CHAPTER 2

Classes and Methods

Review of structures

- Structure is the collection of variables of different types under a single name for better visualization of problem.
- Structure can hold data of one or more types.

The **struct** keyword defines a structure type followed by an identifier (name of the structure). Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
   char name[50];
   int age;
   float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.

Once you declare a structure person as above. You can define a structure variable as:
Person p1;
Here, a structure variable p1 is defined which is of type structure Person.
When structure variable is defined, only then the required memory is allocated by the compiler. Considering having 16 bit system, the memory of float is 4 bytes, memory of int is 2 bytes and memory of char is 1 byte. Hence, 56 bytes of memory is allocated for structure variable p1.

How to access members of a structure?

The members of structure variable is accessed using a dot (.) operator.
Suppose, you want to access age of structure variable p1 and assign it 25 to it. You can perform this task by using following code below:
p1.age = 25;

Example: C++ Structure

C++ Program to assign data to members of a structure variable and display it.

```
#include <iostream>
using namespace std;
struct Person
{
   char name[50];
   int age;
   float salary;
};
int main()
{
   Person p1;
   cout << "Enter Full name: ";
   cin.get(p1.name, 50);
   cout << "Enter age: ";
   cin >> p1.age;
   cout << "Enter salary: ";
   cin >> p1.salary;
   cout << "Displaying Information." << endl;
   cout << "Name: " << p1.name << endl;
   cout <<"Age: " << p1.age << endl;
   cout << "Salary: " << p1.salary;
   return 0;
}
```

Output

```
Enter Full name: Ram karki
Enter age: 24
Enter salary: 25000.35

Displaying Information.
Name: Ram karki
Age: 24
Salary: 25000.35
```

To read the text containing blank space, cin.get function can be used. This function takes two arguments.
First argument is the name of the string (address of first element of string) and second argument is the maximum size of the array.

Specifying a Class

A class serves as a blue print or a plan or a template for an object. It specifies what data and what functions will be included in objects of that class. Once a class has been defined, we can create any number of objects belonging to that class. An object is an instance of a class. A class binds the data and its associated functions together. It allows the data (and function) to be hidden, if necessary, from external use.

Example:

Class Car	Class Computer
Properties Company, Model Color, Capacity	**Properties** Brand, Price, Screen Resolution HDD size, RAM size
Behavior Speed(), Acceleration(), Break()	**Behavior** Processing(), Display(), Printing()

Declaration of class

Class declaration describes the type and scope of its members. The general form of class declaration is:

```
class class_name
{
private:
variable declaration;
function declaration;
public:
variable declaration;
function declaration;
};
```

Example:

```
class student
{
private:
char name[20];
int roll;
public:
void getinfo();
void display();
};
```

- Here, **class** is C++ keyword; **class_name** is name of class defined by user.
- The body of class enclosed within braces and terminated by a semicolon.
- The class body contains the declaration of variables and functions. The variables declared inside the class are known as data members and the functions are known as member functions. These functions and variables are collectively called as class members.
- Keyword private and public within class body are known as visibility labels. i.e. they specify which of the members are private and which of them are public.
- The class members that have been declared as private can be accessed only from within the class but public members can be accessed from outside of the class also. Using private declaration, data hiding is possible in C++ such that it will be safe from accidental manipulation.
- The use of keyword private is optional. By default, all the members are private. *Usually, the data within a class is private and the functions are public.*
- C++ provides a third visibility modifier, protected, which is used in inheritance. A member declared as protected is accessible by the member functions within its class and any class immediately derived from it.
- **The binding of a data and functions together into a single class-type variable is referred to as encapsulation.**

```
#include<iostream>
using namespace std;
class student
{
private:
char name[20];
int roll;
public:
void getinfo()
{
cout<<"Enter name"<<endl;
cin>>name;
cout<<"Enter Rollno"<<endl;
cin>>roll;

}
void display()
{
cout<<"Name:"<<name<<endl;
cout<<"Roll No:"<<roll<<endl;
}
};
```

```
int main()
{
student st;
st.getinfo();
st.display();
return 0;
}
```

Objects:

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it an object is created memory is allocated. To use the data and access functions defined in the class, we need to create objects.

Syntax for creating an object is

class_name object_name;

e.g.

Student st; // here Student is assumed to be a class name

The statement student st; creates a variable st of type Student. This variable st is known as object of the class Student. Thus, class variables are known as objects.

Key differences between a class and its objects

- Once we have defined a class, it exists all the time a program is running whereas objects may be created and destroyed at runtime.
- For single class there may be any number of objects.
- A class has unique name, attributes, and methods. An object has identity, state, and behavior.

Accessing the class member

The data member functions of class can be accessed using the **dot(.)** operator with the object. For example if the name of object is ***st*** and you want to access the member function with the name ***getInfo()*** then you will have to write ***st.getinfo();***

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 the access control of that data member. This access control is given by **Access modifiers in C++.**

```
class Student
{
private:
int age;
public:
int rollno;
getInfo();
displayInfo();
. . . . . .
}
. . .. . . . . . .. . . .
int main()
{
Student st;
st.age=25;   // error, age is private
st.rollno=5; //ok, roll number is public
st.getinfo(); //Ok
. . . .. . . . .. . . . .
}
```

Defining Member functions

Member function can be defined in two places:

