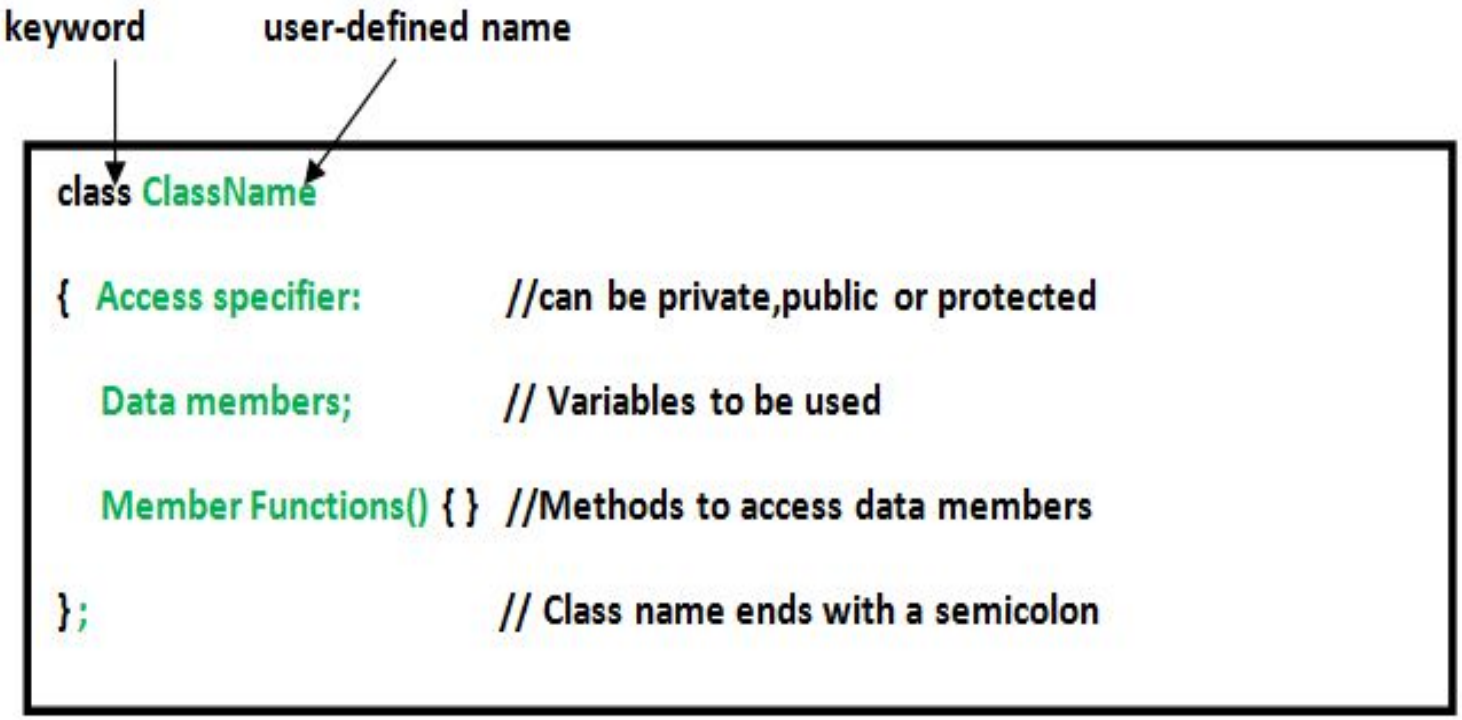
Classes and Objects C++

CLASS & OBJECT

- **Class:** The building block of C++ that leads to Object Oriented programming is a **Class**.
- **It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.**
- For Example: Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc.
- So here, Car is the class and wheels, speed limits, mileage are their properties.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
- In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc and member functions can be *apply brakes*, *increase speed* etc.

