CLASSES & OBJECTS

5.1 Introduction

- A class is a grouping of variables of different data types with function.
- Each variable of a class is called as member variable and function are called as member function.

Example

```
class item
{
private:
    int codeno;
    float price;
    int qty;
    void values;
public:
    void show();
};
void values()
{
cout<<"enter codeno,price,qty";
cin>>codeno>>price>>qty;
}
void show()
{
values();
cout<<codeno<<price<<qty;
}
void main()
{
item a;
a.show();
}
```

- An object can't directly access the member variables & functions declared in private section but it can access those declared in public section.
- The private member of a class can be accessed by public member function of same class

5.2 Declaring objects

Public keyword – it is used to allow an object to access the member variable of a class directly like structure.

Private keyword – it is used to prevent direct access to member variable or member function by the object. By default the class member are private.

Protected keyword – it is same as private. It is frequently used in inheritance

5.3 Defining member function

Program – Write a program to find simple interest using class concept & data should be declared in private section

```
#include<iostream.h>
#include<conio.h>
class interest
{
private:
    float p,i,r;
    float s.i;
    voi in();
public:
    void calc();
    void display();
    void in()
    {
    cout<<"enter p,i,r";
    cin>>p>>i>>r;
    }
    void calc()
    {
    in();
    s.i=p*i*r/100;
    amt= p+s.i;
    }
    void display()
    {
    cout<<"simple interest is";
    }
};
void main()
{
interest i;
i.calc();
i.display();
}
```

5.4 Declaring member function outside the class

```
#include<iostream.h>
#include<conio.h>
class item
{
private:
    int codeno;
    float price;
    int qty;
public:
```

```
        void show(void);
    };

    void item::show()
    {
    cout<<"enter codeno,price,qty";
    cin>>codeno>>price>>qty;
    }
    void main()
    {
    item one;
    one.show();
    }
```

5.5 Static member variable

- Once a data member variable is declared as static only one copy of that member is created for the whole class.

```
#include<iostream.h>
#include<conio.h>
class number
{
static int c;
int k;
public:
        void zero()
        {
        k=0;
        }
        void count()
        {
        ++c; ++k;
        cout<<c<<k;
        }
};
int number::c=0;
void main()
{
number A,B,C;
A.zero;
B.zero;
C.zero;
A.count;
B.count;
C.count;
}
```

5.6 Static member function

- When a function is defined as static it can access only static member variables & functions of same class.
- The non-static member are not available to these function.
- It can also declared in private section but it must invoked using a static public function.

```
#include<iostream.h>
#include<conio.h>
class bita
{
private:
        static int c;
public:
        static void count()
        {
        c++;
        }
        static void display()
        {
        cout<<c;
        }
};
int bita::c=0;
void main()
{
bita::display();
bita::count();
bita::count();
bita::display();
}
```

5.7 Static object

- When an object is declared as static ,all it's data member is initialized to 0;
- The declaration of static object removes garbage of it's data members & initialize them to 0.

```
#include<iostream.h>
#include<conio.h>
class bita
{
private:
    static int c,k;
public:
    void plus()
    {
    c+=2;
```

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

5.8 Object as function arguments

5.8.1 pass-by value

- A copy of an actual object is send to function & assign to a formal argument.
- Both actual and formal copies of objects are stored at different memory location.
- So, changes made to formal objects are not reflected in actual argument.

```
#include<iostream.h>
#include<conio.h>
class life
{
int mfgyr;
float expyr;
int yr;
public:
        void getyr()
        {
        cout<<"enter manufacterer and exp date";
        cin>>mfgyr>>expyr;
        }
        void period(life);
};
void life::period(life y1)
{
yr=y1.expyr-y1.mfgyr;
cout<<"product life="<<yr<<"years";
}
void main()
{
clrscr();
life a1;
a1.getyrs();
```

```
a1.period(a1);
}
```

