CLASSES AND OBJECTS

Access specifiers

- Define scopes of members of a class or structure (in C++) .They are
- **PRIVATE**

(can be accessed by only functions of that structure or class)

- **PUBLIC**

(can also be accessed by all functions outside the structure or class)

- **PROTECTED** (for class only)

(can be accessed by derived class or friend class of container class)

About C structures

- They provide a method for packing together data of different types.Ex

```
struct student
{
 char name[10];
 int roll_no ;
 float total_marks ;
}
```

- The variables can be created like:

```
struct  student  A ;
```

- Member variables can be accessed using ***dot operator*** like:

```
A.roll_no=57;
```

- Structure can have arrays,pointers or structures as members.

Limitations of C Struct

- Cannot be treated as built-in types. Ex:

```
struct complex
{
   float x;
   float y;
};
struct complex c1,c2,c3;
```

- It cannot be used as: `c3= c1+c2;`
- C struct does not permit data hiding. The struct members are ***public*** by default.

About C++ structures

- By default all members of structure are public.
- Some members can be declared as private.
- Structure variable can be declared w/o using STRUCT key word.

  e.g complex c; ***// valid in C++ but gives an error in C***
- Structure may contain functions also.

Structure with function

```
struct data
{
   int val=20; // By default public
   Void display_init()
   {
      cout<<"initial value is"
   <<initial;
            get_initial();
   }
   private:
   void get_initial(){
   cin>>initial;}
   int initial=10;
}
```

```
int main()
{
   data d;
   d.inital=100; //will give an error
   d.get_initial(); // will give an error
   d.display_init();
   d.val=200;
   return 0;
}
```

- Thus private members can be accessed only by other member functions.

Definition of a class

- A class is a data type that binds the ***data*** and ***functions*** to access that data together.
- A class is specified by :

1. **Class declaration** (***WHAT*** are the members,their types and scopes).
2. **Class function definitions** (***HOW*** member functions are implemented).

- The members declared as PRIVATE can only be accessed by member functions of the class (**DATA HIDING)**whereas PUBLIC members can be accessed from anywhere even outside the class.
- Binding data & functions(to access that data)into a class is called **encapsulation.**

- **Eg.**

```
Class inventory
{
    int number;    // By default "PRIVATE"
    float price;
    public:
    void getdata (int a,float b);    //NO DEFINITION
                    //can be called from outside the class
    void display(void);// and can access private member
}
```

- Thus a private member can not be accessed by any non-member function of the class.
- Class variables(called ***OBJECTS***) are to be created to use its properties.

- An object of a class can be created by :

class_name object_name;

Eg: inventory i1,i2;

Then to assign values i1.getdata(19,20.50);

And to display data i1.display(); is to be called.

- Member Functions can be defined *either* **outside** *Or* **inside** the class definition.Outside the class definition the use of scope resolution operator is required
- syntax for the function is

Return-type class_name : : funct_name(arg Declaration)

{ statements}

:: is called scope resolution operator(ie scope of function is limited to this class only)

Scope Resolution Operator(::)

A variable defined with same name as global and local ,then function having its local definition can access global value by this operator.

- This problem is resolved in C++ using scope resolution Operator "::"
- That is in nested block one can write
- ::var_name to access the variable in nesting(outer) block.

e.g of using ::

```
#include<iostream.h>
int i=1234;  //global definition of i
void display()
{
    int i=90;//local definition of  i
    for(int j=1;j<4;j++)
    {
     for(int k=9;k<13;k++)
    cout<<"k is"<<k<<endl;
    cout<<"local and global values are" <<i
    <<"\t"<<::i;
}
}
int main()
 {
//int i=1234;//not global , not accessible
    anyhow  to display()
display();
return 0;
}
```

Output will be

j is 1

j is 2

j is 3

local and global i are 90 1234

- **Member function defined inside the class**

```
class inventory
{
  int number;
  float price;
  public:
    void getdata (int a,float b) //By default INLINE
    {  Number=a;price=b; }
    void putdata (void) // By default INLINE
    {cout<<number<<price;}
};
```

Member function definition outside the class(using ::)

```
#include<iostream.h>
class inventory
{
int no; float price;
public:
void display(void);
void getdata(int ,float);
};
void inventory : : display (void)
{
cout<<"no and price are "<<no<<"\t"<<price;
}
```

```
void inventory :: getdata (int a, float b)
{
no=a; price=b;
}

int main()
{
inventory i;
i.getdata(3,45.67);
i. display();
return 0;
}
```

Some CHARACTERISTICS of member functions..

