CHAPTER : 2

CLASSES AND OBJECTS **20 marks**

Objectives:

- Defining classes & objects.
- Declaring & using static data member & static member function, friend function.
- Programs based on classes & objects.

2.1 Structures in C++.
2.2 Class & Object: Introduction, specifying a class, access specifies, defining member functions, creating Objects, memory allocations for objects.
2.3 Array of Objects, Object as function arguments.
2.4 Static data members, static member function, friend Function

2.1 Structures in C++

- Structure is a user defined data type.
- It provides a method of packing together data of different types.
- Structure is a unique feature of C language that is used to handle group of logically related data items.
- Once the structure type is defined, we can create variables of that type using declarations that are similar to the built –in data types.

Syntax:

```
Struct structure-name
{
  datatype var1;
datatype var2;
.
};
```

Example:

```
Struct student
{
 int rollno;
 float marks;
};
```

- The key word struct declares student as a new data type that can hold two fields of different data type.
- rollno is of type int and marks is of type float. These fields are known as structure member.
- The identifier student is referred to as structure name or structure tag.
- One can create any number of variables of type student.

Example:
To create a variable of type student write following statement,

```
struct student s1;    // C style
student s1;           // C++ style
```

Here s1 is a variable of type student. The s1 variable will have two fields' rollno and marks as its member.

S1

rollno	marks

- Access each individual member of s1 use following expression.

  Syntax: structure-variable **.**structure-member;
  Example: s1**.**rollno =10
  S1.marks = 250
- Here dot (.) is called *'member access operator'*.

Different ways of declaring structure variables:

1) Declare a structure 'student' having data members as rollno and marks and declare 3 structure variables.

```
   struct student
  {
   int rollno;
   float marks;
  } s1,s2,s3;
```

OR

CHAPTER : 2

```
struct student
 {
  int rollno;
  float marks;
 };
 struct student s1,s2,s3;
```

2) Declare a structure 'student' having data members as rollno and marks and declare array of 10 students.

```
Struct student
{
 int rollno;
  float marks;
} s[10];
```

OR

```
Struct student
{ int rollno;
  float marks;
} ;
struct student s[10];
```

PROGRAMS BASED ON STRUCTURE.

Write a program in C++ to create a structure employee that stores name, empno and salary of the employee. Accept and display the details of employee.

```
# include <iostream.h>
struct emp
{ char name[40];
   int empno;
  float salary;
};
void main()
{
  emp E1;
  cout <<"Enter name of Employee : ";
  cin >> E1.name;
  cout << "Enter Employee Number ";
  cin >> E1.empno;
  cout<< "Enter  salary";
  cin>> E1.salary;
  cout << "DETAILS OF EMPLOYEE : " <<endl;
  cout<< "NAME  : " << E1.name <<endl
  cout<< " EMPLOYEE Number : "<< E1.empno <<endl;
  cout<< " SALARY :"<< E1.salary;
}
```

Declare a structure 'Employee' having data member's emp_id and emp_name and basic_salary. Accept this data for 5 variables and display the details of employee having salary > 5000.

```
# include <iostream.h>
struct Employee
{
  char emp_name[40];
  int emp_id;
  float basic_salary;
};
void main()
{
  Employee  e[5];
  int i;
  for ( i=0;  i<5 ;  i++)
 {
  cout << "Enter Employee name ";
  cin >> e[i].emp_name;
```

```
  cout<< ”Enter Employee ID”;
  cin>> e[i].emp_id;
  cout <, “Enter Basic Salary”;
  cin >> e[i].basic_salary;
}
cout << “DETAILS OF EMPLOYEE HAVING SALARY > 5000: “ <<endl;
for ( i=0;  i<5 ;  i++)
 {
  if ( e[i].basic_salary > 5000)
  {     cout<< ”NAME  : “ << e[i].emp_name <<endl
            cout<< ” EMPLOYEE ID : “<< e[i] .emp_id <<endl;
            cout<< ” SALARY :”<< e[i].basic_ssalary;
  }
}
```

Declare structure ‘circle’ containing data members as radius, area, and perimeter. Accept radius for one variable from user and display the value of area and perimeter.

```
# include <iostream.h>
struct circle
{
 int rad;
 float peri;
 float area;
};
main ()
{
 circle c1;
 cout << “Enter radius”
 cin >> c1.rad;
 c1.peri = 2 * 3.14 * c1.rad;
 c1.area = 3.14 * c1.rad * c1.rad;
 cout << “Perimeter = “ <<c1.peri <<endl;
 cout << “Area = “<<c1.area << endl;
 return (0);
}
```

Define a structure ‘time’ having data members as hours and minutes and write program to add two times.
Example (3 hrs 20 mins + 4 hrs 30 mins = 7 hrs 50 mins)

```
# include <iostream.h>
struct Time
{
 int hrs,mins;
};
void main()
{
 Time t1,t2,t3;
  Cout << “enter Hours and minutes of t1”;
  Cin >> t1.hrs;
  Cin >>t1.mins;
  Cout << “enter Hours and minutes of t1”;
  Cin >> t2.hrs;
  Cin >>t2.mins;
  t3.hrs = t1.hrs + t2.hrs;
  t3.mins = t1.mins + t2.mins;
  If ( t3.mins > 60)
   {
          t3.hrs = t3.hrs + (t3.mins / 60);
          t3.mins = t3.mins % 60;
   }
          cout <<” Total Hours = “ << t3.hrs;
```

```
        cout << Total Minutes = " << t3.mins;
 }
```

Declare a structure 'book' having data member's title, author and price. Accept and display data for one variable.