- Outside the class definition
- Inside the class definition

Defining Member function inside the class definition	Defining Member function Outside the class definition
In this case, a member function is defined at the time of its declaration. The function declaration is replaced by the function definition inside the class body.	In this technique, only declaration is done within class body. The member function that has been declared inside a class has to be defined separately outside the class. The syntax for defining the member function is as **return_type class_name::function_name(argument_list)** **{** **//function body** **}** Here, membership label class-name :: tells the compiler that the function function_name belongs to the class class-name. The symbol **::** is called scope resolution operator.

```
class Item
{
private:
int number;
float cost;
public:
void getdata()
{
. . . . . .
}
void display()
{
.. . . . . .
}
};
```

```
class Item
{
private:
int number;
float cost;
public:
void getdata();
void display();
};
void Item::getdata()
{
. . . . . .
}
void Item::display()
{
. . . .. . . . .
}
```

```
#include<iostream>
using namespace std;
class Item
{
private:
int number;
float cost;
public:
 void getdata()
 {
  cout<<"Enter the Item number"<<endl;
  cin>>number;
  cout<<"Enter the cost of item"<<endl;
  cin>>cost;
 }
 void display()
 {
  cout<<"Item number="<<number<<endl;
  cout<<"Cost of item="<<cost<<endl;
 }
};
```

```
#include<iostream>
using namespace std;
class Item
{
 private:
 int number;
 float cost;
 public:
 void getdata();
 void display();
};
void Item::getdata()
{
 cout<<"Enter the Item number"<<endl;
  cin>>number;
  cout<<"Enter the cost of item"<<endl;
  cin>>cost;
}
void Item::display()
{
  cout<<"Item number="<<number<<endl;
  cout<<"Cost of item="<<cost<<endl;
}
```

int main()
{
Item x;
x.getdata();
x.display();
return 0;
}
	int main()
	{
	Item x;
	x.getdata();
	x.display();
	return 0;
	}

In this example, the functions getdata() and display() are defined within body of class item.	In this example, getdata() and display() functions are member functions of class item. They are declared within class body and they are defined outside the class body. While defining member functions, item:: (membership identity label) specifies that member functions belong to the class item. Other are similar to normal function definition.

Some characteristics of the member functions

- Several different classes can use the same function name (function overloading). The membership level will resolve their scope.
- Member functions can access the private data of the class. A non-member function cannot do so.
- A member function can call another member function directly, without using the dot operator.
- A private member function cannot be called by other function that is not a member of its class.

Nesting of member function

A member function can be called by using its name inside another member function of the same class is known as nesting of member functions.

Example:

```
#include<iostream>
using namespace std;
class Largest
{
private:
int a,b;
public:
void getnumber();
int compare();
void display();
};
void Largest::getnumber()
{
 cout<<"Enter two number"<<endl;
 cin>>a>>b;
}
int Largest::compare()
{
if (a>b)
return a;
else
return b;
}
void Largest::display()
{
cout<<"Largest number="<<compare();
}
int main()
{
Largest l;
l.getnumber();
l.display();
return 0;
}
```

Assignment:

Write a program that will demonstrate nested member function. The program that uses the following properties - class name- Addnumber, integer type private member variable- a, b; three member function -inputnumber(), int add(), show(). add() function will add the inputted value a, b like int x=a+b. show() function will call the add() function as a nested function and will display the addition value y as output.

Accessing private member function

Although it is normal practice to place all the data items in a private sections and all the functions in public ,some situations may require certain functions to be hidden(like private data) from outside calls. We can place these functions in the private section.
A private member function can only be called by another function that is member of its class. Even an object cannot invoke a private function using the dot operator.

Consider the class defined below

```
class sample
{
private:
 int m;
void read(); // private member function
public:
void update();
};
```

If s1 is an object of sample, then
s1.read(); // Won't work ;
Object cannot access private members. However, the function read() can be called by the function update() to update the value of m.

```
void sample::update()
{
   read(); // simple call ;no object used
}
```

Example

```
#include<iostream>
using namespace std;
class sample
{
   private:
   int m;
   void read();
   public:
   void update();
   void display();
};
 void sample::read()
 {
 cout<<"Enter the value of m"<<endl;
 cin>>m;
 }
```

```
void sample::update()
{
        read();
        m=m+5;
}
void sample::display()
{
        cout<<"value after updation="<<m<<endl;
}

int main()
{
        sample s1;
        s1.update();
        s1.display();
        return 0;
}
```

Assignment

Write a program that will demonstrate private member function. The program that uses the following properties - class name- sum, integer type private member variable- x, y. One private member function getnumber(), two public member function - int addition(), display(). addition() function will call private member function getnumber() for input number x, y and add the inputted value x, y like int z=x+y. display() function will display the addition value z as output.

Array of object

An object of class represents a single record in memory, if we want more than one record of class type, we have to create an array of object. As we know, an array is a collection of similar date type, we can also have arrays of variables of type class. Such variables are called array of object. First we define a class and then array of objects are declared. An array of objects is stored inside the memory in the same way as multidimensional array.

```
#include<iostream>
using namespace std;
class Employee
{
   char name[25];
   int age;
   float salary;
   public:
   void getdata();
   void display();
};
```

```
 void Employee::getdata()
   {
     cout<<"Enter Employee Name:";
     cin>>name;
     cout<<"Enter Employee Age :";
     cin>>age;
     cout<<"Enter Employee Salary:";
     cin>>salary;
   }
   void Employee::display()
   {
    cout<<"Name:"<<name<<endl;
    cout<<"Age:"<<age<<endl;
    cout<<"Salary:"<<salary<<endl;
   }
int main()
{
int i;
Employee e[10];         //Creating Array of 10 employees
for(i=0;i<10;i++)
{
 cout<<"Enter details of "<<i+1<<" Employee"<<endl;
 e[i].getdata();
}
for(i=0;i<10;i++)
{
cout<<"Details of Employee"<<i+1<<endl;
e[i].display();
}
return 0;
}
```

WAP to define the class in C++ as shown in class diagram .