1. Different classes can use **same** function name.
2. A member function can call other member function **W/O using dot operator**.
3. Some member functions can be defined inside the class and some outside the class (using :: operator.)
4. Member functions can have direct access to private data items
5. Member function defined inside the class are is treated as **inline** function.(*In object oriented programming ,it is a good practice to define member functions outside the class definition.)*

- To **avail** the benefits of inline function,the keyword "**INLINE**" can be associated with a member function outside the class

e.g

```
Class inventory
{
…….;
Public:
 void display(void);
}
 inline void inventory:: display(void)
{cout<<number << price;}
```

Nesting of member functions

- A member function can call some other member function of the same class.e.g

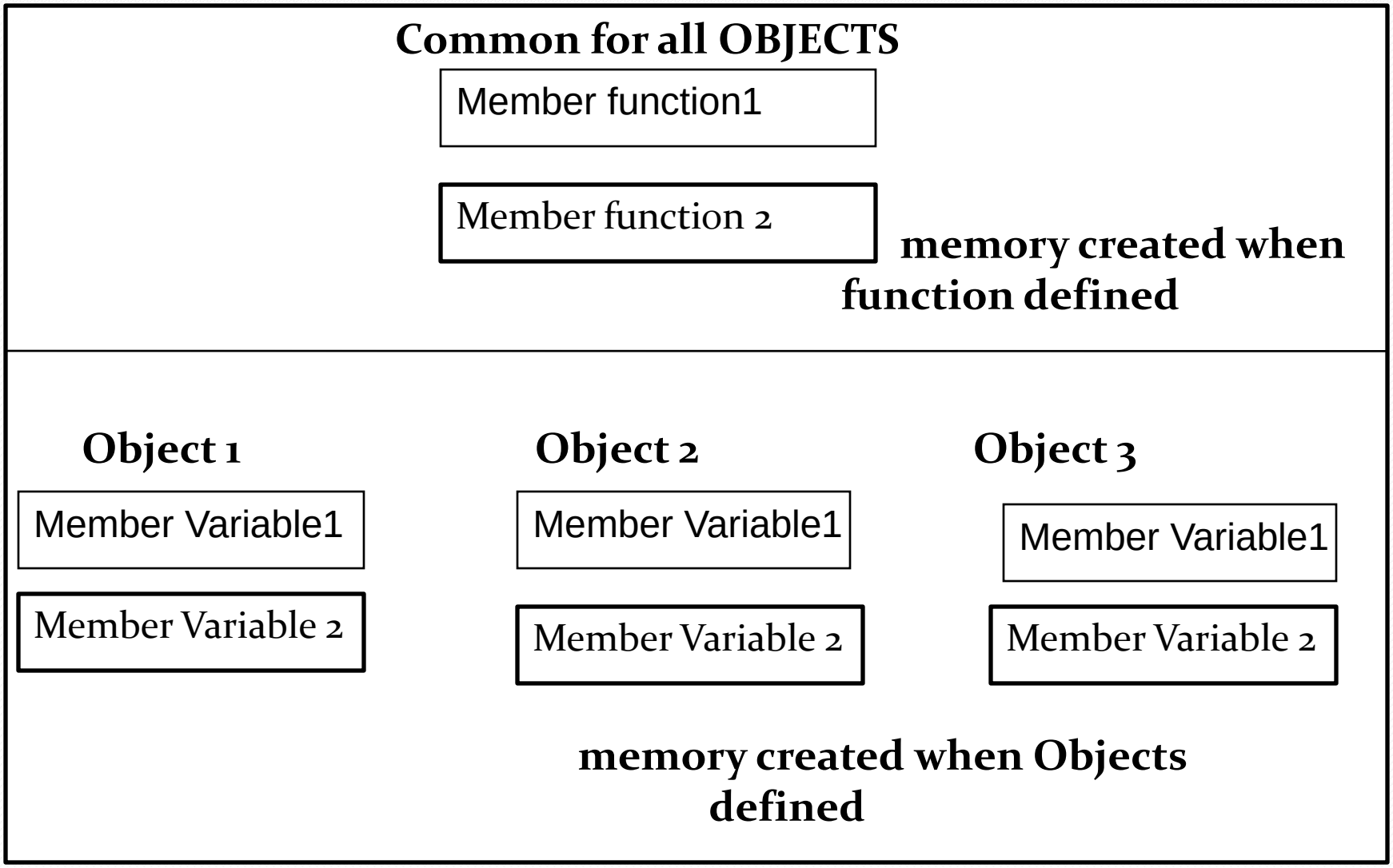
```
class cls
{
int number;                    //Data declaration;
public:
  void Member_func1(void);
   int Member_func2(void);
}
Void cls:: Member_func1(void)
{…….;
  number=Member_func2();
}
```