Declare a structure 'account' having data member's account_no and balance. Accept and display data for 5 variables.

2.2 Class & Object: Introduction

Class :

- Class is an important feature of C++. It is an extension of the idea of the structure in 'C'.
- Class is a new way of creating and implementing a user defined data type.
- Class is similar to the structure data type, but structure in 'C' is having some limitations. One of the main limitations is that structure in 'C' does not permit data hiding. Class overcomes these limitations.
- In C++ class and structure is similar in that, both can have data members as well member functions. The only difference between the structure in C++ and class in C++ is that, by default the members are private in class where as by default members are public in structure.

Specifying a class:

- The class is a way of binding data and functions associated with data together.
- The entire set of data and the functions can be made user defined data type with help of class. Thus Class is nothing but a user defined data type.
- Class act as a template, which is used to create objects. The class contains data and member functions that operate on the data.
- It allows the data to be hidden, if necessary from the external use.

Generally class specification has two parts

1) class declaration
2) class function definitions.

Syntax:

```
class class-name
{
 private :
        variable declarations;
        function declarations;
 public:
        variable declarations;
        function declarations;
};
```

Example :

```
Class student
{
private :
    int rollno;             // data member
    char name[30];          // data member
public:
    void getdata (void);    // member function
    void showdata(void);// member function
};
```

- The class declaration describes the type and scope of its members.
- The class function definitions describe how the class functions are implemented.
- The class body contains the declarations of variables and functions. These variables and functions are collectively called members. The variables are called data members or member variables and the functions are called member functions.
- They are usually grouped under two sections, private and public.
- The keyword private and public are called visibility labels.
- The private members can be accessed only from within the class and the public members can be accessed from outside the class.
- The use of keyword private of optional. By default all the members of class are private.
- Only the member functions can access the private members of class.

Access specifies

In C++ Access specifies are also called as visibility mode. They are as follows: Public, Private, and Protected .

- 1. Public:- When access specifier is public i.e. base class is publicly inherited by derived class all public member of base class become public members of derived class and all protected members become protected in derived class.
- 2. Private: - All public and protected members of base class become private members of derived class.
- 3. Protected: - In this all public and protected members of base class become protected of derived class. (1 mark).

CHAPTER : 2

Difference between structure and class:

Structure	Class
a) It contains logically related data items which can be of similar type or different type.	a) Data and functions that operate on the data are tied together in a data structure called class.
b) The data is not hidden from external use	b) It allows data and functions to be hidden from external use
c) Body of structure contains declaration of variables	c) Body of class contains declaration of variables and functions

d) Syntax- struct structure_name
{
Datatype variable_name;
Datatype variable_n;
} Structure_variable;
	d) Syntax- class class_name
	{
	Access specifier: Declare data members;
	Declare member functions;
	};

e) Structure does not contain function declaration	e) Class contains function declaration
f) Structure does not provide data hinding	f) The basic purpose of class is to hide data from external use

g) Ex. Struct student
{
int roll_no;
char name[20];
} s;
	g) Ex. Class student
	{
	Private: int roll_no;
	char name[20];
	public: void getdata();
	void putdata();
	};

DATA HIDING

- OOP emphasis on data rather than procedures. The primary focus is on Data.
- In OOP the data and the functions that operate on the data are encapsulated in to a single unit known as object.
- Thus object contains data and the member functions that operate on the data.
- Only the member functions can access the private data of the class. The private data is not accessible to the external functions. This insulation of data from the external functions is called Data Hiding or Information Hiding.
- The following diagram explains Data Hiding in C++.