Student
Name Roll Address Percentage
input() display()

input():to input initial values
display():to display the record of students who passed
Note:-45% is pass percentage and Perform above operations for 5 students.

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

```
class student
{
private:
char name[20],address[20];
int roll;
float per;
public:
void input()
{
cout<<"Enter the name";
cin>>name;
cout<<"Enter the address";
cin>>address;
cout<<"Enter the roll";
cin>>roll;
cout<<"Enter the percentage";
cin>>per;
}
void display()
{
if(per>=45)
{
cout<<"Name="<<name<<endl;
cout<<"Roll="<<roll<<endl;
cout<<"Address="<<address<<endl;
cout<<"Percentage="<<per<<endl;
}
}
};
int main()
{
int i;
student st[5];
for(i=0;i<5;i++)
{
cout<<"Enter the information of student"<<i+1<<endl;
st[i].input();
}
for(i=0;i<5;i++)
{
st[i].display();
}
return 0;
}
```

Object as function arguments

Like any other data type, an object can be used as function argument.

This can be done in two ways:

- A copy of the entire object is passed to the function *(pass-by-value)*
- Only the address of the object is transferred to the function*(pass-by-reference)*

Let us illustrate use of objects as function arguments with the help of program which performs the addition of complex numbers.

```
#include<iostream>
#include<conio.h>
using namespace std;
class Complex
{
private:
int real, imag;
public:
void getcomplex();
void addcomplex(Complex, Complex);
void display();
};

void Complex:: getcomplex()
{
cout<<"Enter real part:"<<endl;
cin>>real;
cout<<"Enter imaginary part:"<<endl;
cin>>imag;
}
void Complex::display()
{
cout<<real<<"+"<<imag<<"i"<<endl;
}
void Complex::addcomplex( Complex c1, Complex c2)
{
real=c1.real+c2.real;
imag=c1.imag+c2.imag;
}
```

```
int main()
{
Complex c1,c2,c3;
cout<<"For first complex number"<<endl;
c1.getcomplex();
cout<<"For second complex number"<<endl;
c2.getcomplex();
c3.addcomplex(c1,c2); //objects first and second passed as argument
cout<<"sum of two complex number=";
c3.display();
getch();
}
```

Since the member function **addcomplex ()** is invoked by the object **c3** with the object **c1** and **c2** as arguments, it can directly access the real and imag variables of **c3**. But, the member of **c1** and **c2** can be accessed by using the dot operator (like c1.real and c1.imag). Therefore, inside the function **addcomplex(),** the variable **real** and **imag** refers to **c3**, c1.real and c1.imag refers to **c1** and c2.real and c2.imag refers to **c2.**

Output:

```
For first complex number
Enter real part:
2
Enter imaginary Part:
4
For second complex number
Enter real part:
1
Enter imaginary Part:
8
Sum of two complex number=3+12i
```

Let us consider another example which uses objects as function arguments with the help of program which performs the addition of time in the hour and minute format.

```
#include<iostream>
using namespace std;
class Time
{
private:
int hours;
int minutes;
public:
void gettime(int h,int m)
{
hours=h;
minutes=m;
}
void display()
{
cout<<hours<<"Hours and";
cout<<minutes<<"Minutes"<<endl;
}
void sum(Time t1,Time t2)
{
minutes = t1.minutes + t2.minutes;
hours = minutes/60;
minutes = minutes%60;
hours = hours + t1.hours + t2.hours;
}
};


int main()
{
Time t1,t2,t3;
t1.gettime(2,45);
t2.gettime(3,30);
t3.sum(t1,t2);
cout<<"First time=";
t1.display();
cout<<"Second time=";
t2.display();
cout<<"sum of two times=";
t3.display();
return 0;
}
```

Since the member function sum () is invoked by the object t3, with the object t1 and t2 as arguments, it can directly access the hours and minutes variables of t3. But, the member of t1and t2 can be accessed by using the dot operator (like t1.hours and t1.minutes). Therefore, inside the function sum(), the variable hours and minutes refers to t3, t1.hours and t1.minutes refers to t1 and t2.hours and t2.minutes refers to t2.

Output:

```
First time=2 Hours and 45 Minutes
Second time=3 Hours and 30 Minutes
Total time=6 Hours and 15 Minutes
```

Returning Object as Arguments

A function cannot only receive object as arguments but also return them. The example in program below illustrates how an object can be created (within function) and returned to another function.

Now, let us consider the program to perform addition of complex number by returning object as an argument

```
#include<iostream>
using namespace std;
class Complex
{
private:
int real;
int imag;
public:
void getdata();
Complex addcomplex(Complex,Complex);
void display();
};

void Complex::getdata()
{
cout<<"Enter the real part:"<<endl;
cin>>real;
cout<<"Enter the imaginary part :"<<endl;
cin>>imag;
}
```

```
void Complex::display()
{
cout<<real<<"+"<<imag<<"i"<<endl;
}
Complex Complex::addcomplex( Complex  c1, Complex c2)
{
Complex temp;
temp.real=c1.real+c2.real;
temp.imag=c1.imag+c2.imag;
return temp;
}

int main()
{
Complex c1,c2,c3,result;
cout<<"For first complex number"<<endl;
c1.getdata();
cout<<"For second complex number"<<endl;
c2.getdata();
result=c3.addcomplex(c1,c2);
cout<<"sum of two complex number=";
result.display();
return 0;
}
```

