# OBJECT ORIENTED PROGRAMMING WITH C++ -18BIT23C

**TEXT BOOK:**

**“Object oriented programming with ANSI and Turbo C++” Pearson Education publications, 2006 .Ashok N Kamthane.**

**PREPARED BY DR.M.SORANAMAGESWARI**

# Object oriented programming with C++

**UNIT-II:**

Class and object: Declaring objects- Defining member functions- Static Member functions- Array of object- friend functions- Overloading member functions-Bit Fields and class- Constructor and Destructor: Characteristics – calling Constructor and Destructors- Constructor and Destructors with static member

# Class and object

- The class is used pack data and function.
- The class Can be accessed only through objects .
- The member variable and function are divided into two section:

*private

*public

- The object can not be  access the private variables and function directly.
- It can be accessed only by the public member function.

**Syntax:**

```
class class-name
{
Private:
        Declaration of variables;
        Declaration of functions;
Public:
        declaration of variables;
        declaration of functions;
};
```

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class add
{
Private;
int x,y,z;
Public:
Void sum()
{
Cin>>x>>y;
z=x+y;
}
```

```
void display()
{
cout<<z;
}
};
void main()
{
add a1;
a1.sum();
a1.display();
getch();
}
```

**Declaring object:**

- The declaration of object is same as declaration of variables.
- Object are created when the memory is allocated.

**Accessing class members:**

- The object can be accessed the public member variable and function using the operator.

**Syntax:**

Object name[operator] member name;

**Example:**

a1.add();
a1.add();

# Public keyword

- The member variable and function can be accessed when it is declared as public.

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class add
{
Public:
int x,y,z;
void sum()
{
z=x+y;
}
void display()
{
cout<<z;
} };
```

```
void main()
{
add a1.;
a1.x=10;
a1.y=20;
a1.sum();
a1.display();
getch();
}
};
Void add::sum()
{
cin>>x>>y;
z=x+y;
}
```

```
void add::display()
{
count<<z;
}
void main()
{
add a1.;
a1.sum();
a1.display();
getch();
}
```

# Member function inside the class

- Member function inside the class can be declared in public (or) private section.
- The public member function can access the  private member of the same class.
- The member function that can be declared as public can be accessed outside the class.

```
#include<iostream.h>
#include<conio.h>
Class student
{
Private:
        int sno,m1,m2,m3,Total;
        char sname[20];
Public:
        void getstudent()
{
        Cout<<“Enter the student number”;
```

```
cin>>sno;
cout<<“Enter the student name”;
cin>>sname;
cout<<“Enter the marks”;
cin>>m1>>m2>>m3;
}
void total_mark()
{
total=m1+m2+m3;
}
```

```
void print _details()
{
cout<<“  student number :”;
cout<<sno;
cout<<“ student name :”;
cout<<sname;
cout<<“ marks :”;
cout<<m1<<m2<<m3;
cout<<total;
}
}
void main()
{
student s1;
s1.getstudent();
s1.total _mark();
s1.print _details();
getch();
}
```

**Private member function:**

- It is also possible to declare private function inside the class.
- The private member function can be accessed only by the public member function.
- It can be accessed like a normal function.

**Example:**

```
#include<iostream.h>
#include<conio.h>
{
Private:
        int sno,m1,m2,m3,total;
        Char sname[20];
```

```
void init()
{
        Sno=0,m1=0,m2=0,m3=0,total=0;
}
Public :
        Void get student()
init();
{
Cout<<“Enter the student number”;
cin>>sno;
cout<<“Enter the student name”;
cin>>sno;
cout<<“Enter the student name”;
cin>>sname;
cout<<“enter the marks”;
cin>>m1>>m2>>m3;
}
void total mark()
{
total=m1+m2+m3;
}
```

```
void print details()
{
cout<<“enter the student number”;
cout<<sno;
cout<<“Enter the student name”;
cout<<sname;s
cout<<“enter the marks”;
cout<<m1<<m2<<m3;
cout<<total;
}
}
void main()
{
student s1;
s1.getstudent();
s1.total mark();
s1.print details();
getch();
}
```

**Member function outside the class:**

- The function defined inside the class can be considered as inline function.
- If a function is small it should be defined inside the class.
- If the function is large it must be defined outside the class.

