Lecture Notes

on

Object Oriented Programming

3CS4-06

Unit II

Department of Computer Science & Engineering
Jaipur Engineering College & Research Centre, Jaipur

By: Sweety Singhal & Priyanka Mitra (Dept. of CSE, JECRC)

Vision of the Institute

To become a renowned centre of outcome based learning and work toward academic, professional, cultural and social enrichment of the lives of individuals and communities.

Mission of the Institute

M1: Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.
M2: Identify, based on informed perception of Indian, regional and global needs, the areas of focus and provide platform to gain knowledge and solutions.
M3: Offer opportunities for interaction between academia and industry.
M4: Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

Vision of the Department

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

Mission of the Department

M1: To impart outcome based education for emerging technologies in the field of computer science and engineering.
M2: To provide opportunities for interaction between academia and industry.
M3: To provide platform for lifelong learning by accepting the change in technologies.
M4: To develop aptitude of fulfilling social responsibilities.

Program Outcomes (PO)

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and Computer Science & Engineering specialization to the solution of complex Computer Science & Engineering problems.
2. **Problem analysis**: Identify, formulate, research literature, and analyze complex Computer Science and Engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex Computer Science and Engineeringproblems and design system components or processes that meet the specified needs with appropriateconsideration for the public health and safety,and the cultural, societal, and environmentalconsiderations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of Computer Science and Engineering experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex Computer Science Engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional Computer Science and Engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional Computer Science and Engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the Computer Science and Engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in Computer Science and Engineering.
10. **Communication**: Communicate effectively on complex Computer Science and Engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the Computer Science and Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation andability to engage in independent and life-long learning in the broadest contextof technological changein Computer Science and Engineering.

Program Educational Objectives (PEO)

PEO1: To provide students with the fundamentals of Engineering Sciences with more emphasis in Computer Science & Engineering by way of analyzing and exploiting engineering challenges.
PEO2: To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems inComputer Science and Engineering
PEO3: To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues for Computer Science & Engineering.
PEO4: To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self-motivated life-long learning needed for a successful professional career in Computer Science & Engineering.
PEO5: To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge in Computer Science & Engineering.