WAP to perform the addition of distance with data members feet and inches returning object as argument.

```
#include<iostream>
using namespace std;
class Distance
{
private:
int feet;
int inches;
public:
void getdata();
Distance add(Distance,Distance);
void display();
};
```

```
void Distance::getdata()
{
cout<<"Enter feet:"<<endl;
cin>>feet;
cout<<"Enter inches:"<<endl;
cin>>inches;
}
void Distance::display()
{
cout<<feet<<"feet and"<<inches<<"inches"<<endl;
}

Distance Distance::add(Distance d1, Distance d2)
{
Distance temp;
temp.inches=d1.inches+d2.inches;
temp.feet=temp.inches/12;
temp.inches=temp.inches%12;
temp.feet=temp.feet+d1.feet+d2.feet;
return temp;
}

int main()
{
Distance d1,d2,d3,result;
cout<<"Enter information of first distance"<<endl;
d1.getdata();
cout<<"Enter information of second distance"<<endl;
d2.getdata();
result=d3.add(d1,d2);
cout<<"sum of distance=";
result.display();
return 0;
}
```

Function call by one object passing second object as function argument and return third object adding two objects.

WAP to perform the addition of two objects of complex numbers with data members real and imag and a function call by one object passing second object as function argument and return third object adding two objects.[Hint: c3=c1.addcomplex(c2)]

```
#include<iostream>
using namespace std;
class Complex
{
private:
int real;
int imag;
public:
void getdata();
Complex addcomplex(Complex);
void display();
};
void Complex::getdata()
{
cout<<"Enter the real part:"<<endl;
cin>>real;
cout<<"Enter the imaginary part :"<<endl;
cin>>imag;
}
void Complex::display()
{
cout<<real<<"+"<<imag<<"i"<<endl;
}
Complex Complex::addcomplex( Complex  c2)
{
Complex temp;
temp.real=real+c2.real;
temp.imag=imag+c2.imag;
return temp;
}
```

```
int main()
{
Complex c1,c2,c3;
cout<<"For first Complex number"<<endl;
c1.getdata();
cout<<"For second Complex number"<<endl;
c2.getdata();
c3=c1.addcomplex(c2);
cout<<"sum of two Complex number=";
c3.display();
return 0;
}
```

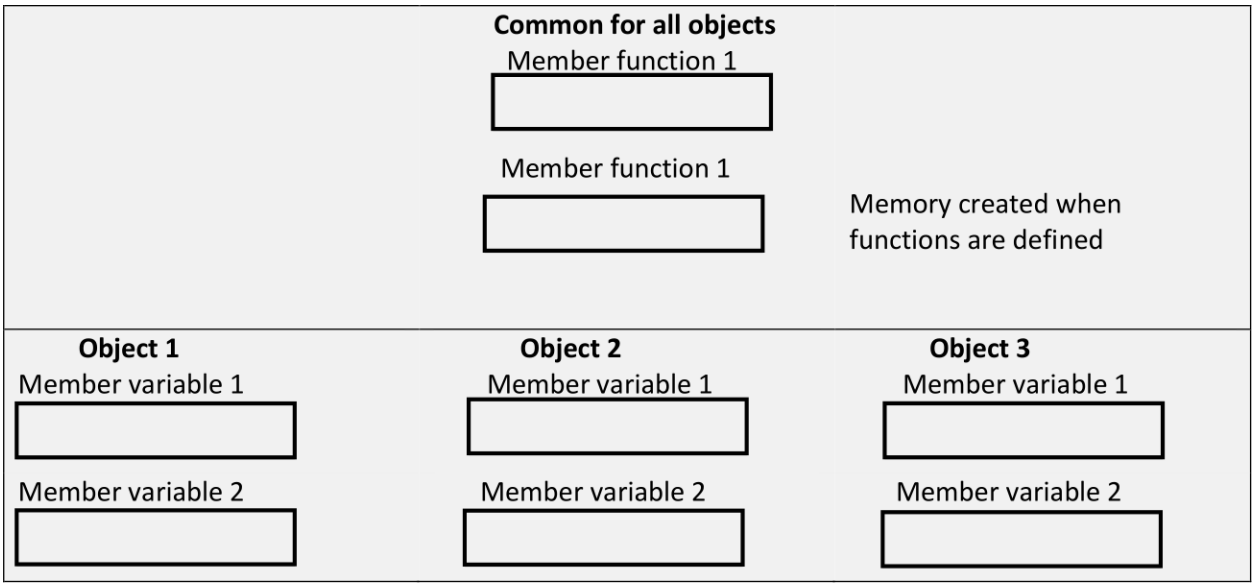
Assignment:

- WAP to perform the addition of time in hours, minutes and seconds format by passing object as an argument.
- WAP to perform the addition of time in hours, minutes and seconds format by returning object as an argument.
- WAP to create two time objects with data members hours, minutes and seconds a function call by one object passing second object as function argument and return third object adding two objects. *Hint:t3=t1.adddistance(t2);*
- WAP to create two distance objects with data members feet, inches and a function call by one object passing second object as function argument and return third object adding two objects. *Hint:d3=d1.adddistance(d2);*

Memory allocation for objects

For each object memory spaces for member variables is allocated separately, because the member variables will hold different data value for different objects.

The member functions are created and placed in the memory space only once when they are defined as a part of class specification. Since, all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when objects are created.