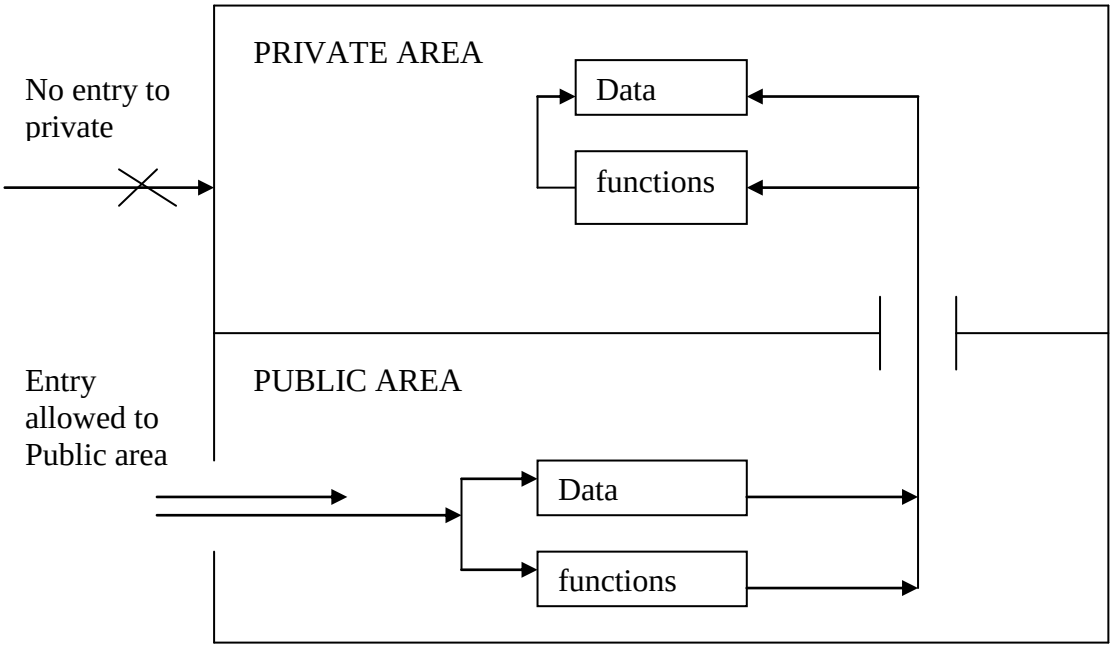

Figure: Data Hiding in Classes

CREATING VARIABLES / OBJECTS OF CLASS:

After declaration of class, one can create any number of variables (objects) of that type by using class name.

Syntax: Class-name variable-name:

Example: Student s1; // memory for s1 is created.
The above statement will create a variable 's1' of type student. These variables are known as objects. Here S1 is an object of class student.

- One may declare more than one variable as shown.

Student s1, s2, s3;

ACCESSING CLASS MEMBERS

The private data of the class can be accessed only through the member functions of that class.
To call a member function use following syntax,
Syntax: Object-name. function-name (argument-list);
E.g. If s1 is an object of student class and showdata() is a member function of the class student , then to invoke a member function we write,

s1**.** showdata();

NOTE: A variable declared as public can be accessed directly but the private variables cannot be accessed by the object directly. Private member can be accessed using member functions.

EXAMPLE:

```
Class student
{
private : int rollno;
          char name[30];
public:  int no-of-student;
};
main()
{
  student s; // create an object
  s. rollno = 10              //Error , rollno is private member , cannot be accesses directly
  s. no_of_student =100;      // OK  no_of_student is public member , can be accessed directly
  return (0);
}
```

DEFINING MEMBER FUNCTIONS:

- The functions, which are declared inside the class, are called member functions of that class.
- Only member functions can access the private members of the class.
- Generally member functions are placed in a public section. But they can be placed in a private section also.
- Member function can be defined in two places.

1) Outside the class definition
2) Inside the class definition

Defining member function out-side the class definition:

- In this, the member functions that are declared inside the class have to be defined outside the class.
- The member function incorporates a' membership identity label' in the header. This label tells the compiler which class the function belongs to.
- The general syntax of defining any member function outside the class is as follows:

```
Return- data-Type   class-name :: function-name (argument –declaration)
{
 function Body
}
```

The membership label 'class-name**::** ' tells the compiler that the function belongs to the spiffed class.

- The symbol "**::"** is called scope resolution operator.

Example :

```
Class student
{
 private :int roll-no;
          float marks;
public :void  getdata (void);       // member  function declaration
          void  showdata (void); // member function declaration
};
```

```
void student :: getdata (void)                // member function definition outside the class
{ cout<< "Enter  Roll No : " ;
  cin >> roll-no;
  cout<< "Enter  Marks  : " ;
  cin >> marks;
}
void student :: showdata (void)        // member function definition outside the class
{  cout<<" Roll No =  " <<  roll-no << endl;
  cout<<"  Marks  = " <<  marks << endl;
}
```

Defining member function inside the class definition:

In this, the function declaration is replaced by the function definition.

Example :

```
Class student
{
 private : int roll-no;
           float marks;
public : void  getdata (void)
           {  cout<< "Enter  Roll No : " ;
              cin >> roll-no;
              cout<< "Enter  Marks  : " ;
              cin >> marks;
           }
};
```

- When the function is defined inside the class, it is treated as inline function.

CHARACTERISTICS OF MEMBER FUNCTION:

1) Several different classes can use the function name. The 'membership label –(class-name **::**)' will resolve their scope.
2) Member functions can access the private data of the class. A non member function cannot do so. However friend function can access the private data. This is an exceptional case.
3) A member function can call another member function directly without using the dot (.) operator.

NESTING OF MEMBER FUNCTIONS:

- A member function of a class can call another member function of the same class. This is known as nesting of member functions.

```
# include <iostream.h>
class result
{
  private :          int rollno;
                     int marks1 ,marks2 ,sum;
                    void total ( void);     // private member function , cannot be accesses by object
public :            void setdata( int ,int ,int);          // public member function
                    void display (void);                   // public member function
};
void  result :: setdata (int r , int m1 ,int m2)
 {          rollno = r;
            marks1 = m1;
            marks2 = m2;
}
void result :: total (void)            // private member function
{           sum = marks1 + marks2;
}
void  result : : display ( void)
{
            total (); // invoking total()  inside display() – nesting of member functions
            cout << " TOTAL MEARKS = " << total;
}
```

```
void main()
{
 result r;
 r.setdata (101, 35, 45);
 r.total () ;          // Error , Object cannot access private members
 r.display();
}
```

- Nesting of member function is useful when we want to access private member function, as private member function cannot be accessed by the object.