Course Outcomes

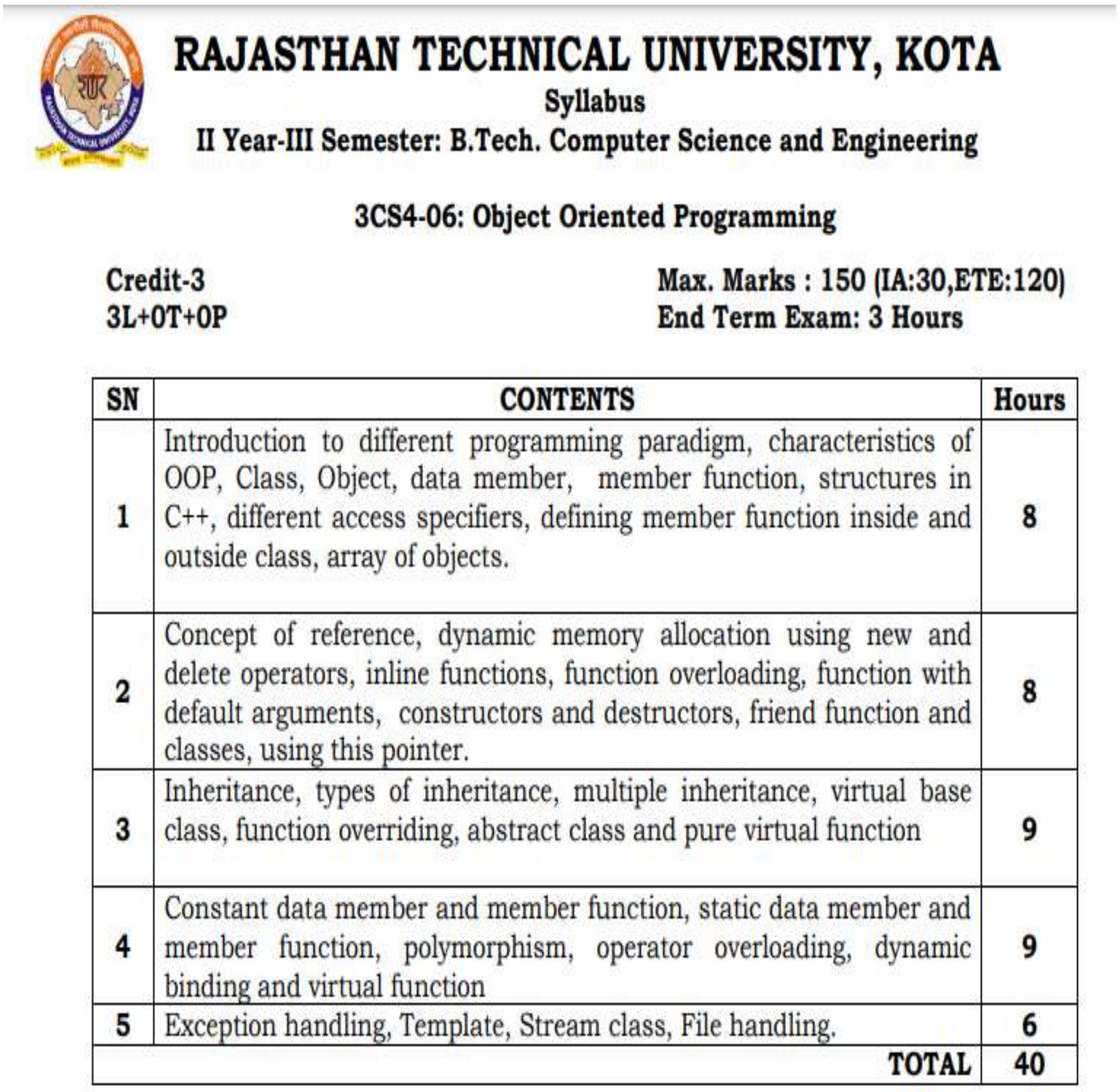
1. Understand the paradigms of object oriented programming in comparison of procedural oriented programming.
2. Apply the class structure as fundamental, building block for computational programming.
3. Apply the major object-oriented concepts to implement object oriented programs in C++.
4. Implement the concept of abstraction inheritance, polymorphism, dynamic binding and generic structure in building reusable code.

Mapping of Course Outcomes with Program Outcomes

H=3, M=2, L=1

Semester	Subject	Code	L/T/P	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
III	Object Oriented Programming	3CS4-06	L	Understand the paradigms of object oriented programming in comparison of procedural oriented programming.	3	3	3	2	2	2	1	1	0	1	1	3
			L	Apply the class structure as fundamental, building block for computational programming.	3	3	3	3	2	2	1	1	1	2	1	3
			L	Apply the major object-oriented concepts to implement object oriented programs in C++.	3	3	3	3	2	1	2	2	1	2	1	3
			L	Implement the concept of abstraction inheritance, polymorphism, dynamic binding and generic structure in building reusable code.	3	3	3	2	1	1	1	1	1	1	1	3

Syllabus



RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

II Year-III Semester: B.Tech. Computer Science and Engineering

3CS4-06: Object Oriented Programming

Credit-3 — **Max. Marks : 150 (IA:30,ETE:120)**

3L+0T+0P — **End Term Exam: 3 Hours**

SN	CONTENTS	Hours
1	Introduction to different programming paradigm, characteristics of OOP, Class, Object, data member, member function, structures in C++, different access specifiers, defining member function inside and outside class, array of objects.	8
2	Concept of reference, dynamic memory allocation using new and delete operators, inline functions, function overloading, function with default arguments, constructors and destructors, friend function and classes, using this pointer.	8
3	Inheritance, types of inheritance, multiple inheritance, virtual base class, function overriding, abstract class and pure virtual function	9
4	Constant data member and member function, static data member and member function, polymorphism, operator overloading, dynamic binding and virtual function	9
5	Exception handling, Template, Stream class, File handling.	6
	TOTAL	**40**

UNIT II

FUNCTION OVERLOADING:

Overloading refers to the use of the same thing for different purposes . C++ also permits overloading functions .This means that we can use the same function name to creates functions that perform a variety of different tasks. This is known as function polymorphism in oops.
Using the concepts of function overloading , a family of functions with one function name but with different argument lists in the functions call .The correct function to be invoked is determined by checking the number and type of the arguments but not on the function type.

