Tutorial and Old Question solution

Object Oriented Programming in

in

C++

Prepared By: Pradip Paudel

Email: pradippaudel89@gmail.com

(BASED ON REVISED SYLLABUS OF POKHARA UNIVERSITY)

UNIT-1(THEORY SOLUTION)

1. **Explain the notation "Everything is an object" in an object oriented Programming. [PU:2017 spring]**

   An entity that has state and behavior is known as an object. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior. If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.
   In object-oriented programming problem is analyzed in terms of object and nature of communication between them. Program object should be chosen such that they match closely with real-world objects. In object-oriented programming object's state is stored as data member and behavior is defined using functions (methods).

   When a program is executed, the object interacts by sending message to one another. For example, if "Customer" and "account" are two objects in a program then the customer object may send a message to the account object requesting for the bank balance. Each object contains data, and code (methods) to manipulate data. Objects can interact without having to know details of each other's data or code. It is sufficient to know the type of message accepted, and the type of response returned by objects. So, from above discussion we can say that "Everything is an object" in an object oriented programming.

2. With the help of object-oriented programming, explain how can object oriented programming cope in solving complex problem. [PU:2014 spring][PU:2018 fall]

When the software sizes grows large, due to the interconnections of software components the complexity of software system increases. The interconnections of software components means the dependence of one portion of code on another portion. Due to complexity, it is difficult to understand program in isolation as well it makes difficult to divide a task between a several programs.
The abstraction mechanism is used to manage complexity. The Abstraction is mechanism displays only essential information and hiding the details.
Abstraction is often combined with a division into components. For example, we divided the automobile into the engine and the transmission. Components are carefully chosen so that they can encapsulate certain key features and interact with another component through simple and fixed interface.

The division of components means we can divide large task into smaller problems that can then be worked on more less or less independently of each other. It is a responsibility of a developer of a component to provide a implementation that satisfies the requirement of interface. So, set of procedure written by one programmer can be used by many other programmers without knowing the exact details of implementation. They needed only the necessary interface.

From above discussion we can conclude that, breaking down the system in a way such that component can be reuse in solution of different problems just by knowing only the interfaces which must utilize. The implementation details can be modified in future without affecting the program. So program can also easily maintained. In this way object oriented programming cope in solving complex problem.

3. What influence is an object oriented approach said to have on software system design? What is your own opinion? Justify through example. [PU:2009 fall]

Object oriented approach contributes greater programmer productivity, better quality of software and lesser maintenance cost.
The influence of an object-oriented approach is discussed below with certain characteristic of OOP which justifies that object-oriented approach leads to better software design.
• It is easy to model a real system as programming objects in OOP represents real objects. the objects are processed by their member data and functions. It is easy to analyze the user requirements.
• Complexity that arises due to interconnections of software components can be managed by abstraction. So, set of procedure written by one programmer can be used by many other programmers without knowing the exact details of implementation. They needed only the necessary interface.
• It modularize the programs. Modular programs are easy to develop and can be distributed independently among different programmers.
• Large problems can be reduced to smaller and more manageable problems. It is easy to partition the work in a project based on objects.
• It makes software easier to maintain. Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.
• Elimination of redundant code due to inheritance, that is, we can use the same code in a base class by deriving a new class from it. The reusability feature of OOP lowers the cost of software development.
• Reuse also enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.
• It provides a good framework for code libraries where the supplied software components can be easily adapted and modified by the programmer. This is particularly useful for developing graphical user interfaces.
• In OOP, data can be made private to a class such that only member functions of the class can access the data. This principle of data hiding helps the programmer to build a secure program that cannot be invaded by code in other part of the program.
• With the help of polymorphism, the same function or same operator can be used for different purposes. This helps to manage software complexity easily.

4. What is the significance of forming abstractions while designing an object oriented System. Explain with a example. [PU:2015 fall]

Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Consider a real life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car.

The significance of forming abstractions while designing an object oriented system can be listed as follows:

• It manage complexity that arises due to interconnections of software components.

• Set of procedure written by one programmer can be used by many other programmers without knowing the exact details of implementation. They needed only the necessary interface.

• Data abstraction increases the reusability of the code by avoiding any chances of redundancy.

• It increases the readability of the code as it eliminates the possibility of displaying the complex working of the code.