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class student
{
Private:
        int sno,m1,m2,m3,total;
        Char sname[20];
```

```
Public:
        void get_student();
        void total_mark();
        void print_details();
};
void student::get_student()
{
cout<<“Enter the student no”;
cin>>sno;
cout<<“Enter the student name”;
cout<<sname;
cout<<“enter the mark”;
cin>>m1>>m2>>m3;
}
void student::total_mark()
{
total=m1+m2+m3;
}
```

```
void student::print_details()
{
cout<<“ The student no is”;
cout<<sno;
cout<<“The student name is”;
cout<<sname;
cout<<“The marks are”;
cout<<m1<<m2<<m3;
cout<<“The total is”;
cout<<total;
}
};
void main()
{
Student s;
s.get_student();
s.total_marks();
s.print_details();
Getch();
}
```

# Characteristic of Member Function

- The Difference between member and normal function is that normal function can be invoked freely where as the member function can be accessed only by the object of the class.
- The same function can be used in any number of classes.
- The private data (or) private function can be accessed by public Member Function.
- The member function can be accessed one another without using any object (or) . (dot) Operator.

# Outside member function inline

- The inline mechanism reduces the overhead relating to access the member function .
- It provides better efficiency quick execution of function.
- Inline Function similar to Macros .call to inline function in the program places the function code in the caller program.
- This is known as inline expansion.

**Rules for inline function**

- Use inline function rarely.
- Inline function can be used when the member function contains few statement.
- If a function takes more time to executed than it must to declared as inline.

# Example:

```
#include<iostream.h>
#include<conio.h>
Class add
{
Private:
    int a,b,c;
Public:
     Void get _in();
     Void sum();
     Void Print();
};
Void inline add:: get_in()
{
     cin>>a>>b;
}
```

```
Void add:: sum()
{
    c=a+b;
}
Void add:: print()
{
    Cout<<a<<b;
    Cout<<c;
}
Void main()
{
    add a1;
    a1.get_in();
    a1.sum();
    a1.print();
}
```

# Member Function inside the class

```
#include<iostream.h>
#include<conio.h>
Class employee
{
Private:
    int eno,ename,des,sal,bpay,hra,ma,pf;
Public:
    Void get _employee()
{
Cout<<“Enter the employee no”;
Cin>>eno;
Cout<<“Enter the employee name”;
Cin>>ename;
```

```
Cout<<“Enter the designation”;
Cin>>des;
Cout<<“Enter the Salary”;
Cin>>salary;
Cout<<“Enter the basic pay”;
Cin>>bpay;
}
Void total_salary()
{
Salary=hra+bp+ma-pf
}
Void print_details()
{
Cout<<“The employee no is:”;
Cout<<eno;
Cout<<“The employee name is:”;
Cout<<ename;
```

```
Cout<<“Designation is:”;
Cout<<des;
Cout<<“Salary :”;
Cout<<salary;
Cout<<“Basic pay is:”;
Cout<<bpay;
}
};
Void main()
{
Employee E1;
E1.get_employee();
E1.total_salary();
E1.print_details();
getch();
}
```

# Static member Variables & Functions

**Static member Variables:**

- If a member variable is declared as static only one copy of the member is created for the whole class.
- The static is a keyword used preserve the value of the variable.

**Syntax:**

Static<Variable definition>

Static<function definition>

## Example:

Static int a;

Static void display();

int sumnum::a=0;

-Sumnum is a classs name

- The Static member variable is to be defined outside the class declaration.

The reason for defining outside the class

1. The static data member or associated with the class not with the object.
2. The Static data members are stored individually rather than an element of an object.
3. The static data member has to initialize.
4. The memory for static data is allocated only once.

5. Only one copy of static member variable is created for the whole class.

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class number
{
    Static int c;
Public:
    Void count()
{
    ++c;
    cout<<c;
}
};
```

```
int number::C=0;
int main()
{
    number a,b,d;
    a.count();
    b.count();
    d.count();
}
```

## Static member function:

- Like member variables functions can also be declared as Static.
- When a function is static it can access only static member variables and functions of the same class.

- The non-static members are not accessed by the static member functions.
- The static keyword makes the function free from the individual object of the class and its scope global in the class.
- The following rules has to be maintained while declaring the static function

1. Only one copy of the static member is created in the memory for the entire class.
2. Static member function can access only static data members and function.
3. Static member functions can be invoked using class name.