INLINE FUNCTIONS:

- Functions are used to avoid repetitive task and also to save some memory space. However each time a function is called, it takes a lot of extra time in executing a series of instructions for task such as jumping to the function, saving registers, pushing arguments in to the stacks and returning to the calling function.
- To eliminate these problems C++ provides a new feature called inline function.
- " An inline function is a function that is expanded in line when it is invoked." That is the compiler replaces each function call with the corresponding function code.
- The inline functions are defined using the keyword inline as follows:

```
inline return- data-type  function-name ( Argument declaration)
{
  function Body
}
```

```
inline double cube ( double a)
{
  return ( a * a * a);
}
```

- The functions defined inside the class definition are treated as inline.
- One can define the function outside the class and still make it inline using inline key word.

```
Class student
{
    int rollno ;
public :   void getdata (void );
};
inline void student :: getdata (void )       // defined outside the class but still inline
{
          cout << "ENTER ROLL NO : ";
           cin >> rollno;
}
```

- Advantages of Inline functions:

In line expansion makes a program run faster.

- Disadvantage of inline functions:

Takes more memory

NOTE: The functions containing for loop or a function with a larger code are not made inline.
If you make them inline compiler gives you a warning.

ARRAYS WITHIN A CLASS:

- Arrays can be used as a member variable in a class.
- It can be declared as private or public member of class and can be used by the member functions like any other member variables.

E.g.

```
class array
{       int rollno [10];
 public :
           void setvalue (void );
           void display (void );
};
```

One can perform any operation on array. In the above example setvalue() function sets the values of array rollno[ ] and display() function displays the values.

Write a program to find out the sum of numbers stored in array.

```
# include <iostream.h>
class array
{
  private :  int a [10] ,n ;    // array as a data member
 public :void getdata( void );
           void add (void );
};
void array  : : getdata (void )
{  cout << "Enter how many numbers ?";
 cin >> n;
 cout << "Enter " << n << "numbers : " << end l;
 for ( int i =0 ; i <n ; i ++)
      cin >>a [ i ];
}
void array :: add ( void )
{   sum =0 ;
 for ( int i =0 ; i <n ; i ++)
 {
  sum = sum + a [ i ];
 }
cout << "SUM = " << sum;
}
void main ()
{ array stud;
  stud.getdata ();
  stud.add();
}
```

PROGRAMS BASED ON CLASS AND OBJECTS:

- **Write a program to declare a class 'emp' containing data member's emp_id and salary. Accept and display this data for 2 objects of the class.**

```
# include <iostream.h>
class emp
{
 private :  int emp_id;
             float salary;
public:  void getdata (void)       // Inline function
           { cout <<"ENTER EMP ID "<< endl;
             cin>>emp_id ;
             cout <<"ENTER SALARY"<<endl;
             cin >> salary;
            }
          void showdata (void)       // Inline function
          { cout <<"EMP ID =  "<< emp_id<<endl;
            cout <<"SALARY = "<< salary << endl;
         }
};  // END OF CLASS
int main()
{ emp e1,e2 ;  // creating 2 objects of class emp
e1.getdata();      // get data for object e1
e2.getdata();      // get data for object e2
e1.showdata(); // display data of object e1
e2.showdata();// display data of object e2
return (0);
}
```

CHAPTER : 2

- **Write a program to declare a class product containing data members product_code , name and price . Accept and display this information for 2 objects.**

```
#include < iostream.h>
# include < string.h>
Class product
{          int code;          // by default members are private
           char name[40 ];
           float price;
public :  void setdata ( int c , char n [ ] , float p)
          { code = c ;
            strcpy (name , n);        // include <string.h> to use strcpy()
            price = p;
          }
          void showdata (void)
          { cout << "product code = " << code;
            cout << "Product name =  " << name;
           cout << " Price = "<<price;
          }
};
void main(void)
{          product p1 , p2;
          p1.setdata (101 , "floppy" , 16);
          p2.setdata (102 . "pen" , 10);
          p1. Showdata ();
          p2.showdata();
}
```

- **Write a program to declare a class student containing data members roll_no and precentage. Accept this data for 2 objects and display the roll_no of the student having higher percentage.**

```
# include <iostream.h>
class student
{          int roll_no;                  // By default members are private
          float per;
public: void accept(void)                    // Inline function
          {          cout <<"ENTER ROLL NO "<< endl;
                     cin>> roll_no;
                     cout <<"ENTER PERCENTAGE"<<endl;
                     cin >> per;
          }
          void display (void)        // Inline function
          {          cout << " ROLL NO=  "<< roll_no << endl;
                     cout <<" PERCEANTAGE = "<< per << endl;
          }
          int getper ( void )                  // Inline function
          {
          return (per);
          }
}; // END OF CLASS
void main()
{ student s1, s2 ;           // creating 2 objects of student
s1.accept();                  // accept data for s1
s2.accept();                 // accept data for s2
if ( s1.getper() > s2.getper())
   s1.display();
else
  s2.display();
}
```

- **Write a program to define a class 'rectangle' having data member's length and breadth. Accept this data for one object and display area and perimeter.**

CHAPTER : 2

```
# include <iostream.h>    # include <conio.h>
 class rectangle
{
  private :          int l ,b;              // private data members
                     float area , peri;
   public :   void setdata(int , int);                    // function declaration
              void display_area(void);        // function declaration
};
void rectangle :: setdata ( int ll , int bb)  // Defining function outside the class
{ l = ll ;
  b = bb;
}
void rectangle :: display(void)    // Defining function outside the class
{ area = l * b;
cout << "AREA = "<< area<< endl;
peri =  2 * ( l + b);
cout << "PERIMETER = " << peri << endl;
}
void main()
{  rectangle r1 ;
  r1. setdata ( 20 ,30);     // set data for r1
  r1.display ();   // display area and perimeter
}
```