Friend function

The private member of a class cannot be accessed from outside the class i.e. a nonmember function cannot access to the private data of a class. But may be in some situation one class wants to access private data of second class and second wants to access private data of first class, or may be an outside function wants to access the private data of a class. However, we can achieve this by using friend functions.

The non-member function that is "friendly" to a class, has full access rights to the private members of the class.

To make an outside function friendly to a class, we have to declare this function as a friend function. Declaration of friend function should be preceded by the keyword **friend**.

The friend function declares as follows:

```
class ABC
{
- - - - - -
- - - - - -
public :
- - - - - -
- - - - - -
friend void xyz() ;     //declaration
} ;
```

We give a keyword friend in front of any members function to make it friendly.

```
void xyz() //function definition
{
//function body
}
```

It should be noted that a function definition does not use friend keyword or scope resolution operator (: :). A function can be declared as friend in any number of classes.

Characteristics of a Friend Function:

- It is not in the scope of the class to which it has been declared as friend. That is why, it cannot be called using object of that class.
- It can be invoked like a normal function without the help of any object.
- It cannot access member names (member data) directly and has to use an object name and dot membership operator with each member name.(eg.A.x)
- It can be declared in the public or the private part of a class without affecting its meaning.
- Usually, it has the objects as arguments

Program to illustrate the use of friend function

```
#include <iostream>
using namespace std;
class sample
{
private:
int a,b;
public:
void setvalue(int x,int y )
{
a=x;
b=y;
}
void dispay()
{
        cout<<"value of a="<<a<<endl;
        cout<<"value of b="<<b<<endl;
}
friend void sum(sample s) ;
};
void  sum(sample s)
{
cout<<"sum of two numbers="<<(s.a+s.b);
}
```

```
int main( )
{
sample s1 ;
s1.setvalue(5,10) ;
s1.dispay();
sum(s1);
return 0 ;
 }
```

Create classes called class1 and class2 with each of having one private member .Add member function to set a value(say setvalue) one each class. Add one more function max() that is friendly to both classes. max() function should compare two private member of two classes and show maximum among them. Create one-one object of each class and then set a value on them. Display the maximum number among them.[PU:2015 fall,2016 fall

```
#include <iostream>
using namespace std;
class class2;
class class1
{
private:
int x;
public:
void setvalue (int num)
{
x=num ;
}
friend void max (class1, class2);
};

class class2
{
private:
int y ;
public:
void setvalue(int num)
{
 y=num ;
}
friend void max(class1, class2) ;
} ;
```

```
void max(class1 m, class2 n)
{
if (m.x>n.y)
cout<<"Maximum value="<<m.x ;
else
cout<<"Maximum value="<<n.y ;
}
int main( )
{
class1 p;
class2 q;
p.setvalue(10) ;
q.setvalue(20) ;
max(p, q);
return 0;
}
```

WAP using friend function to Add Data Objects of two Different Classes

OR

Addition of two private data of different classes using friend function.

```
#include <iostream>
using namespace std;
class ABC;        // Forward declaration
class XYZ
{
private:
int x;
public:
void setdata (int num)
{
x=num ;
}
friend void add (XYZ,ABC);
};
class ABC
{
private:
int y ;
```

```
public:
void setdata (int num)
{
 y=num ;
}
friend void add(XYZ,ABC) ;
} ;

void add (XYZ m, ABC n)          // Definition of friend
{
cout<<"Sum ="<<(m.x+n.y);
}
int main( )
{
XYZ p;
ABC q;
p.setdata(15) ;
q.setdata(20) ;
add(p,q);
return 0;
}
```

Output:

```
Sum=35
```

Alternative solution

```
#include <iostream>
using namespace std;
class ABC; // Forward declaration
class XYZ
{
public:
int x;
void getdata ()
{
cout<<"Enter Value of class XYZ"<<endl;
cin>>x;
}
friend void add (XYZ,ABC);
};
```

```
class ABC
{
int y ;
public:
void getdata ()
{
cout<<"Enter the value of class ABC"<<endl;
cin>>y;
}
friend void add(XYZ,ABC) ;
} ;
void add (XYZ m, ABC n)
{
cout<<"Sum ="<<(m.x+n.y);
}
int main( )
{
XYZ p;
ABC q;
p.getdata() ;
q.getdata() ;
add(p,q);
return 0;
}
```

Swapping Private data of classes

```
#include <iostream>
using namespace std;
class ABC; // Forward declaration
class XYZ
{
private:
int x;
public:
void getdata ()
{
cout<<"Enter Value of class XYZ"<<endl;
cin>>x;
}
void display()
{
cout<<"value1="<<x<<endl;
}
```

```
friend void swap (XYZ &,ABC &);
};
class ABC
{
private:
int y ;
public:
void getdata ()
{
cout<<"Enter the value of class ABC"<<endl;
cin>>y;
}
void display()
{
cout<<"value2="<<y<<endl;
}
friend void swap(XYZ &,ABC &) ;
};
void swap (XYZ &m, ABC &n) // Definition of friend
{
int temp;
temp=m.x;
m.x=n.y;
n.y=temp;
}

int main( )
{
XYZ p;
ABC q;
p.getdata() ;
q.getdata() ;
cout<<"Value before swapping"<<endl;
p.display();
q.display();
swap(p,q);
cout<<"Value after swapping"<<endl;
p.display();
q.display();
return 0;
}
```

Friend class

We can declare all the member functions of one class as the friend functions of another class. In such cases, the class is called friend class. A friend class can use all the data member of a class for which it is friend. For example

```
class B;
class A
{
…….
friend class B;
};
```

Here, class B can use all the data member of class A.