For example an overloaded add() function handles different types of data as shown below.

```
//Declaration
        int add(int a, int b); //prototype 1
        int add (int a, int b, int c); //prototype 2
        double add(double x, double y); //prototype 3
        double add(double p , double q); //prototype4



//function call
cout<<add(5,10); //uses prototype 1
cout<<add(15,10.0); //uses prototype 4
cout<<add(12.5,7.5); //uses prototype 3
cout<<add(5,10,15); //uses prototype 2
cout<<add(0.75,5); //uses prototype 5
```

A function call first matches the prototype having the same no and type of arguments and then calls the appropriate function for execution.

The function selection invokes the following steps:-

a) The compiler first tries to find an exact match in which the types of actual arguments are the same and use that function.
b) If an exact match is not found the compiler uses the integral promotions to the actual arguments such as:

char to int
float to double
to find amatch

c) When either of them tails ,the compiler tries to use the built in conversions to the actual arguments and them uses the function whose match is unique . If the conversion is possible to have multiple matches, then the compiler will give errormessage.

Example:

```
long square (long n);
double square(double x);
```

A function call suchas:- square(lO)

Will cause an error because int argument can be converted to either long or double .There by creating an ambiguous situation as to which version of square( )should be used.

PROGRAM

```
#include<iostream.h>
int volume(double,int);
double volume( double , int );
double volume(longint ,int ,int);
        main( )
        {
                cout<<volume(10)<<endl;
                cout<<volume(10)<<endl; cout<<volume(10)<<endl;
        }
int volume( ini s)
    {
    return (s*s*s); //cube
     }
double volume( double r, int h)
        {
        return(3.1416*r*r*h); //cylinder
        }
long volume (longint 1, int b, int h)
        {
        return(1*b*h); //cylinder
        }

                output:- 1000
                         157.2595
                         112500
```

FRIEND FUNCTIONS:-

We know private members can not be accessed from outside the class. That is a non - member function can't have an access to the private data of a class. However there could be a case where two classes manager and scientist, have been defined we should like to use a function income- tax to operate on the objects of both theseclasses.
In such situations, c++ allows the common function lo be made friendly with both the classes , there by following the function to have access to the private data of these classes .Such a function need not be a member of any of these classes.
To make an outside function "friendly" to a class, we have to simply declare this function as a friend of the classes as shown below :

```
class ABC
{
---------
---------
public:
        --------
        ----------
        friend void xyz(void);
};
```

The function declaration should be preceded by the keyword friend , The function is defined else where in the program like a normal C ++ function . The function definition does not use their the keyword friend or the scope operator **::** . The functions that are declared with the keyword friend are known as friend functions. A function can be declared as a friend in any no of classes. A friend function, as though not a member function , has full access rights to the private members of the class.

A friend function processes certain special characteristics:

a. It is not in the scope of the class to which it has been declared asfriend.
b. Since it is not in the scope of the class, it cannot be called using the object of that class. It can be invoked like a member function without the help of anyobject.
c. Unlike memberfunctions.

Example:

```
#include<iostream.h>
          class sample
          {
                    int a;
                    int b;
          public:
                    void setvalue( ) { a=25;b=40;}
                    friend float mean( sample s);
          }
           float     mean (samples)
                    {
                              return (float(s.a+s.b)/2.0);
                    }
 int main ( )
     {

          sample x;
          x . setvalue( );
          cout<<"mean value="<<mean(x)<<endl;
          return(0);

     }
```

output:
```
mean value : 32.5
```

A function friendly to two classes

```
        #include<iostream.h>
            class abc;
            class xyz
            {
                int x;
            public:
                void setvalue(int x) { x-= I; }
                friend void max (xyz,abc);
            };
            class abc
            {
                int a;
            public:
                void setvalue( int i) {a=i; }
                friend void max(xyz,abc);
            };


            void max( xyz m, abc n)
            {
                if(m . x >= n.a)
                    cout<<m.x;
                else
                    cout<< n.a;
            }

            int main( )
            {
            abc j;
            j . setvalue( 10);
            xyz s;
            s.setvalue(20);
            max( s , j );
            return(0);
            }
```