• With the implementation of classes and objects, comes enhanced security. Since data abstraction is a method of implementing classes and objects any denying access to other classes of accessing the data members and member functions of the base class.

• Helps the user to avoid writing the low level code.

• It separates the entire program into code and implementation making it more comprehensible.

• Can change internal implementation of class independently without affecting the user level code.

• Helps to increase security of an application or program as only important details are provided to the user.

• To increase security of an application or program as only important details are provided to the user.

5.In your opinion how does the object-oriented thinking solve the software crisis? List the basic characteristics of object oriented programming and explain any two characteristics that are directly applicable for software maintenance.

==(Students are encouraged to study about software crisis first== *Ref: E. Balaguruswamy book)*

In my opinion, object-oriented thinking can help solve the software crisis by providing a more modular and flexible approach to software design, development, and maintenance. Object-oriented programming (OOP) emphasizes the use of objects, classes, inheritance, and polymorphism to create reusable and scalable software components. By breaking down complex systems into smaller, more manageable pieces, OOP can help reduce the complexity and improve the maintainability of software.

Here are some basic characteristics of object-oriented programming:

Abstraction: OOP allows developers to create abstract representations of real-world objects and concepts, and to focus on the essential features and behaviors of those objects. This can help simplify the design and implementation of software systems.

Encapsulation: OOP allows developers to encapsulate data and behavior within objects, and to control access to those objects through well-defined interfaces. This can help improve the security, reliability, and maintainability of software systems.

Inheritance: OOP allows developers to define new classes based on existing ones, and to inherit and reuse their properties and methods. This can help reduce code duplication and improve code reuse.

Polymorphism: OOP allows developers to create multiple implementations of the same interface or method, and to choose the appropriate implementation at runtime based on the type of object being used. This can help make software systems more flexible and extensible.

Two characteristics of OOP that are directly applicable for software maintenance are inheritance and polymorphism.

Inheritance: This allows developers to create new classes based on existing ones, and to reuse and extend the properties and methods of those classes. This can help reduce code duplication and improve code reuse, which in turn can help simplify maintenance tasks. For example, if a bug is found in a base class, fixing it in the base class can automatically fix it in all the derived classes that use it. Similarly, if a new feature is added to a base class, it can be automatically available to all the derived classes. This can help reduce the amount of code that needs to be maintained and tested, and can also help improve the consistency and correctness of the code.

Polymorphism: This allows developers to create multiple implementations of the same interface or method, and to choose the appropriate implementation at runtime based on the type of object being used. This can help make software systems more flexible and extensible. For example, if a new type of object is added to the system, it can be integrated seamlessly with existing code as long as it conforms to the same interface or method. This can help simplify maintenance tasks because changes can be made without affecting other parts of the program that rely on the interface or method.

UNIT 2(THEORY SOLUTION)

1. **What sorts of shortcomings of structure are addressed by classes? Explain giving appropriate example.[PU:2014 fall]**

   The shortcoming of structure that are addressed by classes are as follows:

   1) Class can create a subclass that will inherit parent's properties and methods, whereas Structure does not support the inheritance.
   2) Structure members can be easily accessed by the structure variables in their scope but class members doesn't due to feature of data hiding.
   3) Classes support polymorphism (late binding), whereas structure doesn't.
   4) Also we can use "this" pointers in classes due to which you don't have to explicitly pass an object to a member function.
   5) The structure data type cannot be treated like built in types while performing arithmetic operations. But it is valid in class using the concept of operator overloading.
   For example:

   struct complex
   {
   int real;
   int imag;
   }
   struct complex c1,c2,c3;
   c3=c1+c2; Is illegal .

2. **Differentiate between structure and class. Why Class is preferred over structure? Support your answer with suitable examples.***[PU:2016 fall]*

Structure	Class
1. A structure is a collection of variables of different data types under a single unit.	1. A class is a user-defined blueprint or prototype from which objects are created.
2. To define structure we will use "struct" keyword.	2. To define class we will use "class" keyword.
3. A structure has all members public by default.	3. A class has all members private by default.

