". which will be terminated after reading the following 19 characters, the two blank spaces are also taken into count.

→ The write() function displays an entire line and has the following form.

cout.write(line, size)

→ The first argument line represents the name of the string to be displayed and second argument size indicates the number of characters automatically when the null character is encountered.

→ If the size is greater than the length of line, then it displays beyond the bound of line.

→ Difference :-

→ cin does not read complete string including spaces, string terminates as you input space, whilst cin.getline() is used to read unformatted string from the standard input device. Then function reads complete string.

26. Designing manipulators :-

→ The syntax for designing manipulators is as follows:

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

20DCS103 - Rushik. Rathod

```
ostream & form(ostream & output)
{
    output.setf(ios::showpos);
    output.setf(ios::showpoint);
    output.fill('*');
    output.precision(2);
    output << setiosflags(ios::fixed) <<
              setw(10);
    return output;
}

int main()
{
    cout << form << 7864.5;
    getch();
    return 0;
}
```

27. Manipulators explanations :-

(i) setw() : The setw() manipulator changes the width of the next input/output field. When used in an expression out << setw(n) or in >> setw(n) sets the width parameter of the stream out or in into exactly n.

(ii) setprecision() : The setprecision() manipulator changes floating point precision. when used in an expression out << setprecision(n) or in >> setprecision(n) sets the precision parameter of the stream out or into exactly n.

(iii) setfill() : The setfill() manipulator is used by setw manipulator. If a value does not entirely fill a field, then the character specified in the setfill argument of the manipulator is used for filling the fields.

28. Difference between manipulators and ios member functions in implementation :-

* Manipulators	* ios member functions
→ Manipulators does not return a value.	→ ios member function returns value.
→ We can create our own manipulators.	→ We cannot create our own ios member functions.

→ Manipulators are possible to combine in a chain.	→ ios functions are single and not possibe to combine in a cheim.
→ Manipulators need <iomanip> included.	→ ios functions need <iostream> included.
→ Manipulators are not member functions.	→ ios functions are member functions.

29. Program Code :-

```
#include <iostream>
using namespace std;

class Product
{
   float price;
   int number;
public:
   void getdata()
   {
      cout << "Price : ";
      cin >> price;
      cout << "Number of products : ";
      cin >> number;
   }
```

```
    void putdata()
    {
      cout<< " Price : " << price << " \t " <<
            " Number of products : " << number
            << endl ;
    }
friend Product operator + ( Product , Product ) ;
friend int operator == ( Product , Product ) ;
} ;

Product operator + ( Product t1 , Product t2 )
{
   Product temp ;
   temp.price = t1.price + t2.price ;
   temp.number = t1.number + t2.number ;
   return temp ;
}

int operator== ( Product t1 , Product t2 )
{
  if ( (t1.price == t2.price) && ( t1.number == t2.number) )
      return 1 ;
  else
      return 0 ;
}

int main()
{
  Product P1 , P2 , P3 ;
```

Page No.
Date / /

```
cout << endl << " ***** Enter the information
      for P1 ***** " << endl;
P1.getdata();
cout << endl << " ***** Enter the information
      for P2 ***** " << endl;
P2.getdata();

cout << endl << " \t ------: P1 :------ " << endl;
P1.putdata();
cout << endl << " \t ------: P2 :------ " << endl;
P2.putdata();

P3 = P1 + P2;
cout << endl << " ----- AFTER P3 = P1+P2
      ------ " << endl;
cout << endl << " \t ------: P3 :------ " << endl;
P3.putdata();

if ( P1 == P2 )
   cout << " \n Price and number are same
        for both of the products. " << endl;
else
   cout << " \n Price and number are different
        for both of the products. " << endl;

cout << " \n\n Made By : RUSHIK RATHOD \n
      20DCS103 \n " ;
return 0;
}
```

Page No.
Date / /

30. Program Code :-

```
#include <iostream>
using namespace std;

class Book
{
   string name;
   int pages;
   float price;
public:
    Book()          // default constructor
    {       }
    Book(string tname, int tpages, float tprice)
    {                          // parameterized constructor
        name = tname;
        pages = tpages;
        price = tprice;
    }

    Book(const Book &b)  // copy constructor
    {
       name = b.name;
       pages = b.pages;
       price = b.price;
    }

    void putdata()
    {
```