SWAPPING PRIVATE DATA OF CLASSES:

```
    #include<iostream.h>

            class class-2;
            class class-1
            {
                    int value 1;
            public:
                    void indata( int a) { value=a; }
                    void display(void) { cout<<value<<endl; }
                    friend void exchange ( class-1 &, class-2 &);
            };

            class class-2
            {
                    int value2;
            public:
                    void indata( int a) { value2=a; }
                    void display(void) { cout<<value2<<endl; }
                    friend void exchange(class-l & , class-2 &);
                    };
            void exchange ( class-1 &x, class-2 &y)
                    {
                            int temp=x. value 1;
                            x. value I=y.valuo2;
                            y.value2=temp;
                    }

                    int main( )
                    {
                    class-1 c1;
                    class-2 c2;
                    c1.indata(l00);
                    c2.indata(200);
                    cout<<"values before exchange:"<<endl;
                    c1.display( );
                    c2.display( );
                    exchange (c1,c2);
                    cout<<"values after exchange :"<< endl;
                    c1. display ( );
                    c2. display ( );
                    return(0);
output:             }
```

```
    values before exchange
            100
            200
    values after exchange
            200
            100
```

PROGRAM FOR ILLUSTRATING THE USE OF FRIEND FUNCTION:

```
        #include< iostream.h>
        classaccount1;
        classaccount2
        {
        private:
                int balance;
        public:
        account2( ) { balance=567; }
        void showacc2( )
        {
        cout<<"balanceinaccount2 is:"<<balance<<endl;
friend int transfer (account2 &acc2, account1 &acc1,int amount);
        };
class acount1
{
private:
        int balance;
public:
        account1 ( ) { balance=345; }


        void showacc1 ( )
        {
                cout<<"balance in account1 :"<<balance<<endl;
        }
friend int transfer (account2 &acc2, account1 &acc1 ,int amount);
        };

int transfer ( account2 &acc2, account1 & acc1, int amount)
        {
                if(amount <=accl . bvalance)
                        {
                        acc2. balance + = amount;
                        acc1 .balance - = amount;
                        }
                else
                        return(0);
        }
int main()
{
account1    aa;
account2    bb;



        cout << "balance in the accounts before transfer:" ;
        aa . showacc1();
        bb . showacc2();
        cout << "amt transferred from account1 to account2 is:";
        cout<<transfer ( bb,aa,100)<<endl;
```

```
                        cout<< " balance in the accounts after the transfer:";
                        aa . showacc 1 ( );
                        bb. showacc 2( );
                        return(0);
}
```

```
output:
balance in the accounts before transfer
        balance in account 1 is 345
        balance in account2 is 567
and transferred from account! to account2 is 100
        balance in account 1 is245
        balance in   account2 is667
```

RETURNING OBJECTS:

```
        # include< iostream,h>
                class complex
                {
                        float x;
                        float y;
                public:
                        void input( float real , float imag)
                                {
                                        x=real;
                                        y=imag;
                                }
                        friend complex sum( complex , complex);
                                void    show ( complex);
                };
        complex sum ( complex c1, complex c2)
                        {
                        complex c3;
                        c3.x=c1.x+c2.x;
                        c3.y=c1.y+c2.y;
                        return c3;}



                void complex :: show ( complex c)
                {
                cout<<c.x<<" +j "<<c.y<<endl;
                 }

                intmain( )
                {
                complex a, b,c;
                a.input(3.1,5.65);
                b.input(2.75,1.2);
                c=sum(a,b);
                cout <<" a="; a.show(a);
                cout <<" b= "; b.show(b);
                cout <<" c=" ; c.show(c);
                return(0);
    output:     }
```

```
        a =3.1 + j 5.65
        b= 2.75+ j 1.2
        c= 5.55 + j 6.85
```

POINTER TO MEMBERS;

It is possible to take the address of a member of a class and assign it to a pointer. The address of a member can be obtained by applying the operator & to a “fully qualified” class member name.

A class member pointer can be declared using the operator **::** * with the class name.

For Example:

```
classA
{
private:
        int m;
public:
        void show( );
};
```

We can define a pointer to the member m as follows :

```
int A :: * ip = & A :: m
```

The ip pointer created thus acts like a class member in that it must be invoked with a class object. In the above statement. The phrase A **::** * means “pointer - to - member of a class” . The phrase & A **::** m means the “ Address of the m member of a class”

The following statement is not valid :

```
int *ip=&m ; // invalid
```

This is because m is not simply an int type data. It has meaning only when it is associated with the class to which it belongs. The scope operator must be applied to both the pointer and the member.