5.8.2 pass-by reference

- Here, address of actual object is implicitly send to the called function using a reference object.
- As actual object & reference object share the same memory space. So, any change made to reference object will also reflected in the actual object.

```
#include<iostream.h>
#include<conio.h>
class life
{
int mfgyr;
float expyr;
int yr;
public:
    void getyr()
    {
    cout<<"enter manufacterer and exp date";
    cin>>mfgyr>>expyr;
    }
    void period(life &);
};
void life::period(life &y1)
{
yr=y1.expyr-y1.mfgyr;
cout<<"product life="<<yr<<"years";
}
void main()
{
clrscr();
life a1;
a1.getyrs();
a1.period(& a1);
}
```

5.8.3 pass by address

- Here, address of actual object is explicitly send to the called function using a pointer object.
- So, any change made to pointer object will also reflected in the actual object as the pointer object holds the address of actual object.

```
#include<iostream.h>
#include<conio.h>
class life
{
```

```
int mfgyr;
float expyr;
int yr;
public:
    void getyr()
    {
    cout<<"enter manufacterer and exp date";
    cin>>mfgyr>>expyr;
    }
    void period(life *);
};
void life::period(life *y1)
{
yr=y1->expyr-y1->mfgyr;
cout<<"product life="<<yr<<"years";
}
void main()
{
clrscr();
life a1;
a1.getyrs();
a1.period(& a1);
}
```

5.9 Friend Function

- C++ allows a mechanism in which a non-member function has access permission to the private data member of the class
- This can be done by declaring a non-member function as friend to the class whose private data is to be accessed.

Program- Write a program to addition of 3 number using friend function.

```
#include<iostream.h>
#include<conio.h>
class B;
class C;
class A
{
int a[5];
public:
        void in();
        friend C sum(A,B,C);
};
void A::in()
{
int k;
cout<<"\n enter five integer:";
for(k=0;k<5;k++)
```

```
cin>>a[k];
}
class B
{
int b[5];
public:
        void in();
        friend C sum(A,B,C);
};
void B::in()
{
int k;
cout<<"\n enter five integer:";
for(k=0;k<5;k++)
cin>>b[k];
}
class C
{
int c[5];
public:
        void out();
        friend C sum(A,B,C);
};
void C::out()
{
cout<<"\n addition:";
for(int k=0;k<5;k++)
cout<<" "<<c[k];
}
C sum(A a1,B b1,C c1)
{
for(int k=0;k<5;k++)
c1.c[k]=a1.a[k]+b1.b[k];
return c1;
}
void main()
{
clrscr();
A a;
B b;
C c;
a.in();
b.in();
c=sum(a,b,c);
c.out();
```

```
getch();
}
```

5.10 Friend classes

- When all the functions need to access another class in such a situation , we can declare an entire class as friend class
- Friend is not inheritable from one class to another.
- Declaring class A to be a friend of class B doesn't mean that class B is also friend of class A. therefore, friendship is not exchangeable.

```
#include<iostream.h>
#include<conio.h>
class B;
class A
{
private:
        int a;
public:
        void aset()
        {a=30;}
        void show(B);
};
class B
{
private:
        int b;
public:
        void bset()
        { b=40}
        friend void A::show(B b);
};
void Ashow(B b)
{
cout<<a<<b.b;
}
void main()
{
clrscr();
A a1;
a1.aset();
B b1;
b1.bset();
a1.show(b1);
getch();
}
```

5.11 Constant member function

- Member function of a class can also be declared as constant using ‘const keyword’.
- The const function can’t modify any data in the class.
- It is added as suffix.

5.12 Recursive member function

- When function call itself repeatedly, called recursive function .
  1) Direct recursion
  2) Indirect recursion

5.13 Member function & non-member function

program – Write a program to call a member function using non-member function.

```
#include<iostream.h>
#include<conio.h>
void moon();
class mem
{
public:
    void earth()
    {
    cout<<"on earth:";
    }
};
void main()
{
clrscr();
moon();
}
void moon()
{
mem j;
j.earth();
cout<<"on moon";
}
```

5.14 Overloading member function

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class absv
{
public:
        int num(int);
        double num(double);
};
int absv::num(int x)
{
```