Page No.
Date / /

```
    cout << "*****  Book Information  *****" << endl;
    cout << "Name   : " << name << endl ;
    cout << "Pages  : " << pages << endl ;
    cout << "Price  : " << price << endl;
    }
    ~Book() { } // Destructor
};

int main()
{
   string bname;
   int bpages;
   float bprice;
   cout << "Enter the information about book..."
        << endl;
   cout << "Name of the book : ";
   cin >> bname;
   cout << "Number of pages : ";
   cin >> bpages;
   cout << "Price of book : ";
   cin >> bprice;

   Book b;
   cout << endl << "----- From parameterized
        constructor -----" << endl;
   Book b1(bname, bpages, bprice);
   b1.putdata();
```

```
    cout << endl << "----- From copy constructor ----
         " << endl;
    Book b2(b1);
    b2.putdata1();
    cout << "\n\n Made By : RUSHIK RATHOD \n
                    20DCS103 \n";

    return 0;
}
```

31. Program Code :

```
#include <iostream>
#define PI 3.14159265
using namespace std;
class shape_3d                    // abstract class.
{
protected :
    double r, h;
public :
    void getdata();
    {
        cout << "Radius : ";
        cin >> r;
        cout << "Height : ";
        cin >> h;
    }
    virtual void display_volume() = 0;
                    // pure virtual function.
};
```

Page No.
Date / /

```
class Cylinder : public Shape_3d
{
public :
    void display_volume()
    {
        double Cylinder_volume ;
        Cylinder_volume = PI*r*r*h ;
        cout << endl << "Volume " << " : " << Cylinder_
                volume << endl ;
    }
};

class Cone : public Shape_3d
{
  public :
    void display_volume()
    {
        double Cone_volume ;
        Cone_volume = (PI*r*r*h)/3 ;
        cout << endl << " Volume " << " : " <<
                Cone_volume << endl ;
    }
};

int main()
{
   Shape_3d *sptr[2] ;
   Cylinder r[2] ;
```

Page No.
Date / /

```
cout << endl << " ***** Enter The Information
      For 2 Cylinders ***** " << endl;
int x = 1;
for(int i=0; i<2; i++)
{
  cout << endl << " ----- Cylinder : " << x <<
       " ----- " << endl;
  sptr[i] = &r[i];
  sptr[i] -> getdata();
  sptr[i] -> display_volume();
  x++;
}

Shape_3d *sp[3];
Cone n[3];
cout << endl << " ***** Enter The Information
      For 3 Cones ***** " << endl;
int y = 1;
for(int i=0; i<3; i++)
{
  cout << endl << " ----- Cone : " << y << " ----- "
       << endl;
  sp[i] = &n[i];
  sp[i] -> getdata();
  sp[i] -> display_volume();
  y++;
}
cout << "\n\n Made By : RUSHIK RATHOD \n
      20DCS103 \n";
```

```
    return 0;
}
```

32. Program Code :-

```
#include <iostream>
using namespace std;

class Staff
{
   string name;
   int emp_code;
public :
   void getdata()
   {
      cout << endl << "***** Enter The Information *****" << endl;
      cout << "Employee name : ";
      cin >> name;
      cout << "Employee code : ";
      cin >> emp_code;
   }
   void putdata()
   {
      cout << endl << "----- Entered Information -----" << endl;
      cout << "Name : " << name << endl;
      cout << "Code : " << emp_code << endl;
   }
};
```

20DCS103 - Rushik. Rathod

Page No.
Date

```
class Typist : public Staff
{
  float speed;
public:
    void get()
    {
        getdata();
        cout << "Typing speed : ";
        cin >> speed;
    }
    void put()
    {
        putdata();
        cout << "Speed : " << speed << endl;
    }
};

int main()
{
    Typist t1;
    t1.get();
    t1.put();
    cout << "\n\n Made By: RUSHIK RATHOD \n
                20DCS103 \n";
    return 0;
}
```