The pointer ip can now be used toaccessthe m inside the member function (or friendfunction).

Let us assume that “a” is an object of “ A” declared in a member function . We can access "m" using the pointer ip as follows.

```
cout<< a . * ip;
cout<< a.m;
ap=&a;
cout<< ap-> * ip;
cout<<ap->a;
```

The deferencing operator ->* is used as to accept a member when we use pointers to both the object and the member. The dereferencing operator. .* is used when the object itself is used with the member pointer. Note that * ip is used like a member name.

We can also design pointers to member functions which ,then can be invoked using the deferencing operator in the main as shownbelow.

```
(object-name.*      pointer-to-member      function)
(pointer-to -object -> * pointer-to-memberfunction)
```

The precedence of ( ) is higher than that of .* and ->* , so the parenthesis are necessary.

DEREFERENCING OPERATOR:

```
        #include<iostream.h>
                class M
                {
                        int x;
                        int y;
                public:
                        void set_xy(int a,int b)
                                {
                                        x=a;
                                        y=b;
                                }
                friend int sum(M);
                };

                int sum (M m)
                {
                        int M :: * px= &M :: x; //pointer to member x





                                int M :: * py- & m ::y;//pointer to y
                                M * pm=&m;
                                int s=m.* px + pm->py;
                        }       return(s);
                int main ( )
                        {
                        M m;
                        void(M::*pf)(int,int)=&M::set-xy;//pointer to function set-xy (n*pf)( 10,20);
                        //invokes set-xy
                        cout<<"sum=:"<<sum(n)<<cncil;
                        n *op=&n; //point to object n
                        ( op->* pf)(30,40); // invokes set-xy
                        cout<<"sum="<<sum(n)<<end 1 ;
                        return(0);
        output:         }

                sum= 30
                sum=70
```

CONSTRUCTOR:

A constructor is a special member function whose task is to initialize the objects of its class . It is special because its name is the same as the class name. The constructor is invoked when ever an object of its associated class is created. It is called constructor because it construct the values of data members of theclass.

A constructor is declared and defined as follows:

```
//'class with a constructor
        class integer
        {
                int m,n:
        public:
                integer! void);//constructor declared
                ------------
                ------------
        };
integer :: integer(void)
        {
                m=0;
                n=0;
        }
```

When a class contains a constructor like the one defined above it is guaranteed that an object created by the class will be initialized automatically.

For example:-

```
Integer int1; //object int 1 created
```

This declaration not only creates the object int1 of type integerbutalso initializes its data members m and n tozero.

A constructor that accept no parameter is called the default constructor. The default constructor for class A is A :: A( ). If no such constructor is defined, then the compiler supplies a default constructor.

Therefore a statement such as :-

```
A  a  ;//invokes the default constructor of thecompilerof       the
   compiler to create the object "a";
```

Invokes the default constructor of the compiler to create the object a.

The constructor functions have some characteristics:-

- They should be declared in the public section.
- They are invoked automatically when the objects arecreated.
- They don't have return types, not even void and therefore they cannot returnvalues.
- They cannot be inherited , though a derived class cancall

the base class constructor .

- Like other C++ function , they can have defaultarguments,
- Constructor can't bevirtual.
- An object with a constructor can't be used as a member of union.

Example of default constructor:

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

        class abc
        {
        private:
                char nm[];
        public:
                abc ( )
                {
                        cout<<"enter your name:";
                        cin>>nm;
                }
                void display( )

                        {
                                cout<<nm;
                        }

                        };


                        int main( )
                        {
                        clrscr( );
                        abc d;
                        d.display();
                        getch( );
                        return(0);
                        }
```

PARAMETERIZED CONSTRUCTOR:-

the constructors that can take arguments are called parameterized constructors. Using parameterized constructor we can initialize the various data elements of different objects with different values when they are created.

Example:-

```
class integer
{
        int m,n;
public:
        integer( int x, int y);
                --------
                ---------
        };
```

```
integer:: integer (int x, int y)
                {
                        m=x;n=y;
                }
```

the argument can be passed to the constructor by calling the constructor implicitly.