OUTPUT:

AREA = 600

PERIMETER = 100

- **Write a program to declare a class 'account' having data members principal, rate_of_interest and no_of_years. Accept this data for one object of a class and calculate Simple Interest using formulae (SI = (P*N*R) /100)**

```
# include <iostream.h>
 class SI
{
  private :  int p,n;
             float r;
  public:  void setdat       a( int pp ,int nn ,float rr)
             { p= pp;
               n= nn;
               r= rr;
              }
    void display_SI()
     {     float answer  ;              // answer  is a temporary variable  for holding result
           answer = (p*n*r) /100;
           cout<<"Simple Interest :" << answer;
     }
};
           void main()
           {
             SI s1;   // creating object of SI
             S1.setdata (100, 10, 10);          // set data for S1
             S1.display_SI ();                 // display simple Interest of S1
             }
```

MEMORY ALLOCATION FOR OBJECTS:

- Memory space for object is allocated when they are declared and not when the class is specified.
- But the member functions are created and placed in the memory space only once when they are defined as a part of the class specification.

- As all objects belongs to same class, use the same member functions, no separate memory is allocated for member functions when the objects are created i.e. when the objects are created the memory space is allocated only for data members (member variables)
- Space for data members is allocated separately for each object. The memory locations are also distinct for object of same class because the data members of different object will hold different values.

Consider the following example:

```
Class student
{     int rollno;
    float marks;
public :    void getdata(void);
            void showdata(void);
};
```

- When we declare an object at that time the memory is allocated.

Student s1 , s2, s3 ; //separate memory is allocated for s1 , s2 and s3.

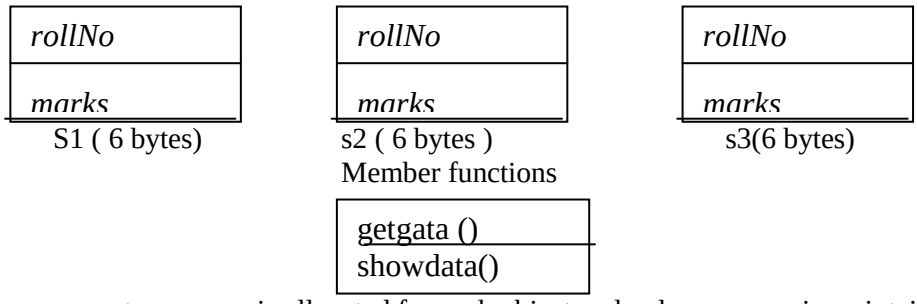

Figure: separate memory is allocated for each object and only one copy is maintained for member functions.

2.3 ARRAY OF OBJECTS

- The way we can create an array of built in data types, we can also create an array of objects of user defined data type It is possible to have array of objects of class.
- Consider the following example

```
class employee
{   int empcode;
    char name[40];
public : void accept(void) ;
         void display(void) ;
};
```

To declare an array of 4 employee objects we write.

```
Employee E[4];                    // array of  4  employee
```

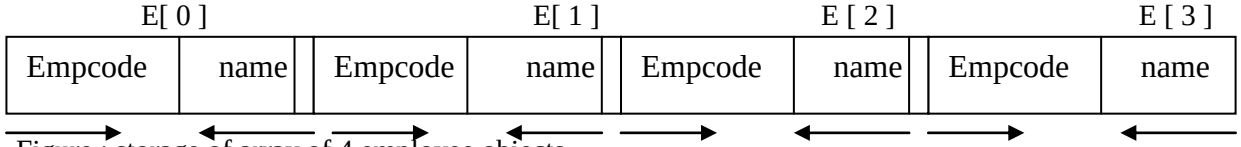

Figure : storage of array of 4 employee objects.

To access the member function of individual objects.

```
E[0].accept();       // Invoking the accept() for E[0]
E[1].accept();       // Invoking the accept() for E[1]
```

Using for loop we can access all the members as follows

```
For ( int I=0 ; I<4 ; I++)
{
  E[ I ].accept();
}
```

PROGRAMS BASED ON ARRAY OF OBJECTS

❖ **Write a program to define a class 'account' having data members as account_no , name and balance. Accept and display this information for 10 objects of this class.**

```
# include <iostream.h>    # include <coinio.h>
class account
{
 private :  int account_no;
            char name [50];
```

```
        float balance;
public : void getdata ( void );                    // function declaration
         void showdata (void);                     // function declaration
};
void account :: getdata (void)              // function definition outside the class
{ cout << "ENTER account no: " ;
 cin >> account_no;
 cout << "ENTER name : "
 cin>> name;
 cout << "ENTER balance : "
 cin >> balance;
}
void account :: showdata (void)      // function definition outside the class
{ cout << " Account no = " << account_no <<endl;
 cout << " Name = " << name <<endl;
 cout << " Balance = " << balance <<endl;
}
void main(void)
{  account  b [10 ] ;        // create array of 10 objects
  for ( int I =0 ; I < 10 ; I++ )
          b [ I ] .getdata ();
   for ( I =0 ; I < 10 ; I++ )
          b [ I ] .showdata ();
   getch () ;                  // include <conio.h> to use getch()
}
```

❖ **Modify the above program to display only names of customers having balance >10000**