4. It is also possible to invoke static member function using object.

5. When one of the object changes the value of data member variables the effect is visible to all object of the class.

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class number
{
    Static int c;
Public:
    Static Void count()
{
    ++c;
    }
    static void display()
{
    cout<<“value of C :”<<c;
}
};
```

```
int number::c=0;

int main()

{

    number::count();

    number::display();

    number::count();

    number::display();

}
```

# Static Private member Function

- Static member function can also be declared as private.
- The private static function can be access using static public function.

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class bita
{
Private:
    static int c;
    static void count()
Public:
    Static void display()
```

```
{
    count();
    cout<<"\n value of c:"<<c;
}
};
void bita ::c=0;
void main()
{
    clrscr();
    bita::display();
    bita::display();
}
```

Output: value of C:1

value of C:2

# Static public member variable

- The static public member variable can be initialized outside the class and also can be initialized in the main function.
- It can be initialized like a normal variable.

**Example:**

```
#include<iostream.h>
#include<conio.h>
int c=11;
Class bita
{
Public:
Static int c;
};
```

```
int bita::c=22;
void main()
{
clrscr();
int c=33;
cout<<“\n class member c=“<<bita::c;
cout<<“\n global variable c=“<<::c;
cout<<“\n local variable c=“<<c;
}
```

Output:c=22

c=11

c=33

# Static Object

- The object is a composition one or more variables. The keyword static can be used to initialize all class data members to zero.
- Declaring object as static will initialize all the data members to zero.

Example:

```
#include<iostream.h>
#include<conio.h>
Class number
{
    int c,k;
Public:
```

```
void plus()
{
    c=c+2;
    k=k+2;
}
void show()
{
    cout<<c;
    cout<<k;
}
void main()
{
    static number x;
    x.plus();
    x.show();
}
```

Output : c-=2

k=2

# Array of Object

- Array is a collection of similar data types it can be of any data type including user defined data type created by struct,class,type,def declarations.
- We can also create an array of objects.
- The array elements of stored in continuous memory location.
- The arrays can be initialized or accessed like an ordinary array.

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class student
{
    int Sno;
    char sname[20];
Public:
    void get_input();
    void display();
}
```

```
void student::get_input()
{
    cout<<“Enter the student number”;
    cin>>sno;
    cout<<“Enter the student name”;
    cin>>sname;
}
void student::display()
{
cout<<sno;
cout<<sname;
}
```

```
Void main()
{
    student s[10];
     int n, i;
    cout<<“Enter the no.of.students”;
    cin>>n;
for(i=1;i<=n;i++)
{
    s[i].get_input();
}
For(i=1;i<=n;i++)
{
    s[i].display();
}
}
```

# Objects as Function arguments

✓ Like variables objects can also pass to functions.

There are three methods to pass an arguments:

1. **Pass by value**

- In this type a copy of an object is sent to function and assigned to object of calling function .
- Both actual and formal copies are objects are stored in different memory location so changes made in formal objects does not reflect to actual object.

**2. Pass by reference**

Address of the object is implicitly sent to function.

3. **Pass by address**

- Address of the object is explicitly sent to function.
- In pass by reference and pass by address methods the address of the actual object is passed to the function.
- So, duplicating of the object is prevented**.**

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class life
{
    int mfgyr;
    int expyr;
    int yr;
Public:
Void getyrs()
{
    cout<<“/n manufacture year:”;
    cin>>mfgyr;
```

```
Cout<<“\n Expiry year:”;
Cin>>expyr;
}
Void period(life);
};
Void life::period(life yr)
{
    yr=y1.expyr-y1.mfgyr;
    cout<<“Life of the product:”<<yr<<“years”;
}
void main()
{
    clrscr();
    life a1;
    a1.getyrs();
    a1.period(a1);
}
```

# Friend Function

- Any non-member function has no access permission to the private data of the class.
- The private member of the class are accessed only from the member function of that class.
- C++ allows a mechanism in which a non-member function has access permission to the private members of the class.
- This can be done by declaring a non member function friend to the class whose private data is to be accessed.