Defining Class and Declaring Objects



- **Declaring Objects:** During class defintion, only the specification for the object is defined; no memory is allocated. To use the data and access functions defined in the class, we need to create objects.
- **Syntax: ClassName ObjectName;**

Example of Class

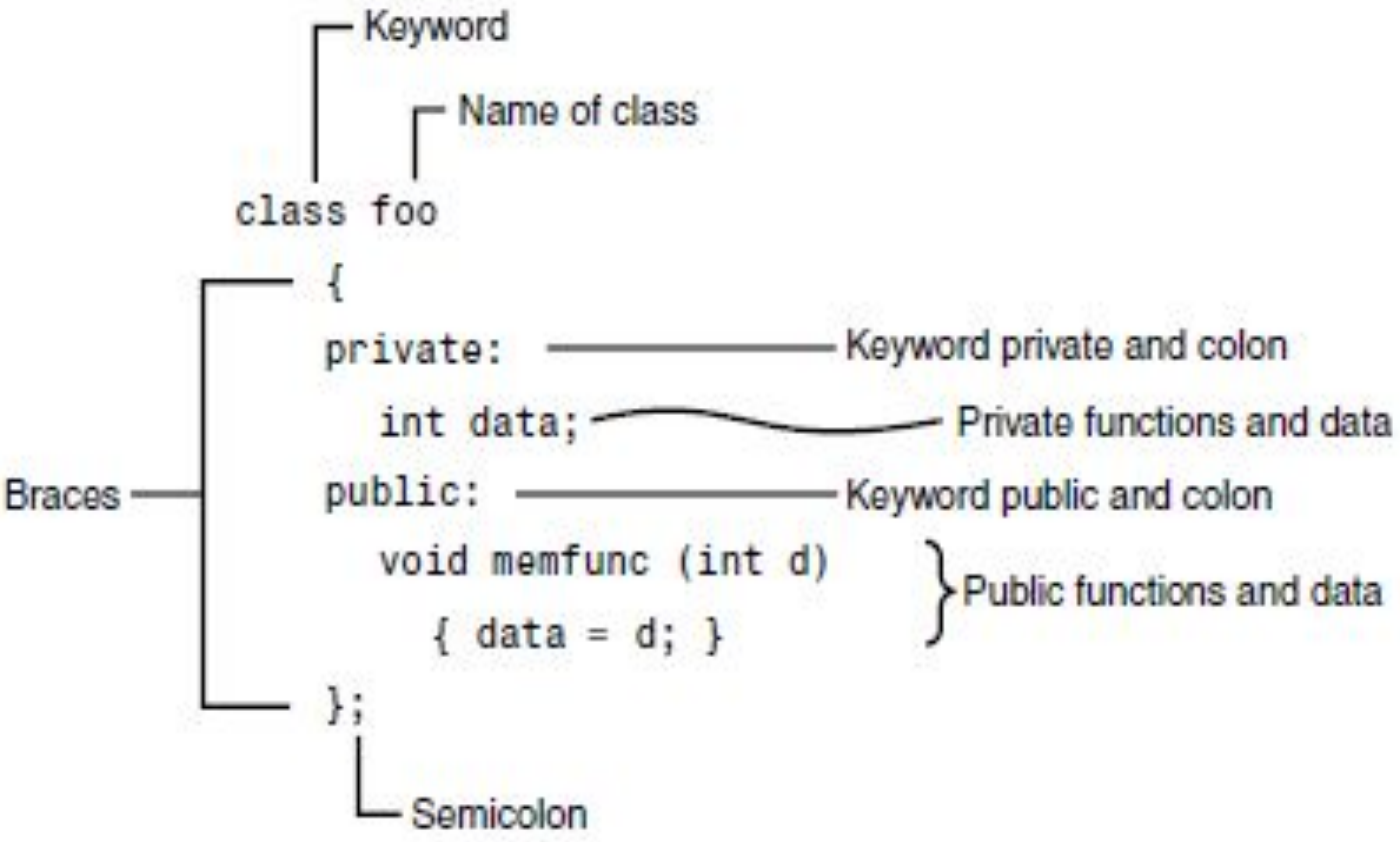


Fig:Syntax of a class definition.

*

Defining the Class (cont..)

Declaring the class **smallobj** which contains one data item and two member functions:

```
class smallobj                    //define a class
{
private:
      int somedata;               //class data or data member
public:
      void setdata(int d) //member function to set data
      {
            somedata = d;
      }
      void showdata()         //member function to display data
      {
            cout << "\nData is " << somedata;
      }
};
```


*

private and public (cont...)