33. Program Code :-

```
#include <iostream>
using namespace std;

class Medicine
{
    string medicine_type;
    string company_name;
    string many_date;
public:
    void getdata()
    {
        cout << " Enter medicine type : ";
        cin >> medicine_type;
        cout << " Enter company name : ";
        cin >> company_name;
        cout << " Enter manufacturing date : ";
        cin >> many_date;
    }
    void putdata()
    {
        cout << " Medicine type : " << medicine_type
             << endl;
        cout << " Company name : " << company_name
             << endl;
        cout << " Manufacturing date : " << many_date
             << endl;
    }
};
```

20DCS103 - Rushik. Rathod

```
class Tablet : public Medicine
{
   string tab_name;
   int tab_quantity;
   float tab_price;
public:
    void input()
    {
      cout << endl << " -------- Enter The Information
            About Tablet --------" << endl;
      getdata();
      cout << " Enter the name of tablet : " ;
      cin >> tab_name;
      cout << " Enter the quantity per pack : " ;
      cin >> tab_quantity;
      cout << " Enter the price of one tablet : ";
      cin >> tab_price;
    }
    void output()
    {
      cout << endl << "\n -------- Entered Information
            About Tablet --------" << endl;
      putdata();
      cout << " Name of tablet      :" << tab_name
           << endl;
      cout << " Quantity per pack   :" << tab_quantity
           << endl;
      cout << " Price of one tablet : " <<
            tab_price << endl;
```

```
    }
};

class Syrup : public Medicine
{
  int syrup_quantity;
  int syrup_dosage;
public:
  void gdata()
  {
    cout << endl << "\n-------- Enter The
        Information About Syrup --------" << endl;
    getdata();
    cout << "Enter the quantity per bottle : ";
    cin >> syrup_quantity;
    cout << "Enter the dosage unit : ";
    cin << syrup_dosage;
  }
  void pdata()
  {
    cout << endl << "\n-------- Entered Information
        About Syrup --------" << endl;
    putdata();
    cout << "Quantity per bottle : " <<
        syrup_quantity << endl;
    cout << "Dosage unit : " <<
        syrup_dosage << endl;
  }
};
```

Page No.
Date / /

```
int main()
{
    Tablet t1;
    t1.input();
    t1.output();
    Syrup s1;
    s1.gdata();
    s1.pdata();
    cout << "\n\n Made By: RUSHIK RATHOD \n
                20DCS103 \n" ;

    return 0;
}
```

34. Program Code :-

```
#include <iostream>
using namespace std;

class Inches;
{
class CM
{
   float cm;
public:
   CM()                    // default constructor
   {  cm = 0;  }

   CM( Inches i );
```

```
    void getc()
    {
        cout<< " Enter cm : ";
        cin >> cm;
    }
    void putc()
    {
      cout << endl << "cm : " << cm << endl;
    }
};

class Inches
{
    float inch;
public:
    Inches()                    // default constructor
    {   inch = 0;   }
    void geti()
    {
        cout << endl << " Enter inches : ";
        cin >> inch;
    }
    void puti()
    {
        cout << " inches : " << inch << endl;
    }
    float return_inch()
    {
      return inch;
    }
```

Page No.
Date / /

```
};

CM :: CM (Inches i)
{
   cm = 2.54 * (i.return_inch());
}

int main()
{
   CM c;
   C.getc();
   Inches IN;
   IN.geti();

   cout << endl << "----- Entered Information
        -----" << endl;
   C.putc();
   IN.puti();

   C=IN;
   cout << endl << "-------- After C=IN ----
        ----" << endl;
   C.putc();
   IN.puti();
   cout << "\n\n Made By: RUSHIK RATHOD \n
                20DCS103 \n";

   return 0;
}
```