Here friend is a keyword

➢ The friend function have the following properties:

1. There is no scope restriction for the friend function . So they can be called directly without using objects.
2. Friend function cannot access the member directly . It uses object and dot operator to access the private and public member.
3. By default friendship is not shared that is if class x , if declared as friend of class y. It does not mean y has rights to access private member of class x.
4. Use of friend function is rarely done because it violets rule of encapsulation and data hiding.
5. The function can be declared as public (or) private without changing its meaning.

**Example:**

```
#include<iostream.h>
#include<conio.h>
class first;
class second
{
    int s;
public:
    void get value()
{
    cin>>s;
}
friend void sum(second,first);
};
```

```
class first
{
    int f;
public:
    void get value()
    {
        cin>>f;
    }
friend void sum(second ,first);
};
void sum(second d, first t)
{
    cout<<t.f+d.s;
}
```

```
void main()
{
    first a;
    second b;
    a.get value();
    b.get value();
    sum(b,a);
}
```

# Friend Classes

- It is possible to declared one or more functions as friend function or entire class can be declared as friend class.
- The friend is not transferable or inheritable one class to another .
- The friend classes are applicable when we want to make available private data of a class to another class.

**Example:**

```
#include<iostream.h>
#include<conio.h>
class B
class A
{
    int a;
public:
    void get A()
    {
        a=30;
    }
void show(B);
};
```

```
Class B
{
    int b;
Public:
    void get B()
    {
        b=40;
    }
friend void A:: show(B bb);
};
void  A:: show(B bb)
{
    cout<<a;
    cout<<bb.b;
}
```

```
Void main()
{
A.a1;
B.b1;
a1.getA();
b1.getB();
a1.show(b1);
}
```

# Overloading member function

- Member functions are also overloaded in the same way as ordinary functions.
- Overloading is nothing but one functions is defined with multiple definition with same function name.

**Example:**

#include<iotstream.h>

#include<conio.h>

#include<stdlib.h>

#include<math.h>

Class absv

```
{
Public:
    int num(int);
    double num(double);
};
int absv::num(int x)
{
    int ans;
    ans=abs(x);
    return(ans);
}
double absv:: num(double d)
{
    double ans;
    ans=fabs(d);
    return(ans);
}
```

```
int main()
{
    clrscr();
    absvn;
    cout<<“\n Absolute value of -25 is”<<n.num(-25);
    cout<<“\n Absolute value of -25.1474 is”<<n.num(-25.1474);
    return 0;
}
```

# Constructor & destructor

- When a variable is declared and if it is not initialized it contains garbage value.
- The compiler itself cannot initialize the variable the program explicitly assign a value to the variable.

**CONSTRUCTORS**

- The constructor constructs the object. Constructors are special member functions
- When an object is created constructor is executed.
- The constructor are automatically executed when the objects are created.

- The constructor may contain arguments to pass the values to the function.
- A program may contain one or more constructor.
- Constructors are also called when local or temporary objects of a class are created.

**Characteristics of Constructor**

1. Constructor has the same name as the class name.
2. Constructor is executed when an object is declared.
3. Constructor have neither return value nor void.
4. The main function of constructor is to initialize objects.
5. Constructors are executed implicitly and they can be invoked explicitly.

6. Constructor can have default and can be overloaded.

7. The constructor without arguments is called default construtor.

**DESTRUCTOR**

1. Destructor are special member function used to destroy the object.
2. The destructor is executed at the end of the function or at the end of the program.
3. The destructor can be defined same as class name preceded with ~(tilde) operators.

4. The destructor is automatically executed when an object goes out of scope.
5. It is also invoked when delete operator is used to free the memory allocation.
6. Destructors are not overloaded.
7. The class can have only one destructor.

**Characteristic of destructor**

1. Destructor has the same name as that of the class it belongs to and proceeded by ~(tilde)
2. Like constructor, the destructor does not have return type and not even void.

3.Constructor and destructor cannot be inherited, though a derived class can call the constructor and destructors of the base class.

4. Destructor can be virtual, but constructors cannot.

5.Only one destructor can be defined in the class. The destructors not have any argument.

6. The destructor neither have default values nor can be overloaded.

7. Programmer cannot access addresses of constructors and destructors.

8. TURBO C++ compiler can define constructor and destructor if they have not been explicitly defined. They are also called on many cases without explicit calls in program. Any constructor (or) destructor created by the compiler will be public.