- The data member ***somedata*** follows the keyword private, so it can be accessed from within the class, but not from outside.
- setdata() and showdata() follow the keyword public, they can be accessed from outside the class.
- **Generally Functions Are Public, Data Is Private:**
  - Usually the data within a class is private and the functions are public.
  - The data is hidden so it will be safe from **accidental manipulation**, while the functions that operate on the data are public so they can be accessed from outside the class.

Look back to ***data hiding***

- *data hiding term does not refer to the* activities of particularly paranoid programmers; rather it means that data is concealed within a class so that it cannot be accessed mistakenly by functions outside the class.
- **The primary mechanism for hiding data is to put it in a class and make it private.**
- Private data or functions can only be accessed from within the class. Public data or functions, on the other hand, are accessible from outside the class.

*

Hidden from Whom?

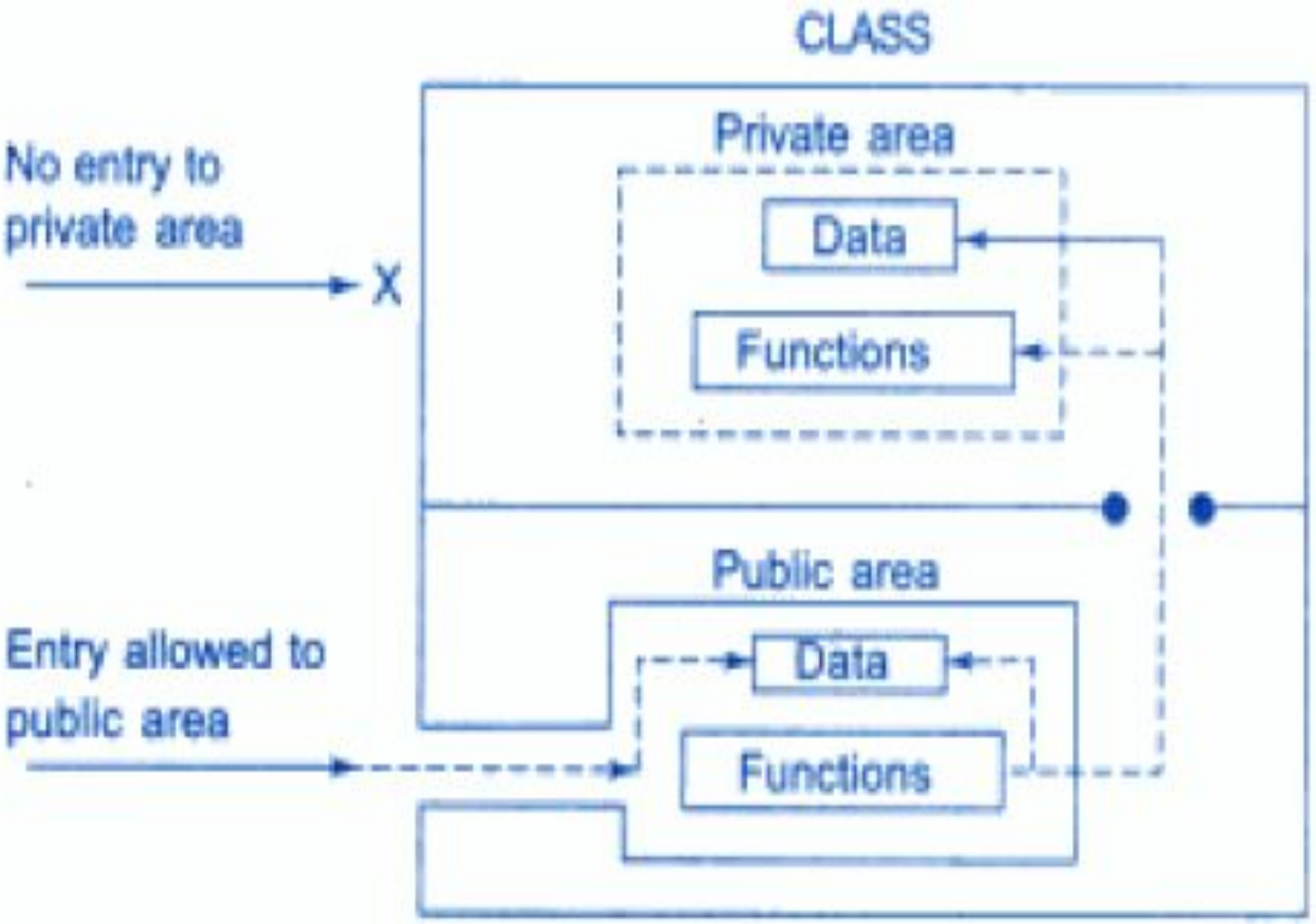
- *Data hiding, on the other hand, means hiding data from parts of the program that don't need to access it. More specifically, one class's data is hidden from other classes.*

- Data hiding is designed to protect well-intentioned programmers from honest mistakes.