```
int ans=abs(x);
return(ans);
}
double absv::num(double d)
{
double ans=fabs(d);
return(ans);
}
void main()
{
clrscr();
absv n;
int i;
double j;
cout<<"enter value of i & j";
cin>>i>>j;
cout<<n.num(i)<<n.num(j);
}
```

5.15 Overloading main() function

```
#include<iostream.h>
#include<conio.h>
class A
{
public:
        void main(int t)
        {
        cout<<t;
        }
        void main(double f)
        {
        cout<<f;
        }
        void main(char *s)
        {
        cout<<s;
        }
};
void main()
{
clrscr();
A *a;
int i;
double j;
char k[10];
```

```
cout<<"enter value of i,j,k";
cin>>i>>j>>k;
a->main(i);
a->main(j);
a->amin(k);
}
```

5.16 Indirect recursion

- When a function call itself repeatedly, known as direct recursion.
- When two function call each other repeatedly, known as indirect recursion.

program-write a program to find the factorial of number using indirect recursion.

```
#include<iostream.h>
#include<conio.h>
int n=5,f=1,j;
main(int x)
{
void pass();
if(x==0)
{
cout<<"factorial ="<<f;
}
f=f*x;
pass();
return x;
}
void pass()
{
main(m--);
}
```

CONSTRUCTOR & DESTRUCTOR

1. C++ provides a pair of inbuilt special member function called constructor & destructor.
2. The constructor constructs the object allocates memory for data members & also initializes them.
3. The destructor destroys the object when it is of no use or goes out of scope & deallocates the memory.
4. The compiler automatically executes these functions.
5. When an object is created ,compiler invokes the constructor function.
6. Destructor is executed at the end of the function when objects are of no use & goes out of scope.

Example

```
Class num
{
Private:
    int a,b,c;
public:
    num();
~num();
};
num:;num()
{
a=0;b=0;c=0;
}
num::~num()
{
cout<<"destructor invoked";
}
void main()
{
num x;
}
```

6.1 Constructor with Arguments

- The constructors are also called parameterized constructor.
- It is necessary to pass values to the constructor when an object is created

Program

```
#include<iostream.h>
#include<conio.h>
class num
{
Private:
        int a,b,c;
public:
        num(int m,int n,int k)
        void show()
        {
        cout<<a<<b<<c;
```

```
        }
};
num::num(int m,int j,int k)
{
a=m; b=j; c=k;
}
void main()
{
clrscr();
num x=num(4,5,7);
num y(1,2,8);
x.show();
y.show();
}
```

6.2 Constructor overloading

- When a class contains more then one constructor all defined with the same name as the class but with different number of arguments, is called constructor overloading
- Depending upon number of arguments the compiler executes the appropriate constructor.

Program – Write a program to find the simple interest using constructor overloading.

```
#include<iostream.h>
#include<conio.h>
class si
{
private:
        float p,i,r;
public:
        float SI  ;
        si(float a,float b,float c);
        si(float x,float y);
        si(float z);
        si();
        void out();
};
si::si(float a,float b,float c)
{
p=a;
i=b;
r=c;
}
si::si(float x,float y)
{
p=x;
i=y;
r=5;
}
```

```
si::si(float z)
{
p=z;
i=3;
r=2;
}
si::si()
{
p=20;
i=4;
r=2;
}
void si::out()
{
SI=(p*i*r)/100;
cout<<"simple interest:"<<SI<<endl;
}
void main()
{
clrscr();
si a(5.3,2.1,6.3) ;
a.out();
si b(4.5,3.2) ;
b.out();
si c(5.3) ;
c.out();
si d ;
d.out();
getch();
}
```

6.3Constructor with default Argument

Program – Write a program to find the power of a number using default arguments.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class power
{
        Private:
                int num;
                int pow,ans;
        public:
                power(int n=9,int p=3)
                void show()
                {
                Cout<<ans;
```

```
        }
};
power::power(int n,int p)
{
num=n;
pow=p;
ans=pow(n,p);
}
void main()
{
clrscr();
power(p1,p2(5));
p1.show();
p2.show();
getch();
}
```