Private Member Function

- Private members cannot be invoked by any object using dot operator. Example…..

```
class Example
{  int I  ;
    void display(void);
    public:
    void displ (void)
    {display();}
 };
  int main()
  {  Example t1;
     t1.I=12;              //is wrong
     t1.display();  //  is wrong ,private member
     t1.displ();     //will in turn call display() ,thus no direct
    access to private members.
   }
```

Memory Allocation for Objects



Memory Allocation for Objects

- Memory space for objects is allocated when they are declared not when the class is specified.
- In a class specification **member functions** are allocated memory space (ie once) when they are defined.
- The **objects** created take the physical memory space equal to data part in class (member functions are stored separately and accessed by every object of that class).
- Separate memory locations for the **member variables** is allocated because member variables will hold different values for different objects.

Static Data Members

Characteristics of Static member variables are:

- It is initialized to **zero** when the first object of that class is created.
- Only **one copy** of that member is created for the entire class and is shared by all the objects of that class.
- It is **visible** only within the class but its lifetime is the entire program.
- Static Variables are normally used to maintain values common to the entire class.
- Static Variables are like non inline member functions. We can initialize the value

```
int item :: count =10;
```

```
class item
{   static int count;
    int number;
    public: void getdata(int a)
    { number =a;
      count++ ;
     }
     void getcount(void)
     { cout<<"count :";
       count<<count << "\n";
     }
};
int item::count ;
// definition outside class declaration
 int  main()
{ item a,b,c;
a.getcount();
b.getcount();
c.getcount();
a.getdata();
 b.getdata();
 c.getdata();

 cout<<" After reading
    data"<<"\n" ;
a.getcount();
b.getcount();
c.getcount();
 return 0;
}
```

Static Member Functions

A Static member Function's Characteristics are:

- It can have access to only other static members (functions or variables) declared in the same class.
- A Static member function can be called using the class name(instead of its objects) as:

```
class-name  : :  function-name ;
```

-
```
class abc
{  int a ;
   public: static void show(void)
                { cout<< a;}  // a is not static;
};
```

```
class test
{   static int count;
    int code;
    public:
    void setcode(void)
    { code=++count ;
     }
     void showcode(void)
     { cout<<"object no :"<<code<< "\n";
     }
    static void showcount(void)
    {
     cout<<"count:"<<count<<"\n";
    }
};
int test : : count ;
// definition outside class declaration
```

```
int  main()
{    test t1,t2;
 t1.setcode();
 t2.setcode();
 test : : showcount();
 test t3 ;
 t3.setcode();
 test : : showcount();
 t1.showcode();
 t2.showcode();
 t3.showcode();
 return 0;
}
```

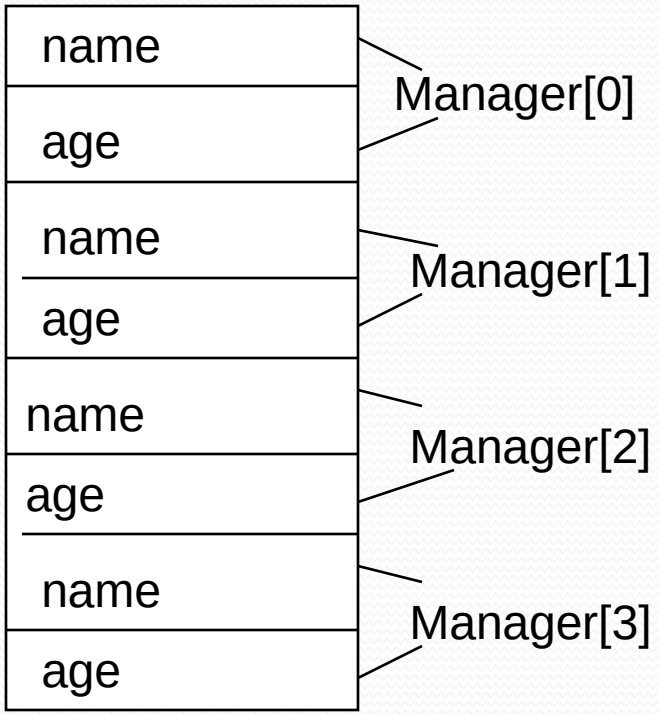
Arrays Of Objects

- Arrays of objects can also be created

```
Class employee
{
   char name[30];
   float age;
 public: void getdata(void);
         void putdata(void);
};
e.g employee manager[4];
    employee worker[35];
```

- To access member functions

```
manager[i].putdata();
```



Storage of data items of an object array

Passing objects into functions

- Can be done in two ways :

1. Copy of an object is being passed

   (PASS BY VALUE)

2. Address of the object is passed

   (PASS BY REFERENCE)

In pass by value changes made to the passed objects are not reflected in calling program whereas in pass by reference it happens.