- Programmers who really want to manipulate, can figure out a way to access private data, but they will find it hard to do so by accident.

*

private and public access specifier



Accessing members of class

- The data members and member functions of class can be accessed using the dot(".") operator with the object. For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .
- **Accessing Data Members**

  The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object.
- Accessing a data member depends solely on

Accessing members of class

- General syntax:

  Object_name.member_function_name(arguments);

Examples:

Obj1.getData();

Obj2.showData();

Obj3.publicData=200;//public data member can be accessed by object

Following are invalid:

getData();//syntax error

Obj.private_Data;//syntax error

*

Using the Class

```
#include <iostream>
using namespace std;
class smallobj //define a class
{
private:
        int somedata; //class data
public:
        void setdata(int d) //member function to set data
        { somedata = d; }
        void showdata() //member function to display data
        { cout << “Data is “ << somedata << endl; }
};
*********************
int main()
{
        smallobj s1, s2;            //define two objects of class smallobj
        s1.setdata(1066);        //call member function to set data
        s2.setdata(1776);
        s1.showdata();           //call member function to display data
        s2.showdata();
        return 0;
}
```


*

Defining Member Functions

- Member functions can be defined in two ways:

1. **Inside the class definition (Already discussed)**
2. **Outside the class definition:**

General syntax

```
Return_type   class_name ::
  func_name(arguments)
{
//Function body
}
```

:: operator is scope resolution operator which tells to compiler that func_name function belong to class_name .

*

Important point

- Note that all the member functions defined inside the class definition are by default **inline**,
- We can also make any non-class function inline by using keyword **inline** with them.
- Inline functions are actual functions, which are copied everywhere during compilation, like pre-processor macro, so the overhead of function calling is reduced.

*

Nesting of member functions

- A member function of a class can be called only by an object of that class using a dot operator.
- However, there is an exception to this. A member function can be called by using its name inside another member function of the same class. This is known as **nesting of member functions**.
- Example:

```
void abc ::func()
{
Display();
}
```


*

Private member function

- Generally we keep all the data members in private section and function in public section, some time we may need to hide member function from outside calls. Like function Delete_Account, Update_Salary etc.
- A private member function can not be called by the object of the same class; **then how to call private function?**

Private member functions can be called by the other member function of its class.

*

Memory allocation for Objects

- Member functions get memory only once when they are defined.
- Space for all data members are allocated for each object separately.
- Example:

```
class area{
   int l,b,area;
   public:
   void input();
   void output(int,int);
}Obj;
```

sizeof(Obj) will be 12 Bytes

*

Static Data Members

- Data member of class can be static.
- **Static data member has following properties:**

1. It is initialized to zero when the first object of its class is created.
2. Only one copy of static data member is created for the entire class and is shared by all the objects of that class.
3. It is visible only within the class but lifetime is the entire program.
4. Static variable are normally used to maintain values common to entire class

- For making a data member static, we require :
- (a) Declare it within the class.

*

Static Data Members; example

Class student
{
static int count; //declaration within class

};

The static data member is defined outside the class as :

int student :: count; //definition outside class

The definition outside the class is a must.

We can also initialize the static data member at the time of its definition as:

int student :: count = 10;

*

Static Member Function

- A static member function can access only the static members of a class. We can do so by putting the keyword static before the name of the function while declaring it for example,

```
class student
{
   static int count;
   ----------------
   public :
   ----------------
   ----------------
   static void showcount (void)     //static member function
   {
   Cout<<"count="<<count<<"\n";
   } };
```


*

Static Member Function

- In C++, a static member function differs from the other member functions in the following ways:

(i) Only static members (functions or variables) of the same class can be accessed by a static member function.