Ans : Change the Showdata() function as shown below.

```
void account :: showdata (void)      // function definition out side the class
{
 if ( balance > 10000)
 {
 cout << " Account no = " << account_no <<endl;
 cout << " Name = " << name <<endl;
 cout << " Balance = " << balance <<endl;
 }
}
```

❖ **Write a program to declare a class employee having data members as name and basic salary. Accept this data for 5 employees and calculate and display Gross Sarary.**

(Gross Salary = Basic + DA + HRA where DA = 44.5 % of basic HRA = 15% of basic)

```
# include <iostream.h>
Class employee
{
 private : char name[30];
           float basic;
public: void getdata (void)
        { cout << "enter name " ;  cin >> name;
          cout << "enter basic salary ";  cin>>basic;
        }
        void showdata(void)
        {
          float GS , HRA ,DA ;
          HRA = 0.445 * basic;
          DA = 0.15 * basic;
          GS = HRA + DA + basic;
          cout << "NAME = " <<name <<endl;
          cout << "GROSS SALARY = "<< GS;
        }
};
```

CHAPTER : 2

```
void main()
{ employee e [5];
for (int I=0 ; I<5 ; I++)
  e [ I ] .getdata();
for (I=0 ; I<5 ; I++)
  e [ I ] . showdata();
}
```

❖ **Write a program to create a class worker having data members as name and skill ('plumber' , 'fitter' etc). Accept this details for 5 workers and display the names of workers having skill as plumber.**

```
# include <iostream.h>    # include < string.h>
class worker
{
 private : char name[40];
           char skill[40];
public: void accept ( void)
          {  cout << "Enter name : " ;
           cin >> name;
           cout << "Enter skill"
           cin >> skill;
          }
           void display (void )
          {  int i ;
           i = strcmp ( skill , "plumber");
           if (i == 0)
                  Cout << " NAME = " << name << endl;
          }
};        // end of class
void main ( void)
{   worker w [5];
for ( int i=0 ; i< 5 ; i++)
     w[ i  ] . accept();
cout << "NAMES OF PLUMBERS ARE : " << endl;
for ( int i=0 ; i< 5 ; i++)
   w[ i  ] . display();
}
```

❖ **Write a program to find the sum of 1 to n numbers using class.**

```
# include <iostream.h>
class sum
{         int num,;
public: void  setnumber ( int n)
          {   num = n;
           }
          void cal_sum ( void )
          { int ans = 0;
           for ( int I = 0; I < num ; I ++ )
           {
              ans = ans + I;
            }
            cout << " SUM OF NUMBERS BETWEEN 1 TO " << num  << "IS " << ans;
          }
};
void main(void)
{ sum s1
s1.setnumber(6);
 s1.cal_sum () ;
}
```

OUTPUT :

SUM OF NUMBERS BETWEEN 1 TO 6 IS 21

CHAPTER : 2

❖ **Write a program to define a class cube with following members – length, width, depth and volume () . Write code for the function volume which calculates volume of a cube.**
(Volume of a cube = length * width * depth) (Summer –2003)

```
# include < iostream.h>
class cube
{     int l ,b,h;
public:void  accept(void)
          {  cout << " Enter Length , width , height ";
           cin>>l >> b >> h;
          }
           void volume(void)
          {  int vol;
            vol = l * b * h;
            cout << "VOLUME OF CUBE =" << vol;
          }
};
void main(void)
{ cube c1;
c1.accept();
c1.volume();
}
```

PASSING OBJECT AS ARGUMENT TO MEMBER FUNCTIONS:

Object can be used as function arguments. The object can be passed to the function by two ways:

1) Pass by values: A copy of entire object is passed to the function .
2) Pass – by –reference : Only the address of the object is passed to the function

- As in first case, a copy of entire object is passed to the function. The function creates another copy of that object and therefore any changes made to the object inside the function do not affect the actual object. This method is also called Pass-by-value
- The second method is called Pass-by-reference. In this method, the address of the object is passed to the function. Any changes made to the object inside the functions are directly reflected to the actual object.

Advantage of Pass-By- Reference

- The pass-by-reference method is more efficient than pass-by value since it requires passing only the address of the object and not the entire copy. Thus it saves memory space.
- Any changes made in the functions are reflected to the actual object.

PROGRAMS BASED ON PASSING OBJECT AS FUNCTION ARGUMENT.

- Write a program to add two complex numbers.

The same program can be done by following three methods.

1) **Adding complex numbers by passing two objects as arguments.**

```
Class complex
{    int real,;
   float img;
public :
          void setdata ( int , float);
          void add ( complex , complex);
          void showdata(void);
};
void complex ::setdata ( int r, float I)
  {
          real = r;
          img =I;
  }
void  complex :: add ( complex c1 , complex c2) //takes two objects as argument c1 , c2
{
 real = c1.real + c2.real ;
 img = c1.img + c2.img;
}
```

```
void complex :: showdata (void)
{
 cout << real << "+" << "i" << img;
}
void main(void)
{
  complex c1,c2,c3;
  c1.setdata(6,-7);
  c2.setdata ( 3, 4);
  c3.add(c1,c2);   // c1 and c2 will be explicitly passed and result will be in c3.
  c3.showdata();
}
```