Example:

```
#include<iostream>
using namespace std;
class B;            //forward declaration
class A
{
int x,y;
public:
void getdata()
{
cout<<"Enter the value of x and y:"<<endl;
cin>>x>>y;
}
friend class B;
};
class B
{
int z;
public:
void getdata()
{
cout<<"Enter the value of z:";
cin>>z;
}
void sum(A t)
{
cout<<"Sum of x,y and z ="<<(t.x+t.y+z)<<endl;
}
};
```

```
int main()
{
A  p;
B q;
p.getdata();
q.getdata();
q.sum(p);
return 0;
}
```

Inline functions

When function is called, it takes a lot of extra time in executing a series of instructions for tasks such as jumping to the function, saving register, pushing arguments into the stack, and returning to the calling function. When a function is small, a substantial percentage of execution time may be spent in such overheads.
To eliminate the cost of calls to small functions, C++ proposes 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 the function call with the corresponding function code.

A function is made inline by using a keyword **inline** before the function definition. Inline functions must be defined before they are called.

Example

```
#include<iostream>
#include<conio.h>
using namespace std;
inline int max(int a, int b)
{
return (a>b)?a:b;
}
int main()
{
int x,y;
cout<<"Enter x and y:"<<endl;
cin>>x>>y;
cout<<"Maximum  value is:"<<max(x,y);
getch();
return 0;
}
```

Inline functions provide following advantages:

1) Function call overhead doesn't occur.
2) It also saves the overhead of push/pop variables on the stack when function is called.
3) It also saves overhead of a return call from a function.
4) When you inline a function, you may enable compiler to perform context specific optimization on the body of function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of calling context and the called context

Note:
Some of the situations where inline expansion may not work are

- For functions returning values, if a loop, a switch or a goto exists.
- For functions not returning values, if a return statement exists.
- If function contain static variables.
- If inline functions are recursive.

Static data member

A data member of class can be qualified as static using the prefix static. A static member variable has following characteristics. They are:

- Static member variable of a class is **initialized to zero** when the first object of its 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, no matter how many object are created.
- It is **visible only within the class** but its **life time is the entire program.**

The type and scope of each static member variable must be defined outside the class definition.

Static variables are normally used to maintain values common to the entire class. For example, a static data member can be used as a counter that records the occurrences of all the objects.

Example:

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

```
class item
{
static int count ; // static member variable
int number ;
public:
void getdata (int a)
{
number=a;
count++ ;
}
void displaycount()
{
cout<<"count:"<<count<<endl;
}
};
int item ::count;   //static member definition

int main()
{
item x,y,z ;                //count is initialized to zero
x.displaycount();          // display count
y.displaycount();
z.displaycount();
x.getdata(10) ;            //getting data into object x
y.getdata(20) ;             //getting data into object y
z.getdata(30) ;            //getting data into object z
cout<<"After reading data"<<endl ;
x.displaycount();         // display count
y.displaycount();
z.displaycount();
return 0;
}
```

Output:

```
Count:0
Count:0
Count:0
After reading data
Count:3
Count:3
Count:3
```

Static Member functions

Like static member variables, we can also have static member functions.
A member function that is declared static has the following properties.

- A static function 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 objects) as follows.

class-name::function name;

```
#include<iostream>
using namespace std;
class test
{
private:
int code;
static int count;
public:
void setcode()
{
code=++count;
}
void showcode()
{
cout<<"code="<<code<<endl;
}
static void showcount()
{
cout<<"count="<<count<<endl;
}
};
int test::count;
```

```
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;
}
```

Output:

```
Count=2
Count=3
Code=1
Code=2
Code=3
```

Reference Variable

A reference variable provides an alias(alternative name) for a previously defined variable.
A reference variable can be created as follows:

data-type &reference-name = variable name;

eg float total = 100;
float &sum = total; // creating reference variable for 'total'.

Example:

```
#include<iostream>
using namespace std;
int main()
{
        float total=100;
        float &sum=total;
        cout<<"Total="<<total<<endl;
        cout<<"Sum="<<sum<<endl;
        total=total+100;
        cout<<"Total="<<total<<endl;
        cout<<"Sum="<<sum<<endl;
        return 0;
}
```

Output:

```
Total=100
Sum=100
Total=200
Sum=200
```

In the above example, we are creating a reference variable 'sum' for an existing variable 'total'. Now these can be used interchangeably. Both of these names refer to same data object in memory. If the value is manipulated and changed using one name then it will change for another also. Eg- the statement
total = total + 100; will change value of 'total' to 200. And it will also change for 'sum'. So the statements
cout<<total;
cout<<sum; both will print 200. This is because both the variables use same data object in memory.

- A reference variable must be initialized at the time of declaration, since this will establish correspondence between the reference and the data object which it names.
- The symbol & is not an address operator here. The notation float & means reference to float type data.

Major application of reference variable is in passing arguments to function. Consider the following example.

```
#include<iostream>
using namespace std;
void fun(int &x);
int main()
{
int m;
m=10;
fun(m);
cout<<"Updated value="<<m;
return 0;
}
void fun(int &x)
{
x=x+10;
}
```

When the function call fun(m) is executed, the following initialization occurs:
int &x=m;
Thus x becomes an alias of m after executing the statement.
fun(m);

such function calls are known as call by reference. Since the variables x and m are aliases, when the function increments **x, m** is also incremented. The value of m becomes 20 after the function is executed.