```
integer int 1 = integer(0,100); // explicit call
integerint1(0,100);     //implicitecall
```

CLASS WITH CONSTRUCTOR:-

```
#include<iostream.h>
class integer
{
        int m,n;
public:
        integer(int,int);
        void display(void)




        {
                cout<<"m=:"<<m ;
                cout<<"n="<<n;
        }
};
        integer :: integer( int x,int y) // constructor defined
                {
                m=x;
                n=y;
                }
        int main( )
        {
                integer int1(0,100);     // implicit call
                integerint2=integer(25,75);
                cout<<" \nobject1"<<endl;
                        int1.display();
                        cout<<" \n object2 "<<endl;
                        int2.display();
        }
```

output:

```
        object 1
        m=0
        n=100
        object2
        m=25
        n=25
```

Example:-

```
            #include<iostream.h>
            #include<conio.h>
            class abc
            {
            private:
                    char nm [30];
                    int age;
            public:
                    abc ( ){ }// default
                    abc ( char x[], int y);
                    void get( )
                    {
                    cout<<”enter your name:”;
                    cin>>nm;
                    cout<<” enter your age:”;
                    cin>>age;
                    }
            void display( )
            {
                    cout<<nm«endl;
                    cout«age;
            }
    };
            abc : : abc(char x[], int y)
                    {
                            strcpy(nm,x);
                            age=y;
                    }
    void main( )
    {
    abc 1;
    abc m=abc("computer",20000);
    l.get();
    l.dispalay( );
    m.d isplay ();
    getch( );
    }
```

OVERLOADED CONSTRUCTOR:-

```
#include<iostream.h>
#include<conio.h>
        class sum
        {
        private;
                int a;
                int b;
                int c;
                float d;
                double e;
        public:
                sum ( )
{
cout<<"enter a;";
cin>>a;
cout<<"enter b;";
cin>>b;
cout<<"sum= "<<a+b<<endl;
        }
        sum(int a,int b);
        sum(int a, float d,double c);
        };
        sum :: sum(int x,int y)
                {
                        a=x;
                        b=y;
                }
        sum :: sum(int p, float q ,double r)
                {
                        a=p;
                        d=q;
                        e=r;
                }
        void main( )
        {
        clrscr( );
        sum 1;
        sum m=sum(20,50);
        sum n= sum(3,3.2,4.55);
        getch();
        }
```

```
output:
enter a : 3
enter b : 8
sum=11
sum=70
sum=10.75
```

COPY CONSTRUCTOR:

A copy constructor is used to declare and initialize an object from another object.

Example:-

the statement

```
integer 12(11);
```

would define the object 12 and at the same time initialize it to the values of 11.

Another form of this statement is : integer 12=11;

The process of initialization through a copy constructor is known as copy initialization.

Example:-

```
#incliide<iostream.h>
        class code
        {
                int id;
```

```
                    public
                              code ( ) { } //constructor
                              code (int a) { id=a; } //constructor
                              code(code &x)
                              {
                                        Id=x.id;
                              }
                              void display( )
                              {
                              cout<<id;
                              }
                    };
          int main( )
          {
          code A(100);
          code B(A);
          code C=A;
          code D;
          D=A;
          cout<<" \n id of A :"; A.display( );
          cout<<" \nid of B :"; B.display( );
          cout<<" \n id of C:"; C.display( );
          cout<<" \n id of D:"; D.display( );
          }
```

```
output :-
          id of A:100
          id of B:100
          id of C:100
          id ofD:100
```

DYNAMIC CONSTRUCTOR:-

The constructors can also be used to allocate memory while creating objects . This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size, thus resulting in the saving of memory.

Allocate of memory to objects at the time of their construction is known as dynamic constructors of objects. The memory is allocated with the help of new operator.

Example:-