2**) Adding complex numbers by passing only one object as argument.**

The same program can be done by passing only one object as an argument.
Make the following changes in the add function.

```
void  complex :: add ( complex c2)        // take only one object c2  as argument
{
 real = real + c2.real ;
 img = img + c2.img;
}
```

Note that here we are passing only one object c2 as argument to the add function.

```
void main(void)
{
 complex c1,c2;
 c1.setdata (3,4);
 c2.setdata ( 5,7);
 c1.add (c2);                  // c2 will be explicitly passed ,result will be in c1.
 c1.showdata();
}
```

3) Returning entire object from the function

```
class complex
{    int real,;
     float img;
public : void setdata ( int , float);
            complex add (complex);
            void showdata(void);
};
void complex :: setdata ( int r, float I)
   {
            real = r;
            img =I;
   }
complex  complex :: add ( complex c2)
{
 complex temp; // create a temporary object
 temp. real = real + c2.real ;
temp. img = img + c2.img;         // result is in temp object
 return temp;                  // return temp object to the main prog.
}
void complex :: showdata (void)
{
 cout << real << "+" << "i" << img;
}
void main(void)
{
  complex c1,c2,c3;
  c1.setdata(5,-6)
  c2.setdata ( 3,4);
```

```
  c3 = c1.add (c2);   // c2 is explicitly passed, result will be in c3.
  c3.showdata();
}
```

❖ **Create a class time having data members as hour and minutes and W A P to add to Time objects.**

```
# include <iostream.h>
Class time
{
              int hr , min;
public :void setdata (int , int);
            void add (time ,time );
            void showdata (void);
};
void time :: setdata (int hh , int mm)
{
 hr = hh;
 min =mm;
}
void time ::  add (time t1 , time t2)
{
  hr = t1.hr + t2.hr;
  min = t1.min + t2,min;
  if ( min >= 60 )
 {
   hr = hr + (min / 60);
   min = min % 60;
  }
}
void time :: showdata(void)
{
cout << hr << ":" << min ;
}
void main()
{
 time t1,t2,t3;
t1.setdata (3,50);
t2.setdata (4 ,30);
t3.add (t1, t2);    // result is in t3
t3.showdata();
}
```

OUTPUT:

8 : 20

❖ **Declare a class distance having data members as feet and inches. Write a program to add two distances.**

```
# include <iostream.h>
Class distance
{
              int feet , inches;
public : void setdata (int , int );
            void add (distance , distance );
            void showdata (void);
};
void distance :: setdata (int f , int i)
{
 feet = f;
 inches = i;
}
void distance ::  add (distance d1 , distance d2)
{
  feet = d1.feet + d2.feet;
```

```
  inches = d1.inches + d2.inches;
  if ( inches >= 12 )
 {
   feet = feet + (inches / 12);
   inches = inches % 12;
  }
}
void distance :: showdata(void)
{
cout << feet << "Feet and  " << inches << " Inches " ;
}
void main()
{ distance d1,d2,d3;
d1.setdata (3, 6);
d2.setdata (4 ,9);
d3.add (d1, d2); // result is in d3
d3.showdata();
}
```

OUTPUT:
8 Feet and 3 Inches

2.4 STATIC DATA MEMBERS:

Characteristics of static data members

1. It is initialized to zero when the first object of the class is created. No other initialization is permitted.
2. Only one copy of that member is created for entire class and is shared by all the objects of that class. It doesn't matter with the number of objects created.
3. It is visible only within the class, but its lifetime is the entire program.
4. The static data members are associated with the class rather than with any object ,that is why they are sometimes called **class variables**.
   To access static data always use name of the class instead of object with scope resolution operator (**::**).
   Eg: item **::** count;
   Where item name of the class and the count is is the static data member.
5. The type and the scope of each static variable must be declared outside the class definition

int item :: count

This type of variable is normally used to maintain values, which is common to entire class.

PROGARM ILLUSTRATING THE USE OF STATIC DATA MEMBER

```
# include < iostream.h>
class student
{   static int count;
    int rollno;
    float marks;
public:void setdata(int r , float m)
          {
            rollno = r;
            marks = m;
            count ++;
          }
          void displaycount(void)
          {
          cout << "COUNT = "<<count << endl;
          }
};
int student :: count ;       // count IS REDIFINED OUT SIDE THE CLASS
void main()
{    student s1, s2;         // count is initialize to zero
  s1.displaycount();
  s2.displaycount();
  s1.setdata (101,345);
```

```
  s1.displaycount();
  s2.setdata(102,450);
  s2.displaycount();
}
```

```
OUTPUT:
COUNT = 0
COUNT = 0
COUNT = 1
COUNT = 2
```

NOTE : static data members can be called using class name instead of object as shown below

Student **::** count

STATIC MEMBER FUNCTIONS

Like member variables, member functions can also be declared as static.

Characteristics of static member functions:

1. A static function can have access to other static members declared in the same class.
2. A static member function can be called using the class name and **::**

   Example : If display() is static function of class student then display() can be called as

   Student :: display();

PROGARM ILLUSTRATING THE USE OF STATIC MEMBER FUNCTIONS

```
# include < iostream.h>
class student
{
 static int count;
 int rollno;
 float marks;
public: void setdata(int r , float m)
        {
          rollno = r;
          marks = m;
          count ++;
        }
        static void display_count(void)
        {
        cout << "COUNT = "<<count << endl; }
};
int student :: count ;      // count IS REDIFINED OUT SIDE THE CLASS
void main()
{
  student s1, s2;// count is initialize to zero
  student :: display_count();
  s1.setdata (101,345);
  student :: displaycount();
  s2.setdata(102,450);
  student :: displaycount();
}
```

```
OUTPUT:
COUNT = 0
COUNT = 1
COUNT = 2
```

FRIEND FUNCTION

- Private members of a class cannot be accessed from outside the class. However there could be a situation where more than one classes want to share a particular function.