Default argument

C++ allows a function to assign a parameter the default value in case no argument for that parameter is specified in the function call. Default values are specified when the function is declared. Default value is specified in a manner syntactically similar to a variable initialization. If a value for that parameter is not passed when the function is called, the default value is used, but if a value is specified, this default value is ignored and the passed value is used instead.

For example:

float amount(float principal, int period, float rate=0.15);

The above prototype declares a default value of 0.15 to the argument rate.
A subsequent function call like
value=amount(5000,7); // one argument missing
Passes the value 5000 to principal and 7 to period and then lets a function use default value of 0.15 for rate,
The call
Value=amount(5000,5,0.12); // no missing argument
Passes the explicit value of 0.12 to rate.

Note:

- Default argument are useful in situations where some arguments always have the same value. Eg. bank interest may remain the same for all customers for a particular period of deposit.
- Only the trailing arguments can have default values and therefore we must add defaults from right to left.
- We cannot provide a default value to a particular argument in the middle of an argument list.

```
int add (int a, int b = 5, int c); // illegal
int add (int a = 5, int b, int c = 6); //  illegal
int add (int a  int b , int c = 5); // legal
int add (int a = 5, int b = 6, int c = 7); // legal
```

Example:

```
#include<iostream>
#include<conio.h>
using namespace std;
int add(int a=1, int b=2, int c=3);
int main()
{
int x=5,y=10,z=15;
cout<<"Sum="<<add(x,y,z)<<endl;
cout<<"Sum="<<add(x,y)<<endl;
cout<<"Sum="<<add(x)<<endl;
cout<<"Sum="<<add()<<endl;
getch();
return 0;
}
```

```
int add(int a, int b, int c)
{
return (a+b+c);
}
```

Output:
Sum=30
Sum =18
Sum =10
Sum =6

State and Behavior of Object

State

- State tells us about the type or the value of that object**.**
- It tells, "**What the objects have?"**
  Example: Student have a first name, last name, age, etc...
- An objects state is defined by the attributes (i.e. data members or variables) of the object.
- In OOP state is defined as data members.
- It is determined by the values of its attributes.

Behavior

- Behavior tells us about the operations or things that the object can perform.
- It tells **"What the objects do"**
- Example Student attend a course "OOP", "C programming" etc…
- An objects behavior is defined by the methods or action (i.e. Member functions) of the object.
- In OOP behaviors are defined as member function
- It determines the actions of an object.

Let us consider another example

Imagine a dog. This is an example of an object.

A state can be on or off. If it's asleep, its state is off. If it's awake, the state is on.

Behavior: If the dog is awake, what is it doing? Examples of behavior might be barking, sniffing, eating or drinking, pawing, jumping, playing, to name a few.

Responsibility of Object

An object must contain the data (attributes) and code (methods) necessary to perform any and all services that are required by the object. This means that the object must have the capability to perform required services itself or at least know how to find and invoke these services. Rather than attempt to further refine the definition, take a look at an example that illustrates this responsibility concept.

Consider standalone application when looking for an example to illustrate object responsibility. One of my favorite examples is that of a generic paint program that allows a user to select a shape from a pallet and then drag it to a canvas, where the shape is dropped and displayed. From the user's perspective, this activity is accomplished by a variety of mouse actions. First, the user hovers over a specific shape with the mouse (perhaps a circle), right clicks, drags the shape to a location on the canvas and then lifts his/her finger off the mouse.

This last action causes the shape to be placed at the desired location. In fact, what is really happening when our user's finger is lifted from the mouse button? We can use this action to illustrate our object responsibility concept.

Let's assume that when the finger is lifted from the mouse button that the resulting message from this event is simply: draw. Consider the mouse as an object and that the mouse is only responsible for itself as well. The mouse can, and should only do mouse things. In this context, when the finger is lifted from the mouse button, the mouse thing to do is to draw. But draw what? Actually, the mouse doesn't care. It is not the mouse's responsibility to know how to draw anything. Its responsibility is to signal when to draw, not what to draw.
If this is the case (remember that a circle was selected), how does the right thing get drawn? The answer is actually pretty straightforward; the circle knows how to draw itself. The circle is responsible for doing circle things—just like the mouse is responsible for doing mouse things. So, if you select a circle, the circle object that is created knows everything possible about how to use a circle in the application.

This model has a major implication. Because the mouse does not need to know anything about specific shape objects, all it needs to do to draw a shape is to send the message draw. Actually, this will take the form of a method called draw( ). Thus, more shapes can be added without having to change any of the mouse's behavior. The only constraint is that any shape object installed in the application must implement a draw( ) method. If there is no draw( ) method, an exception will be generated when the mouse mouse-up action attempts to invoke the object's non-existent draw( ) method. In this design methodology, there is a de facto contract between the mouse class and the shape class and these contracts are made possible by the powerful object-oriented design technique called polymorphism.