4. A Structure does not support inheritance.	4. A Class can inherit from another class. Which means class supports inheritance.
5. Structures are good for small and isolated model objects.	5. Classes are suitable for larger or complex objects.
6. Structure is a value type and its object is created on the stack memory.	6. Class is a reference type and its object is created on the heap memory.
7. Function member of the struct cannot be virtual or abstract	7. Function member of the class can be virtual or abstract.

Due to the following reasons class is preferred over structure.

• Structures in C++ doesn't provide data hiding where as a class provides data hiding
• A Structure is not secure and cannot hide its implementation details from the end user while a class is secure and can hide its programming and designing details.
• Classes support polymorphism (late binding), whereas structure doesn't.
• Class support inheritance but structure doesn't.
• "this" pointer will works only with class.
• Class lets us to use constructors and destructors.

3. Where do you use friend function? What are the merits and di-merits of friend function?

In some situation non-member function need to access the private data of class or one class wants to access private data of second class and second wants to access private data of first class. This can achieve 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.

For example when we want to compare two private data members of two different classes in that case you need a common function which can make use of both the private variables of different class. In that case we create a normal function and make friend in both the classes, as to provide access of theirs private variables.

Merits of friend function:

• It can access the private data member of class from outside the class
• Allows sharing private class information by a non-member function.
• It acts as the bridge between two classes by operating on their private data's.

• It is able to access members without need of inheriting the class.
• It provides functions that need data which isn't normally used by the class.

Demerits of friend function:
• It violates the law of data hiding by allowing access to private members of the class from outside the class.
• Breach of data integrity
• Conceptually messy
• Runtime polymorphism in the member cannot be done.

4. Does friend function violate the data hiding? Explain Briefly.[PU:2017 fall]
The concept of data hiding indicates that nonmember functions should not be able to access the private data of class. But, it is possible with the help of a "friend function". That means, a function has to be declared as friend function to give it the authority to access to the private data.
Hence, friend function is not a member function of the class but can access private members of that class. So that's why friend function violate data hiding.

Let's illustrate the given concept with the following example.
```
#include<iostream>
using namespace std;
class sample
{
private:
int a,b;
public:
void setdata(int x,int y)
{
a=x;
b=y;
}
friend void sum(sample s);
};
void sum(sample s)
{
cout<<"sum="<<(s.a+s.b)<<endl;
}
int main()
{
sample s;
s.setdata(5,10);
sum(s);
return 0;
}
```

Here in class named sample there are two private data members a and b. The function sum() is not a member function but when it is declared as friend in class sample then sum() can access the private data a and b. which shows that friend function violate the concept of data hiding.

5. What are the limitations of static member functions

Limitations of static member functions:

- It doesn't have a "this" pointer.
- A static member function cannot directly refer to static members.
- Same function can't have both static and non static types.
- It can't be virtual.
- A static member function can't be declared as const or volatile.

6. Data hiding Vs Encapsulation

Data hiding	Encapsulation
1) Data Hiding means protecting the members of a class from an illegal or unauthorized access.	1) Encapsulation means wrapping the implementation of data member and methods inside a class.
2) Data hiding focus more on data security.	2) Encapsulation focuses more on hiding the complexity of the system.
3) The data under data hiding is always private and inaccessible.	3) The data under encapsulation may be private or public.
4) When data member of class are private only member function of that class can access it.	4) When implementation of all the data member and methods inside a class are encapsulated, the method name can only describe what action it can perform on an object of that class.
5) Data hiding is a process as well as technique.	5) Encapsulation is a sub-process in data hiding.

7. Abstraction Vs Data hiding

Abstraction	Data hiding
1) Abstraction is a mechanism of expressing the necessary properties by hiding the details.	1) Data Hiding means protecting the members of a class from an illegal or unauthorized access
2) It's purpose is to hide complexity.	2) It's purpose is to achieve encapsulation.
3) Class uses the abstraction to derive a new user-defined datatype	3) Data hiding is used in a class to make its data private.
4) It focuses on observable behavior of data.	4) It focuses on data security.