Page No.
Date / /

35. Program Code :-

```
#include <iostream>
using namespace std;

class Height
{
    float cm;
public:
    Height()                    // default constructor
    {   cm = 0;   }
    void getdata()
    {
      cout << "Enter centimeter for H2 : ";
      cin >> cm;
    }
    void putdata()
    {
      cout<<" Centimeter : "<<cm<< endl;
    }
    friend Height operator+(float, Height);
};

Height operator+(float f, Height h)
{
    Height temp;
    temp.cm = f + h.cm;
    return temp;
}
```

```
int main()
{
    Height H1, H2;
    cout << endl << "***** Enter centimeter
         for H2 *****" << endl;
    H2.getdata();
    cout << endl << "Entered Value..." << endl;
    H2.putdata();

    H1 = 2.0 + H2;
    cout << endl << "-----: After H1 = 2.0+H2 :-
         ----" << endl;
    cout << endl << "----- H1 -----" << endl;
    H1.putdata();
    cout << endl << "----- H2 -----" << endl;
    H2.putdata();
    cout << "\n\n Made By: RUSHIK RATHOD
                 20DCS103 \n";

    return 0;
}
```

36. Program Code :-

```
#include <iostream>
using namespace std;

class Num
{
```

Page No.
Date / /

```
    int a, b;
public:
    void getdata() -
    {
        cout << "Enter a : ";
        cin >> a;
        cout << "Enter b : ";
        cin >> b;
    }
    void putdata()
    {
        cout << " a : " << a << "\t" << " b : "
             << b << endl;
    }
    friend Num operator - ( Num, Num );
    friend int operator == ( Num, Num );
};

Num operator - (Num x, Num y)
{
    Num temp;
    temp.a = x.a - y.a;
    temp.b = x.b - y.b;
    return temp;
}

int operator == ( Num n1, Num n2 )
{
```

```
    if((n1.a == n2.a) && (n1.b == n2.b))
        return 1;
    else
        return 0;
}

int main()
{
    Num N1, N2, N3;
    cout << endl << "***** Enter value for
        N2 ***** " << endl;
    N2.getdata();
    cout << endl << "***** Enter value for
        N3 ***** " << endl;
    N3.getdata();

    cout << endl << "\t N2" << endl;
    N2.putdata();
    cout << endl << "\t N3" << endl;
    N3.putdata();

    N1 = N2 - N3;
    cout << endl << "\n ----- AFTER N1 = N2 - N3 ---
        --" << endl;
    cout << endl << "\t N1" << endl;
    N1.putdata();

    Num N4, N5;
```

20DCS103 - Rushik. Rathod

```
cout << endl << " ***** Enter value for
      N4 ***** " << endl;
N4.getdata() ;
cout << endl << " ***** Enter value for
      N5 ***** " << endl;
N5.getdata() ;

cout << endl << " \t N4 " << endl;
N4. putdata() ;
cout << endl << " \t N5 " << endl;
N5. putdata() ;

cout << endl << " \n ------ AFTER N4 == N5
      ------ " << endl;

if (N4 == N5)
    cout << endl << " Both numbers are
          same. " << endl;
else
    cout << " endl << " Both numbers are
          different. " << endl;

cout << " \n\n Made By: RUSHIK RATHOD
      \n 20DCS103 \n " ;
return 0;
}.
```

Page No.
Date / /

37 Program Code :-

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int count = -1;
    ofstream op;
    op.open("my_file.txt");
    string str;
    cout << "Enter a word : ";
    cin >> str;
    op << str;
    op.close();
    ifstream fp;
    char ch;
    fp.open("my_file.txt");

    while (fp.eof() == 0)
    {
        fp.get(ch);
        if (ch == ' ' || ch == '\0')
        {
            goto end;
        }
        else
        {   count++;   }
    }
```

```
    end:
    cout << "\n A word in the file is : "
         << count << endl;
    fp.close();

    cout << "\n\n Made By: RUSHIK RATHOD
            \n 20DCS103 \n";
    return 0;
}
```

38. Program Code :

```
#include <iostream>
using namespace std;

class Distance
{
    int cm, mm;
public:
    Distance()          // default constructor
    {
        cm = 0;
        mm = 0;
    }
    Distance(int temp)
    {
        mm = temp % 10;
        cm = temp / 10;
    }
```

```
    void putdata()
    {
        [redacted]
        cout << cm << " cm" << " 1to" << mm <<
             " mm" << endl;
    }
};

int main()
{
    Distance D1, D2;
    D2 = 24;
    D2.putdata();

    cout << "\n\n Made By: RUSHIK RATHOD \n
                        20DCS103 \n";

    return 0;
}
```