Previous Board Exam Questions

1) Declare a C++ structure (Program) to contain the following piece of information about cars on a used car lot: [**PU:2013 spring]**

i. Manufacturer of the car
ii. Model name of the car
iii. The asking price for the car
iv. The number of miles on odometer

2) Differentiate between class and structure. Explain them with example.**[PU:2010 spring]**
3) What sorts of shortcomings of structure are addressed by classes? Explain giving appropriate example.**[PU:2014 fall]**
4) Differentiate between structure and class. Why Class is preferred over structure? Support your answer with suitable examples.**[PU:2016 fall]**
5) Differentiate between structure and class in C++?What are the various access specifiers in C++?[**PU:2019 spring]**
6) Explain the various access specifiers used in C++ with an example. **[PU:2010 fall][PU:2016 spring]**
7) What is information hiding? What are the access mode available in C++ to implement different levels of visibility? Explain through example.**[PU:2014 spring][PU:2016 fall]**
8) What is data hiding? How do you achieve data hiding in C++? Explain with suitable program.**[PU:2019 fall]**
9) What is encapsulation? How can encapsulation enforced in C++?Explain with suitable example code. **[PU:2017 spring]**
10) What are the common typed of function available in C++? Define the 3 common types of functions in C++ with a program. **[PU:2015 spring]**
11) What is a function? Discuss the use of friend function taking into consideration the concept of data hiding in object oriented programming.**[PU:2009 spring]**
12) Does friend function violate the data hiding? Explain briefly**.[PU:2017 fall]**
13) "Friend function breaches the encapsulation." Justify. Also mention the use of friend function. **[PU:2015 spring]**
14) Where do you use friend function? **[PU:2014 spring]**
15) What are the merits and demerits of friend function? **[PU:2009 fall]**
16) Private data and function of a class cannot be accessed from outside function. Explain how is it possible to access them with reference of an example. **[PU:2018 fall]**
17) What are the advantages of using friend function? List different types of classes and explain any two. **[PU:2010 spring]**
18) What are the advantages and disadvantages of using friend function? Explain with example program.**[PU:2018 fall]**
19) Illustrate the role of friend function in object oriented programming with its pros and cons. Also write suitable program.**[PU:2019 spring]**

20) Friend function is the non-member of the class. Justify this statement with suitable example.**[PU:2020 fall]**
21) What is inline function? Explain its importance with the help of example program. **[PU:2015 fall]**
22) What is the role of static data in C++ classes? Give example.**[PU:2006 spring]**
23) What do you mean by static member of a class ?Explain the characteristic of static data member.**[PU:2013 fall][PU:2017 fall]**
24) When and how do we make use of static data members of a class? Differentiate between virtual functions, friend functions and static member functions**.[PU:2013 spring]**
25) What are the static data member and static member functions? Show their significance giving examples.**[PU:2014 fall]**
26) Write short notes on:
    - Friend function
      [PU:2006 spring][PU:2013 spring] [PU:2017 spring][PU:2005 fall]
    - Inline function**[PU:2009 fall][PU:2010 spring][PU:2013 spring][PU:2019 spring]**
    - Reference variable
    - Default argument

Programs

1. Declare a C++ structure (Program) to contain the following piece of information about cars on used car lot. [PU:2013 spring]

i. Manufacturer of the car
ii. Model name of the car
iii. The asking price of car
iv. The number of miles on odometer

2. Create a class called Employee with three data members (empno , name, address), a function called readdata() to take in the details of the employee from the user, and a function called displaydata() to display the details of the employee. In main, create two objects of the class Employee and for each object call the readdata() and the displaydata() functions. **[PU:2005 fall]**
3. Create a class called student with three data members (stdnt_name[20],faculty[20],roll_no), a function called readdata() to take the details of the students from the user and a function called displaydata() to display the details the of the students. In main, create two objects of the class student and for each object call both of the functions. **[PU:2010 fall]**
4. Modify the **Que.no 2** for 20 students using array of object.

5. WAP to perform the addition of time in hours, minutes and seconds format.
6. WAP to perform the addition of time using the concept of returning object as argument.
7. WAP to create two time objects with data members hours, minutes and seconds a function call by one object passing second object as function argument and return third object adding two objects. *Hint:t3=t1.adddistance(t2);*
8. WAP to create two distance objects with data members feet, inches and a function call by one object passing second object as function argument and return third object adding two objects. *Hint:d3=d1.adddistance(d2);*
9. Create a class called Rational having data members nume and deno and using friend function find which one is greater.
10. WAP to add the private data of three different classes using friend function.
11. Write a program to find the largest of four integers .your program should have three classes and each classes have one integer number.[**PU:2014 spring]**
12. WAP to swap the contents of two variables of 2 different classes using friend function.
13. WAP to add two complex numbers of two different classes using friend function.
14. WAP to add complex numbers of two different classes using friend class.
15. Using class write a program that receives inputs principle amount, time and rate. Keeping rate 8% as the default argument, calculate simple interest for three customers.**[PU:2019 fall]**
16. Create a new class named City that will have two member varibles CityName (char[20]), and DistFromKtm (float).Add member functions to set and retrieve the CityName and DistanceFromKtm separately. Add new member function AddDistance that takes two arguments of class City and returns the sum of DistFromKtm of two arguments. In the main function, Initialize three city objects .Set the first and second City to be pokhara and Dhangadi. Display the sum ofDistFromKtm of Pokhara and Dhangadi calling AddDistance function of third City object. **[PU: 2010 Spring]**
17. Create a class called Volume that uses three Variables (length, width, height) of type distance (feet and inches) to model the volume of a room. Read the three dimensions of the room and calculate the volume it represent, and print out the result .The volume should be in (feet3) form ie. you will have to convert each dimension into the feet and fraction of foot. For instance , the length 12 feet 6 inches will be 12.5 ft)
    [PU: 2009 spring]
18. WAP to read two complex numbers and a function that calls by passing references of two objects rather than values of objects and add into third object and returns that object.
19. WAP in C++ to calculate simple interest from given principal, time and rate. Set the rate to 15 % as default argument when rate is not provided.
20. Create a class student with six data members (roll no, name, marks in English,math, science and total).Write a program init() to initializes necessary data members calctotal() and display().Create a program for one student (i.e one object only necessary)