8. What is the difference between message passing and function call? Explain the basic message formalization. [PU:2006 spring]

| Message Passing	Function Call
1) In a message passing there is a designated receiver for that message; the receiver is some object to which message is sent.	**1)** In function call, there is no designated receiver.
2) Interpretation of the message (that is method is used to respond the message) is determined by the receiver and can vary with different receivers.	**2)** Determination of which method to invoke is very early binding of a name to fragment in procedure calls
3) Message passing must involve name of the object, function name and information to be sent.	3) Simply function name and its arguments is used to made function call.
4) A message is always given to some object, called the receiver.	4) Name will be matched to a message to determine when the method should be executed Signature the combination of return type and argument types.
5) Example: st.getdata(2,5)	5) **Example:** getdata(2,5);

Message passing formalization

Message passing means the dynamic process of asking an object to perform a specific action.

  ✓ A message is always given to some object, called the receiver.
  ✓ The action performed in response to the message is not fixed but may be differ, depending on the class of the receiver. That is different objects may accept the same message and yet perform different actions.

There are three identifiable parts to any message-passing expression. These are

1) **Receiver:** the object to which the message is being sent
2) **Message selector:** the text that indicates the particular message is being sent.
3) **Arguments** used in responding to the message.

s.getdata(100,200);

receiver   Message   arguments
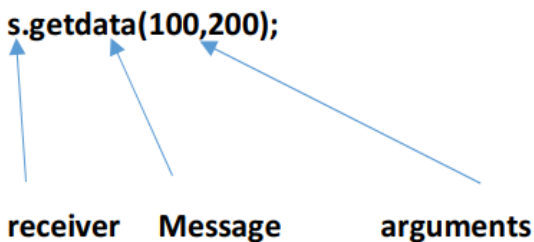
Example of message passing

```cpp
#include<iostream>
using namespace std;
class student
{
int roll;
public:
void getdata(int x)
{
roll=x;
}
void display()
{
cout<<"Roll number="<<roll;
}
};
int main()
{
student s;
s.getdata(325); //objects passing message
s.display(); //objects passing message
return 0;
}
```

9. What are the possible memory errors in programming. [PU:2014 spring]

Memory errors occur very commonly in programming, and they can affect application stability and correctness. These errors are due to programming bugs. They can be hard to reproduce, hard to debug, and potentially expensive to correct as well. Applications that have memory errors can experience major problems.

Memory errors can be broadly classified into Heap Memory Errors and Stack Memory Errors.

Some of the challenging memory errors are:

1. **Invalid Memory Access in stack and heap**: This error occurs when a read or write instruction references unallocated or deallocated memory.

2**. Memory leaks**

Memory leaks occur when memory is allocated but not released.

3. **Mismatched Allocation/Deallocation**

This error occurs when a deallocation is attempted with a function that is not the logical counterpart of the allocation function used.

To avoid mismatched allocation/deallocation, ensure that the right de-allocator is called. In C++, new[] is used for memory allocation and delete[] for freeing up.

4. **Missing allocation**

This error occurs when freeing memory which has already been freed. This is also called "repeated free" or "double free".

5. **Uninitialized Memory Access**

This type of memory error will occur when an uninitialized variable is read in your application. To avoid this type of memory error, always initialize variables before using them.

10. Is it mandatory to use constructor in class. Explain?

Constructor is a 'special' member function whose task is to initialize the object of its class.

It is not mandatory to use constructor in a class. As we know, Constructor are invoked automatically when the objects are created.

So that object are initialized at the time of declaration. There is no need to separate function call to initialize the object. Which reduces coding redundancy and minimizes the programming error such as uninitialized memory access.

11. **Differentiate between constructor and destructor.**

Constructor	Destructor
1) Constructor is used to initialize the instance of the class.	1) Destructor destroys the objects when they are no longer needed.
2) Constructors is called when new instances of class is created.	2) Destructor is called when instances of class is deleted or released.
3) Constructor allocates the memory.	3) Destructor releases the memory.
4) Constructor can have arguments.	4) Destructor cannot have any arguments.
5) Overloading of constructor is possible.	5) Overloading of Destructor is not possible.
6) Constructor have same name as class name.	6) Destructor have same name as class name but with **tlide (~)** sign.

7) Syntax:
Class_name(arguments)
{
//Body of the constructor
}
	7) Syntax:
	~Class_name()
	{
	//Body of the destructor
	}

12. **Discuss the various situations when a copy constructor is automatically invoked. How a default constructor can be equivalent to constructor having default arguments.**

Various situations when the copy constructor is called or automatically invoked are:
- When compiler generates the temporary object.
- When an object is constructed based on the object of the same class.
- When an object of the class is passed to function by values as an argument.
- When an object of the class is returned by value.