```
#include<iostream.h>
#include<string.h>
class string
{
          char *name;





                    int    length;
          public:
                    string ()
                    {
                    length=0;
                    name= new char [length+1]; /* one extra for \0 */
                    }
          string( char *s) //constructor 2
                    {
                    length=strlen(s);
                    name=new char[length+1];
                    strcpy(name,s);
                    }
          void display(void)
          {
                    cout<<name<<endl;
          }
          void join(string &a .string &b)
                    {
                    length=a. length +b . length;
                    delete name;
                    name=new char[length+l]; /* dynamic allocation */
                    strcpy(name,a.name);
                    strcat(name,b.name);
                    }
                    };
          int  main()
          {
          char * first = “Joseph” ;
          string name1(first),name2(“louis”),naine3( “LaGrange”),sl,s2;
          sl.join(name1,name2);
          s2.join(s1,name3);
          namel.display( );
          name2.display( );
          name3.display( );
          s1.display();
          s2.display();
          }
          output :-
                    Joseph
                    Louis
                    language
                    Joseph Louis
                    Joseph Louis Language
```

DESTRUCTOR:-

A destructor, us the name implies is used to destroy the objects that have been created by a constructor. Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by atilde.

For Example:-

```
~ integer( ) { }
```

A destructor never takes any argument nor does it return any value. It will be invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer accessible. It is a good practice to declare destructor in a program since it releases memory space for future use.

Delete is used to free memory which is created by new.

Example:-

```
matrix : : ~ matrix( )
{
        for(int i=0; i<11;i++)
                delete p[i];
                delete p;
}
```

IMPLEMENTATAION OF DESTRUCTORS:-

```
#include<iostream.h>
        int count=0;
        class alpha
        {
        public:
                alpha( )
                {
                count ++;
                cout<<"\n no of object created :"<<endl;
                }
                ~alpha( )
                {
                        cout<<"\n no of object destroyed :" <<endl;
                        coutnt--;
                }
        };



int  main()
{

cout<<" \n \n enter main \n:";
alpha A1,A2,A3,A4;
{
        cout<<" \n enter block 1 :\n";
```

```
                    alpha A5;
          }
          {
          cout<<" \n \n enter block2 \n";
          alphaA6;
          }
cout<<\n re-enter main \n:";
          return(0);
}
```

```
          output:-
enter main
no of object created 1
no of object created 2
no of object created 3
no of object created 4
enter block1
no of object created 5
no of object destroyed 5
enter block2
no of object created 5
no of object destroyed 5
re-enter main
no of object destroyed 4
no of object created 3
no of object created 2
no of object created1
```

Example :-

```
          #include<iostream.h>
          int x=l;
          class abc
          {
          public:
                    abc( )
                    {
                              x--;
                              cout<<"construct the no"<<x<<endl;
                    }
                    ~abc( )
                    {
                    cout<<"destruct the no:"<<x<<endl;
                           x--;
                                        }
                                        };
                              int main( )
                              {
                              abc I1,I2,I3,I4;
                              cout«ll«12«13«l4«endl;
                              return(0);
                              }
```

New & Delete Operators

Dynamic memory allocation means creating memory at runtime. For example, when we declare an array, we must provide size of array in our source code to allocate memory at compile time.

But if we need to allocate memory at runtime me must use new operator followed by data type. If we need to allocate memory for more than one element, we must provide total number of elements required in square bracket[ ]. It will return the address of first byte of memory.

Syntax of new operator

```
ptr = new data-type;
//allocte memory for one element

ptr = new data-type [ size ];
//allocte memory for fixed number of element
```

Delete operator is used to deallocate the memory created by new operator at run-time. Once the memory is no longer needed it should by freed so that the memory becomes available again for other request of dynamic memory.

Syntax of delete operator

```
delete ptr;
//deallocte memory for one element

delete[] ptr;
//deallocte memory for array
```

Example of c++ new and delete operator

P.T.O

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

P.T.O

```
void main()
{

        int size,i;
        int *ptr;

        cout<<"\n\tEnter size of Array : ";
        cin>>size;

        ptr = new int[size];
        //Creating memory at run-time and return first byte of address to ptr.
  for(i=0;i<5;i++)      //Input arrray fromuser.
  {
        cout<<"\nEnter any number : ";
        cin>>ptr[i];
  }
  for(i=0;i<5;i++)       //Output arrray toconsole.
  cout<<ptr[i]<<", ";
  delete[] ptr;
  //deallocating all the memory created by new operator

}
```

Output :

```
                Enter size of Array : 5
                Enter any number : 78
                Enter any number : 45
                Enter any number : 12
                Enter any number : 89
                Enter any number : 56

                78, 45, 12, 89, 56,
```