(ii) It can be called using the **name of the class rather than an object as given below:**

Name_of_the_class :: function_name

For example,

student::showcount();

*

- What is default visibility mode for members of classes in C++ ?

A. Private
B. Public
C. Protected
D. Depends

*

Friend Function

- The concepts of encapsulation and data hiding dictate that nonmember functions should not be able to access an object's private or protected data.
- However, there are situations where such rigid discrimination leads to considerable inconvenience.
- So, there is mechanism built in C++ programming to access private or protected data from non-member function which is **friend function** and **friend class**.

*

Declaration of Friend Function

- The declaration of friend function should be made inside the body of class.

```
class class_name
   {
   ..... .... ........
   friend return_type function_name(argument/s);
   //Declaration
   ..... .... ........
   };
return_type function_name(argument/s)
{
//friend defintion
```


*

Characteristics of friend function

1. The declaration of friend function should be made inside the body of class
2. Friend function declaration can be put anywhere inside class either in **private or public** section without affecting its meaning
3. No keywords is used in function definition of friend function, friend keyword is used only in declaration.
4. friend function can access the private and protected data of the class where it has been declared.
5. A friend function is not in the scope of

Characteristics of friend function(Cont..)

6. It cannot be called using the object of that class, it can be invoked like a normal function without any object

7. Unlike member functions, it cannot use the member names directly.

8. Usually, it has objects as arguments. Although it is possible to write friend function without arguments.

*

Where friend function are used

1. Friend function is especially useful for operator overloading.
2. Friend function allow to operate on objects of two different classes.

- Example:

```
#include <iostream>
using namespace std;
class beta;     //needed for frifunc declaration
class alpha
{
private:
int data;
public:
alpha() { data=3 }  //no-arg constructor
friend int frifunc(alpha, beta); //friend function
};
```


*

Example (cont..)

```
class beta
{
private:
int data;
public:
beta() { data=7 } //no-arg constructor
friend int frifunc(alpha, beta); //friend function
};
int frifunc(alpha a, beta b) //function definition
{
return( a.data + b.data );
}
```

```
int main()
{
alpha aa;
beta bb;
cout << frifunc(aa, bb) << endl; //call the function
return 0;
}
```


*

Cont...

- A minor point: Remember that a class can't be referred to until it has been declared. Class beta is referred to in the declaration of the function frifunc() in class alpha, so beta must be declared before alpha. Hence the declaration

class beta; at the beginning of the program.

- Note the programmer who does not have access to the source code for the class cannot make a function into a friend. In this respect, the integrity of the class is still protected.

*

Friend Function without argument

```
#include <iostream>
class sample {
 private:  int x;
 public:  friend void fun();
};
 void fun()
{ sample s;
    s.x = 50000;
   cout << s.x;
}
int main
fun();
return 0; }
```


*

Friend class

```
class TWO; // forward declaration of the class
   TWO
class ONE
{
.....
.....
public:
.....
.....
friend class TWO; // class TWO declared as
   friend of class ONE
};
```


*

Friend class example

```
class Child {

friend class Mother; //Mother class members
   can access the                private parts of class
   Child.
public: void  Show_name( void );
            void setName( char * newName );

};
```

const Member Functions

- A const member function guarantees that it will never modify any of its class's member data.
- It is recommended practice to make as many functions const as possible so that accidental changes to objects are avoided.

```
class aClass
{
private:
      int alpha;
public:
      void nonFunc()     //non-const member function
      { alpha = 99;      //OK
      }
      void conFunc() const     //const member function
      { alpha = 99;            //ERROR: can't modify a member
      }
};
```


*

const Member Functions

- A function is made into a constant function by placing the keyword const after the declarator but before the function body. If there is a separate function declaration, **const must be used in both declaration and definition.**
- Making a function const helps the compiler flag errors, and tells anyone looking at the listing that you intended the function not to modify anything in its object.