Passing objects by value

```
class marks
{ int sub1_mar;  int sub2_mar;
Public :
void getmarks (int s1 ,int s2)
{sub1_mar=s1; sub2_mar=s2;}
void sum(marks m1,marks m2);
 };
void marks:: sum (marks m, marks n)
{  sub1_mar = m.sub1_mar + n.sub1_mar;
   sub2_mar = m.sub2_mar + n.sub2_mar;
}
```

```
int main()
{
marks m1,m2,m3;
m1.getmarks(23,34);
m2.getmarks(45,12);
m3.sum(m1,m2);
 return(0);
}
```

Thus sub1_mar of m3 is set as sum of sub1_mar of m1 and m2. **AND** sub2_mar of m3 is set as sum of sub2_mar of m1 and m2.

Passing objects by references

```
class marks
{ int sub1_mar; int sub2_mar;
public :
void getmarks (int s1 ,int s2)
{sub1_mar=s1; sub2_mar=s2;}
void sum (marks &,marks &);
 };
void marks:: sum(marks &m, marks &n)
{  sub1_mar = m.sub1_mar + n.sub1_mar;
   sub2_mar = m.sub2_mar + n.sub2_mar;
}
```

```
int main()
{
marks m1,m2,m3;
m1.getmarks(23,34);
m2.getmarks(45,12);
m3.sum(m1,m2);
 return(0);
}
```

Thus sub1_mar of m3 is set as sum of sub1_mar of m1 and m2. **AND** sub2_mar of m3 is set as sum of sub2_mar of m1 and m2.

Access a private member using a non member function-------FRIEND FUNCTION

- A friend function is a non member function that can access private members of one ,two or more classes.(it must be declared in all)
- that is two(or more) classes can be made friends using friend function.
- declared with keyword " **friend**" in any of the three sections of the classes ***(function can access private data of these classes)***

```
Class abc
{ ……
Public: …..
……..
 friend void xyz(void);        //declaration
}
```

More about friend function

- can be defined elsewhere in program
- definition doesn't have **friend** keyword or :: operator (thus cant be called using object of any class)

OR…it can be called as a normal function without the object of the class *(not in scope of that class)*

- ***Takes objects of classes as arguments.***

(Thus access members of class(es) using dot operator)

Charateristics of Friend function

- It is not in the scope of the class to which it has been declared as **friend.**
- It is not called using the object of that class.
- It can be invoked like normal function without using object.
- Unlike member function it cannot access the member names directly & has to use an object name & dot operator (A.x)
- It can be declared either in public, private part of class without effecting its meaning.
- Usually, it has the objects as arguments.

Example of a friend function

```
#include <iostream.h>
class first
{  private:
//friend int largest(first F); can be declared here also
 int a,b,c;
 public:
 void get_nums(void)
 { cout<<“enter a b and c”;
  cin>>a>>b>>c;  }
 friend int largest ( first F );
  protected:
//friend int largest(first F); can be declared here also
   };
```

```
int largest (first obj)  //defined W/O any
                                  class name and ::
{
 int max;  // local definition
max=obj.a;      // accessing private member a
if(obj.b>max)
max=obj.b;     // accessing private member b
if(obj.c>max)
max=obj.c;     // accessing private member c
return(max);
}
```

```
 int main()
{
int rslt;
first obj1;
obj1.get_nums(); // called with object obj1
rslt=largest(obj1);  // called W/O any object
cout<<“largest no is”<<rslt;
}
```

Friend function of two classes

- Member functions of one class can be friend functions of another class.
- They are defined using scope resolution operator.

```
Class X                    Class Y
{ .......                  { .......
  .......                    .......
  int fun1();                friend int X : : fun1();
  .......                    .......
};                                 };
```

Example :

```
# include<iostream.h>
# include<conio.h>
class second;   // forward declaration of class
                   SECOND
class first
{
int a;
public:
void get_num()
{   cout<<"enter one integer number";
    cin>>a; }
friend void display(first,second);
};
```

```
class second
{
float b;
friend void display(first,second);  //defined under
                                                PRIVATE
public:
void get_num()
{
cout<<"enter one float number";
cin>>b;
}
// friend void display (first , second);
};
```

```
void display (first f,second s)
{  cout<<"integer of class first is"<< f.a;
   cout<<"float of class second is"<<s.b;}
int main()
{
clrscr();
first obj_f;
second obj_s;
obj_f.get_num();
obj_s.get_num();
display(obj_f,obj_s);  // no need for any object to call it
return 0;
}
```