9.Constructor and destructors can make implicit calls to operators new and delete if memory allocation/de-allocation is needed for an object.

10. An object with a constructor or destructor cannot be used as a member of a union.

**Example:**

```
#include<iostream.h>
#include<conio.h>
class sum
{
    int a,b,s;
public:
    sum()
{
    s=0;
}
void getdata()
{
    cin>>a>>b;
}
```

```
void calculate()
{
    s=a+b;
}
void display()
{
cout<<a<<b;
cout<<s;
}
~sum()
{
    cout<<“constructor destroyed”;
}
};
void main()
{
    sum.s1;
    s1.getdata();
    s1.calculate();
    s1.display();
}
```

# Constructors with arguments

- It is also possible to create constructors with arguments and such constructors are called parameterized constructor.
- For this type of constructor it is necessary to pass value to the constructor when the object is created.

**Example:**

```
#include<iostream.h>
#include<conio.h>
Class sum
{
    int a,b,s;
Public:
    sum(int x, int y)
```

```
{
      a=x;
      b=y;
}
Void calculate()
{
      s=a+b;
}
Void display()
{
      cout<<a<<b;
      cout<<s;
}
~sum()
{
      cout<<“Constructor destroyed”;
}
};
Void main()
{
Sum s1=sum(3,4);//explicit
Sum s1(3,4);//implicit
S1.calculate();
S1.display();
}
```

# Overloading Constructor

- It is also possible to overload constructors.
- Constructors may contain arguments or may not contain arguments.
- A class can contain more than one constructor.
- This is known as constructor overloading.
- All constructors are defined with the same name as the class.
- All the constructor contain different number of arguments.

- Depending upon the number of arguments the compiler executes the appropriate constructor.

**Example:**

```
#include<iostream.h>
#include<conio.h>
class sum
{
int a,b;
float c,d;
public:
    sum(int x, int y)
{
        a=x;
        b=y;
}
```

```
Sum(float x1,float y1)
{
    c=x1;
    d=y1;
}
void display()
{
    cout<<a<<b;
    cout<<c<<d;
}
~sum()
{
    cout<<“Constructor destroyed”;
}
};
```

```
void main()
{
    sum s1=sum(3,4);
    s1.display();
    sum s2=sum(5.5,10.5);
    s2.display();
}
```

# Copy Constructor

- The constructor can accept argument of any data type including user-defined data type.
- It is possible to pass reference of object to the constructor.
- This declaration is known as copy constructor.
- In this type a temporary copy of the object is created.
- All copy constructors require one arguments with reference to an object of that class.

- Using copy constructor it is possible for the programmer to declare and initialize one object using reference of another object.
- Whenever the constructor are called the copy of object is created.

**Example:**

```
#include<iostream.h>
#include<conio.h>
class num
{
     int a;
public:
    num(){}
    num(int k)
```

```
{
a=k;
}
num(int x)
{
    a=x.a;
}
void show()
{
    cout<<a;
}
};
void main()
{
    num n(10);
    num(n);
    n.show();
    c.show();
}
```

# Destructors

- It will destroy the object. Destructors is also a special member function like constructor.
- It destroy the class object created by constructor.
- The destructors have the same name as the class name proceeded by ~(tilde)operator.
- For local and non-static object the destructor is executed when the object goes  out of scope. It is not possible to define more than one destructor.
- Destructor releases memory location occur the memory object.

**Example:**

```
#include<iostudio.h>
#include<conio.h>
class num
{
int a;
public:
    num()
{
    cout<<“constructor created”;
}
~num()
{
    cout<<“constructor destroyed”;
}
};
void main()
{
    num n1;
}
```

# Constructor and destructor with static members

- Every object has its own set of data members.
- When a member function is called only copy of data is available to the function.
- Sometimes it is necessary for all the object to share the data field which is common for an object.
- If the member variable is declared has static.
- Only one copy of the member is created for entire class.

- The static member variable used to count number of object for a particular class.

**Example:**

```
#include<iostudio.h>
#include<conio.h>
class man
{
    static int no;
    char name;
    int age;
public:
    man()
{
    no++;
```

```
cout<<no;
}
~man()
{
    --no;
}
};
int man::no=0;
void main()
{
    man A,B,C;
    cout<<“destroy the object”;
}
```