*

Arrays of Objects

- Arrays of variables that are of type class are called arrays of objects
- Declaring Arrays:**Class_name array_name[SIZE];**

Example: class student

```
{   int rollno;
    int marks;
    public:
    void getdata();
            void putdata();
};
```

- The objects created as
  Student BTECH[50]; // array of object of type Student
  Student MTECH[20];

*

WAP to create Array of Objects

```
#include<iostream.h>
class Student
{
  int rollno;
  int marks;
 public:
   void getdata(void);
     void putdata(void);
};
```


*

Define getdata() and putdata() function

```
void student ::
   getdata(void)
{  cout<<“Enter
   Rollno”<<“\n”;
   cin>> rollno;
   cout<<“Enter
   marks”<<“\n”;
   cin>>marks;
}
```

```
void Student::
   putdata(void)
{
   cout<<“
   Rollnumber”<<rollno;
   cout<<“Marks”<<marks;
```

Define main() function

```
int main()
{
 Student Btech[3];
 for(int i=0;i<3;i++)
{ cout<<“ Read Btech Students Detail”;
 Btech[i].getdata();
 }
for(int j=0;j<3;j++)
{ cout<<“\n Btech Details are as follows”;
  Btech[j].putdata();
}
return 0;
}
```

Output of the program

Read Btech Students Detail
Enter Rollno: 1
Enter Marks:25
Enter Rollno: 3
Enter Marks:23
Enter Rollno: 5
Enter Marks:24
Btech Details are as follows:
Rollno:1
Marks:25
Rollno:3
Marks:23
Rollno:5
Marks:24

Objects as function arguments

- An Object can be used as function argument in two ways:
  - A copy of entire Object is passed to the function.(Pass_by_Value)
  - Only the address of the object is transferred to the function.(Pass_by_Reference)

Passing objects as arguments: Call by Value

```
class time
{
 int hours;
 int minutes;
public:
 void gettime(int h, int m){hours=h;minutes=m;}
void puttime(void)
{cout<<hours<<“hrs &”<<minutes<<“mins”<<“\n”;}
void sum(time,time);
};
void time::sum(time t1,time t2)  //outside class
{
 minutes=t1.minutes+t2.minutes;
hours=t1.hours+t2.hours+minutes/60;
minutes=minutes%60;
}
```

Main( ) function

```
int main()
{
   time T1,T2,T3;
   T1.gettime(2,45);
   T2.gettime(3,30);
   T3.sum(T1,T2);
   cout<<"T1= "; T1.puttime();
   cout<<"T2= "; T2.puttime();
   cout<<"T3= "; T3.puttime();
}
```

Output:
T1= 2 Hrs & 45 min
T2= 3 Hrs & 30 min
T3= 6 Hrs & 15 min

Passing objects as arguments: Call by Reference

```
class time
{
 int hours;
 int minutes;
public:
 void gettime(int h, int m){hours=h;minutes=m;}
void puttime(void)
{cout<<hours<<“hrs &”<<minutes<<“mins”<<“\n”;}
void sum(time &,time &);
};
void time::sum(time &t1,time &t2)  //outside class
{
 minutes=t1.minutes+t2.minutes;
hours=t1.hours+t2.hours+minutes/60;
minutes=minutes%60;
}
```

Main( ) function

```
int main()
{
   time T1,T2,T3;
   T1.gettime(2,45);
   T2.gettime(3,30);
   T3.sum(T1,T2);
   cout<<"T1= "; T1.puttime();
   cout<<"T2= "; T2.puttime();
   cout<<"T3= "; T3.puttime();
}
```

Output:
T1= 2 Hrs & 45 min
T2= 3 Hrs & 30 min
T3= 6 Hrs & 15 min

Passing objects as arguments: Call by Pointer(Call by Address)

```
class time
{
 int hours;
 int minutes;
public:
 void gettime(int h, int m){hours=h;minutes=m;}
void puttime(void)
{cout<<hours<<“hrs &”<<minutes<<“mins”<<“\n”;}
void sum(time *,time *);
};
void time::sum(time *t1,time *t2)  //outside class
{
 minutes=t1->minutes+t2->minutes;
hours=t1->hours+t2->hours+minutes/60;
minutes=minutes%60;
}
```

Main( ) function

```
int main()
{
   time T1,T2,T3;
   T1.gettime(2,45);
   T2.gettime(3,30);
   T3.sum(&T1,&T2);
   cout<<"T1= "; T1.puttime();
   cout<<"T2= "; T2.puttime();
   cout<<"T3= "; T3.puttime();
}
```

Output:
T1= 2 Hrs & 45 min
T2= 3 Hrs & 30 min
T3= 6 Hrs & 15 min

Which method to use when?