Another Example

```
# include<iostream.h>
# include<conio.h>
 class B;
 class A
 { int a;
   public : void aset() { a=30;}
                  void show(B);
 };
 class B
{ int b;
  public: void bset() {b=40;}
   friend void A : : show (B bb);
};
```

```
 void A: : show(B bb)
{  cout <<“\n a=”<<a;
   cout<<“\n b=“<<bb.b;
}
 int main()
{  A a1;
   a1.aset();
   B b1;
   b1.bset();
   a1.show(b1);
    return 0;
}
```

What is a friend class?

- A friend class (say F) of a class (say C) can access all private members of C.
- Irrespective of the access specifier, any member function of F can access any member function of C.

```
class C
{.....
friend class F ; // All members of F are friend to C.
};
```

Example of a Friend class

```
#include<iostream.h>
class two;// forward declaration of TWO class
class one
{
int a,b;
void get_val(void)
{  cout<<"enter two integers“<<endl;
   cin>>a>>b;
}
friend class two;
};
```

```
class two
{
  public:
  void display (one obj1)
  {   obj1.get_val();
    cout<<"private members of class one are”
   <<obj1.a<<"and"<<obj1.b;  }
};
int main()
{
two obj2;
one obj1;
obj2.display(obj1);
return 0;
}
```

Local Classes

- Classes can be defined & used inside a function or a block. Such classes are called local classes.
- Example:

```
 void test(int a)              //function
{   ……..
    ……..
    class student             //local class
    {……
    …….                       //class defination
    };
    ……..
    student  s1 ;
}
```

- Local classes can use global variables declared above the functions and static variables declared inside the function.
- The global variables should be used with scope resolution operator(: :).

Example

```
# include<iostream.h>
# include<conio.h>
 class A
{  int a;
   public: void get()
                { cout<<"\n Enter value for a:";
                   cin>> a;
                  }
                  void show()
                  { cout<<endl<<"a="<<a;
                  }
};
 int main()
{ class B
   {  int b;
```

```
        public : void get()
                        { cout<<"\n Enter the value of b";
                          cin>> b;
                         }
                        void show()
                         { cout<<"b="<<b;
                          }
     };
    A a_obj;
    B b_obj;
    a_obj .get();
    b_obj .get();
   a_obj.show();
    b_obj.show();
    return 0;
}
```

Returning Objects

- A function cannot only receive objects as arguments but also can return them.

```
# include<iostream.h>
# include<conio.h>
 class complex
   { float x,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;
 }
```

```
void complex : : show (complex c)
{ cout<<c.x<<“+j”<<c.y<“\n”;
}
int main()
{ complex A, B, C;
A.input(3.1 ,6.65);
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;
}
```

The const Argument

- The constant variable can be declared using const keyword.The const variable should be initialized while decalring. Syntax:

const <variable name>=<value>;

<function name>(const <type>*<variable name>);

Eg: int const x; // invalid

int const x=5; //valid

int cube (const int *x);

int cube (const int &s);

The const Member Function

- The constant function can be declared using const keyword.The constant function cannot modify any data in class. Syntax:

<Return type><function name>(<arguments>) **const**;

Eg: void mul (int ,int) **const** ;

```
class A
{ int c;
   public : void add( int a, int b ) const
                { // c=a+b; // invalid
                     int tot=a+b;
                     cout<<"a+b="<<tot;
                 }
};  int main()
    { A a;
      a.add(5,7);
      return 0;}
```

Pointer to Members

- Pointers can be declared to hold the address of a member of a class.
- A class member pointer can be declared using the operator **::*** with the class name

Example:

```
 class A
{ int m;
   public: void show();
};
```

Pointer declared as:

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

- The pointer ip can access member m as:

```
A a;
cout << a.*ip;          // display m
cout << a.m;            // display m
```

- Now pointer to object can also be declared:

```
ap= &a;                 // ap is pointer to object a
cout<< ap -> *ip;   //display m
cout <<ap->m;     //display m
```

- **The dereferencing operator** (->*) is used to access a member when we use pointers to both the object and the member. And (.*) used when the object itself is used with the member pointer.

object name .* pointer to member function ;

pointer to object ->* pointer to member function ;

Assignment II

A) Write a c++ program to overload average function to compute average of three and four numbers respectively.

B) Write a C++ program to overload a function for computing area of a rectangle and a circle .

C) Write any C++ program to swap the values using friend function.

D) Write any C++ program to calculate simple interest using friend function which returns float value.