- For example consider two classes 'manager' and 'scientist'. We can use a function incom_tax ( ) that will operate on the objects of both these classes and calculate the total income tax. Here the function income_tax () need to access private members of both the classes.
- This can be done with the help of friend function.
- "Friend function is a normal c++ function that can access private members of the class to whom it is declared as friend."
- To make a normal c++ function "friendly" to the class , precede the declaration of the function with keyword 'friend' inside the class.

Example

```
Class fun
{
private:
        int a, b;
public:
    friend void xyz ( fun s);  // function xyz is a friend of class fun
};
```

- The function is defined somewhere in the program (outside the class) like a normal c++ function.
- The function can be declared as to any number of classes.
- The friend function is not a member of a class, but it has full access rights to the private members of the class.

CHARACTERISTICS OF FRIEND FUNCTION:

- It is not in the scope of the class to which it is declared as friend.
- Since it is not in the scope of the class it cannot be called using an object of the class.
- It can be invoked like a normal c++ function without the help of any object.
- Unlike member function, it cannot access the data members directly and has to use an object name and dot (.) operator.
- It can be declared either in the public or private sections of a class without affection its meaning.
- Usually it has object as argument.

Differentiate member function and friend function

Member function	Friend function
It is in the scope of the class.	It is not in the scope of the class to which it has been declared as friend
Member function is called using object and dot operator e.g. If student is class then getdata() member function can be invoke as, student s1; s1. getdata();	It is invokes like a normal c++ function without using object. E.g. if fun () is a friend function os class ABCthen it can be called as ABC a1; Fun (a1);

Member functions can access private members directly.
e.g.
class sample
{
int a;
public :
void show()
{
cout << a; // access directly
}
	Friend function requires the object of the class to access its private members.
	e.g.
	class sample
	{ int a;
	public :
	friend show (sample);
	}
	void show (sample s)
	{
	cout << s.a; // access using object s }

Member function cannot share more than one class.	Friend function can be declared a friend of more than one class. I.e. It can access private data of more than one class.

NOTE: member function of one class can be a friend of another class.

- Member function of one class can be friend functions of another class. In such cases they are defined using the scope resolution operator.

```
class x
{ ------
int fun(); // member function of x
};
class y
{---------
----------
friend int x : : fun()
}; ---------
```

- We can declare all the member functions of one class as the friend functions of another class by declaring that class as a friend class.

```
Ex . class z
{
--------
--------
friend class x ;
};
```

PROGRAM BASED ON FRIEND FUNCTION

/*Program to find the mean value of a given number using friend function.

```
#include<iostream.h> #include<conio.h>
class  sample
{
   int val1,val2;
  public:
   void get()
   {
     cout<<"Enter two values:";
     cin>>val1>>val2;
   }
   friend float mean(Sample ob);
};
float mean(sample ob)
{
  return float(ob.val1+ob.val2)/2;
}
void main()
{
   clrscr();
   sample  obj;
   obj.get();
   cout<<"\n Mean value is : "<<mean(obj);
   getch();
}
```

Output:

Enter two values: 10, 20

Mean Value is: 15

- **Write a program where test1 and test2 are two classes. Write a friend function which will calculate average of test1 and test2. (pass by value)**

```
# include <iostream.h>
class test2;        // prior declaration of class2
class test1
{
 private: float m1,m2;
public:  void setdata (float a , float b)
        {
```

```
        m1=a ;
        m2 =b;
      }
      friend void average ( test1 ,test2);  // function average is a friend of test1
};
class test2
{
 private: float m3,m4;
public:  void setdata (float a , float b)
       {
        m3=a ;
        m4 =b;
       }
      friend void average ( test1 ,test2);      // function average is a friend of test1
};
void average ( test1 t1 , test2 t2)
{
  float avg;
  avg =( t1.m1 + t1.m2 + t2.m3 + t2.m4)/4;
  cout << avg;
}
void main ()
{
 test t1;
 test t2;
 t1.setdata (34, 70);
 t2.setdata (40, 60);
 average ( t1 ,t2 );  // friend fun. is invoked like a  normal  c++ function without any object
}
```

SR.NO	Structure	Class
1	It contains logically related data items which can be of similar type or different type.	Data and functions that operate on the data are tied together in a data structure called class.
	The data is not hidden from external use	It allows data and functions to be hidden from external use
	Body of structure contains declaration of variables	Body of class contains declaration of variables and functions

	Syntax- struct structure_name
	{
	Datatype variable_name;
	Datatype variable_n;
	} Structure_variable;
		Syntax- class class_name
		{
		Access specifier:
		Declare data members;
		Declare member functions;
		};

	Structure does not contain function declaration	Class contains function declaration
	Structure does not provide data hinding	The basic purpose of class is to hide data from external use

	Ex. Struct student
	{
	int roll_no;
	char name[20];
	} s;
		Ex. Class student
		{
		Private: int roll_no;
		char name[20];
		public: void getdata();
		void putdata();
		};