Advantages of passing by reference:

1. It allows us to have the function change the value of the argument, which is sometimes useful.
2. Because a copy of the argument is not made, it is fast, even when used with large structs or classes.
3. We can pass by **const reference** to avoid unintentional changes.
4. We can return multiple values from a function.

Note: The performances of call by reference and call by address are exactly the same, as

Returning Objects from function

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

```
class complex
{
   float x;
   float y;
public:
   void input(float real,float imag)
   { x=real; y=imag;}
   friend complex sum(complex,complex);
    void show(complex);
};
```

```
complex sum(complex c1,complex c2)
{
        complex c3;
        c3.x=c1.x+c2.x;
        c3.y=c1.y+c2.y;
        return c3;
}
```

Show function defined outside class

```
void complex::
   show(complex c)
{


   cout<<c.x<<“+j”<<c.y<<“
   /n”;
}
```

```
int main()
{
   complex A,B,C;
   A.input(3.1,5.65);
   B.input(2.75,1.2);
   C=sum(A,B);
   cout<<“A=“<<A.show
   (A);
   cout<<“B=“<<B.show
   (B);
   cout<<“C=“<<C.show
   (C);
   return 0;
```

Output:
A=3.1+ j5.65
B=2.75+j1.2
C=5.85+J6.85

Pointers to Data member

- Pointer to data member can hold address of data member of a class.
- Declaration Syntax:

type class_name::* pointer_name=&class_name::DataM;

class_name::* tells pointer to member of class class_name.

&class_name::DataM means "address of DataM of class class_name"

Example:

```
Class student{
public:
int mark;
 void  show(){
      cout<<mark;}
  };
```


*

Pointers to Data member

- student ob;
- student *p=&ob;//p is pointer to object
- ob.mark=80;
- int student::*ptr=&student::mark;//ptr is pointer to member;

1. cout<<ob.*ptr;
2. cout<<ob.mark;
3. cout<<p->*ptr;
4. cout<<p->mark;
5. cout<<(*p).*ptr;
6. cout<<(*p).mark;

*

Predict the output of following program

```
#include <iostream>
using namespace std;
void func(int a, bool flag = true)
{
if (flag == true )
{
cout << "Flag is true. a = " << a;
}
else
{
cout << "Flag is false. a = " << a;
}
}
int main()
{
func(200, true);
return 0;
}
```

Flag is true. a = 200

*

Predict the output of following program

```
#include<iostream>
#include<conio.h>
using namespace std;
class demo{
	int a,b,c;
	public: void getdata()
		{
			cout<<"Enter a,b, and c:";
			cin>>a>>b>>c;
		}
		void f()const;
};
void demo::f() {  cout<<a+b*c; }
int main()
{
	demo ob;
	ob.getdata();
	ob.f();
	getch();
}
```

[Error] prototype for 'void demo::f()' does not match any in class 'demo'

*

Problem

1.

```
#include<iostream>
using namespace std;

class Empty {};

int main()
{
  cout << sizeof(Empty);
  return 0;
}
```

Non Zero

2.

```
class Test {
    int x;
};
int main()
{
  Test t;
  cout << t.x;
  return 0;
}
```

Compilation Error: In C++, the default access is private. Since x is a private member of Test, it is compiler error to access it outside the class.

*

Problem

1.
Which of the following is true?
(A) All objects of a class share all data members of class
(B) Objects of a class do not share non-static members. Every object has its own copy.
(C) Objects of a class do not share codes of non-static methods, they have their own copy
(D) None of the above

Answer: (B)
Explanation: Every object maintains a copy of non-static data members. For example, let Student be a class with data members as name, year, batch. Every object of student will have its own name, year and batch. On a side note, static data members are shared among objects.
All objects share codes of all methods. For example, every student object uses same logic to find out grades or any other method.