6.5 Copy Constructor

- When we pass the reference of an object to a constructor function, declaration is known as copy constructor.
- All copy constructor required one argument with reference to an object of that class.
- Using copy constructor ,it is possible for the programmer to declare and initialize ne object using reference of another objects

Program – Write a program to show copy constructor

```
#include<iostream.h>
#include<conio.h>
class num
{
        int n;
        public:
                num(){ }
                num(int k)
                {
                n=k;
                }
                num(num &j)  //copy constructor
                {
                n=j.n;
                }
                void show()
                {
                cout<<n;
                }
};
void main()
{
```

```
clrscr();
num J(50);
num K(J);
num L=J; //copy constructor invoked
num M;
M=J;
J.show();
K.show();
L.show();
M.show();
}
```

6.6Const Object

- An object can be declared constant using const keyword
- A constructor function can initialize the data member of a constant object.
- A const object can access only constant function.

Program – Write a program to show

```
#include<iostream.h>
#include<conio.h>
class ABC
{
int a;
public:
        ABC(int m)
        {
        a=m;
        }
        void show() const
        {
        cout<<a;
        }
};
int main()
{
clrscr();
const ABC x(5);
x.show();
return 0;
}
```

6.7 Destructor

- For real 7 non-static object the destructor is executed ,when object goes out of scope.
- It is not possible to define more than one destructor .
- The destructor is only way to destroy the object. so, they can't be overloaded.
- Destructor neither required any argument nor return any value.

Program – Write a program to destroy an object

```
#include<iostream.h>
#include<conio.h>
int c=0;
class A
{
public:
    A();
    ~A();
};
A::A()
{
c++;
cout<<"\n object created:"<<c;
}
A::~A()
{
c--;
cout<<"\n object destroyed:"<<c;
}
void main()
{
clrscr();
A a1,a2,a3;
getch();
}
```

6.8 Qualifier & Nested classes

- The class declaration can be done in another class. While declaring object of such class ,it is necessary to precede the name of outer class.
- The name of outer class is called qualifier name & class defined inside class is called as nested class .

```
#include<iostream.h>
#include<conio.h>
class A
{
public:
        int x;
        A()
        {
        x=5;
        cout<<x;
        }
        class B
        {
        public:
                int y;
                B()
```

```
        {
        y=10;
        cout<<y;
        }
        class C
        {
        public:
                int z;
                C()
                {
                z=15;
                cout<<z;
                }
        };
    };
};
void main()
{
clrscr();
A a;
A:: B b;
A::B::C c;
}
```

6.9 Anonymous object

- Objects are created with names. it is possible to create object without name & such objects are known as anonymous objects.
- We can gate the address of anonymous object using ‘this pointer’.

```
#include<iostream.h>
#include<conio.h>
class noname
{
private:
        int x;
public:
        noname(int j)
        {
        x=j;
        cout<<x;
        }
        noname()
        {
        x=15;
        cout<<x;
        cout<<this;
        }
```

```
        ~noname()
        {
        cout<<"in destructor";
        }
};
void main()
{
clrscr();
noname();
noname(12);
}
```

6.10 Private Constructor and Destructor

- When a function is declared as private section ,it can be invoked by public member function of the same class, but when constructor & destructor are declared as private they can't executed implicitly & it is a must to execute them explicitly.

```
#include<iostream.h>
#include<conio.h>
class A
{
private:
        int x;
        A()
        {
        x=7;
        cout<<"in constructor";
        }
        ~A()
        {
        cout<<"in destructor";
        }
public:
        void show()
        {
        this ->A::A();
        cout<<x;
        this ->A::~A();
        }
};
void main()
{
clrscr();
A *a;
a->show();
}
```

6.11 main() as constructor & destructor

```
#include<iostream.h>
#include<conio.h>
class main()
{
public:
        main()
        {
        cout<<"in constructor main";
        }
        ~main()
        {
        cout<<"in destructor main";
        }
};
void main()
{
clrscr();
class main a;
}
```

6.12 Local versus Global Object

- The object declared outside function bodies is known as global object.
- All function can access the global object.
- The object declared inside function bodies is known as local object.
- Scope is local to its current block.