*

Problem

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        cout<<1["abcd"];
        getch();
}
```

Answer: b
Explanation: 1[abcd] is treated as *(1+"abcd") which is *("abcd"+1) ie. address of abcd +1 indicating b

*

Problem

```
#include<iostream>
using namespace std;
class Test{
    static int x;
    int *ptr;
    int y;
};
int main(){
    Test t;
    cout << sizeof(t) << " ";
    cout << sizeof(Test *);
}
```

Explanation:
For a compiler where pointers take 4 bytes, the statement "sizeof(Test *)" returns 4 (size of the pointer ptr). The statement "sizeof(t)" returns 8. Since static is not associated with each object of the class, we get (8 not 12).

*

Problem

When one object reference variable is assigned to another object reference variable then
(A) a copy of the object is created.
(B) a copy of the reference is created.
(C) a copy of the reference is not created.
(D) it is illegal to assign one object reference variable to another object reference variable.

Answer: (B)

*

Problem

```
#include <iostream>
 using namespace std;
struct Test {
    int x; // x is public
};
int main()
{
  Test t;
  t.x = 20;
  cout<<t.x;
  return 0;
}
```

Output: 20
Explanation:
Members of a class are private by default and members of struct are public by default.

*

Problem

```
#include<iostream>
using namespace std;
int x = 1;
void fun()
{
    int x = 2;
    {
        int x = 3;
        cout << ::x << endl;
    }
}
int main()
{
    fun();
    return 0;
}
```

Output: 1

Explanation:

The scope resolution operator when used with a variable name, always refers to global variable.

Problem

```
#include<iostream>
using namespace std;

int fun(int x = 0, int y = 0, int z)
{  return (x + y + z); }

int main()
{
   cout << fun(10);
   return 0;
}
```

Output: Error default argument missing for parameter 3 of ‘int fun(int, int, int)’
Explanation:
All default arguments must be the rightmost arguments

*

Problem

Which of the following overloaded functions are NOT allowed in C++?

1)Function declarations that differ only in the return type

```
int fun(int x, int y);
void fun(int x, int y);
```

2) Functions that differ only by static keyword in return type

```
int fun(int x, int y);
static int fun(int x, int y);
```

3)Parameter declarations that differ only in a pointer * versus an array []

```
int fun(int *ptr, int n);
int fun(int ptr[], int n);
```

4) Two parameter declarations that differ only in their default arguments

```
int fun( int x, int y);
int fun( int x, int y = 10);
```

(A) All of the above **(B)** All except 2) **(C)** All except 1) **(D)** All except 2 and 4

Explanation: A

*

Problem

```
#include <iostream>
using namespace std;
class Test{
    static int x;
public:
    Test() { x++; }
    static int getX() {return x;}
};
int Test::x = 0;
int main()
{
    cout << Test::getX() << " ";
    Test t[5];
    cout << Test::getX();
}
```

0 5

Explanation:

Static functions can be called without any object. So the call “Test::getX()” is fine. Since x is initialized as 0, the first call to getX() returns 0. Note the statement x++ in constructor. When an array of 5 objects is created, the constructor is called 5 times. So x is incremented to 5 before the next call to getX().

Problem

If a function is friend of a class, which one of the following is wrong?
(A) A function can only be declared a friend by a class itself.
(B) Friend functions are not members of a class, they are associated with it.
(C) Friend functions are members of a class.
(D) It can have access to all members of the class, even private ones.

Answer: (C)

Explanation:
Friend of the class can be member of some other class but Friend functions are not the members of a particular class.

*

Problem

Which one of the following is correct, when a class grants friend status to another class?

(A) The member functions of the class generating friendship can access the members of the friend class.

(B) All member functions of the class granted friendship have unrestricted access to the members of the class granting the friendship.

(C) Class friendship is reciprocal to each other.

(D) There is no such concept.

Answer: (B)

*

Problem

```
#include <iostream>
int const s=9;
int main()
{
    std::cout << s;
    return 0;
}
```

Answer: 9
Explanation:
The above program compiles & runs fine. Const keyword can be put after the variable name or before variable name. But most programmers prefer to put const keyword before the variable name.

*

Problem

```
#include<iostream>
using namespace std;
class Point {
public:
    Point() { cout << "Constructor called"; }
};

int main()
{
   Point t1, *t2;
   return 0;
}
```

(A) Compiler Error **(B)** Constructor called Constructor called
(C) Constructor called

Answer: (C)

Explanation: Only one object t1 is constructed here. t2 is just a pointer variable, not an object

*