Constructor with default arguments is a is parameterized constructor which has default values in arguments. Thus the arguments defined in such constructor are optional. We may or may not pass arguments while defining object of the class. When an object is created with no supplied values, the default values are used. By this way, constructor with default constructor is equivalent to default constructor.

Example:
```cpp
#include<iostream>
using namespace std;
class Box
{
private:
float l,b,h;
public:
Box(float le=10,float br=5,float he=5)
{
l=le;
b=br;
h=he;
}
void displaymember()
{
cout<<"Length,breadth and height"<<l<<b<<h<<endl;
}
float getvolume()
{
return (l*b*h);
}
};
int main()
{
Box b;
float vol;
vol=b.getvolume();
cout<<"Volume="<<vol<<endl;
return 0;
}
```

13. What is de-constructor? can you have two destructors in a class? Give example to support your reason.[PU:2014 spring]

De-constructor is a member function that destroys the object that have been
created by a constructor.
Destructor doesn't take any arguments, which means destructor cannot be
overloaded. So, that there will be only one destructor in a class.
Destructors are usually used to deallocate memory and do other cleanup for a class
object and its class members when the object is destroyed. A destructor is called for
a class object when that object passes out of scope or is explicitly deleted.
So, that there cannot be more than one destructor (two destructor) in a same class.
(Write example of destructor yourself)

14. Differentiate methods of argument passing in constructor and destructor.

A constructor is allowed to accept the arguments as the arguments can be used to initialize the data members of the class.
A destructor does not accept any arguments as its only work is to deallocate the memory of the object.
In class, there can be multiple constructors which are identified by the number arguments passed.
In class, there is only one destructor.
Constructors can be overload to perform different action under the name of the same constructor whereas, destructors cannot be overloaded.

Example:

```cpp
#include <iostream>
using namespace std;
class test
{
private:
int a;
public:
test()
{
a=10;
cout<<"Default constructor is called"<<endl;
}
test(int x)
{
a=x;
cout<<"Parameterized constructor is called"<<endl;
}
~test()
{
cout<<"Destructor is called"<<endl;
}
void display()
{
cout<<"value of a="<<a<<endl;
}
};
```

```
int main()
{
test t1;
test t2(5);
t1.display();
t2.display();
return 0;
}
```

15. What do you mean by stack vs Heap? Explain memory recovery. Explain the use of new and delete operator. [PU:2009 spring]

Stack

- It's a region of computer's memory that stores temporary variables created by each function (including the main() function).
- The stack is a "Last in First Out" data structure and limited in size. Every time a function declares a new variable, it is "pushed" (inserted) onto the stack.
- Every time a function exits, all of the variables pushed onto the stack by that function, are freed or popped (that is to say, they are deleted).
- Once a stack variable is freed, that region of memory becomes available for other stack variables.

Heap

- The heap is a region of computer's memory that is not managed automatically and is not as tightly managed by the CPU.
- We must allocate and de-allocate variable explicitly. (for allocating variable new and for freeing variables delete operator is used.
- It does not have a specific limit on memory size.

Memory recovery

Because in most languages objects are dynamically allocated, they must be recovered at run-time. There are two broad approaches to this:

- Force the programmer to explicitly say when a value is no longer being used: delete aCard; // C++ example
- Use a garbage collection system that will automatically determine when values are no longer being used, and recover the memory.

Use of new and delete operator

new is used to allocate memory blocks at run time (dynamically). While, delete is used to de-allocate the memory which has been allocated dynamically.

Example of new and delete in C+
```
#include <iostream>
using namespace std;
int main ()
{
int i,n;
int *ptr;
int sum=0;
cout << "How many numbers would you like to Enter? ";
cin >>n;
ptr= new int[n];
for (i=0; i<n; i++)
{
cout << "Enter number:"<<i+1<<endl;
cin >> ptr[i];
}
for (i=0; i<n; i++)
{
sum=sum+ptr[i];
}
cout<<"sum of numbers="<<sum<<endl;
delete[] ptr;
return 0;
}
```

16. What are the advantages of dynamic memory allocation? Explain with suitable example. [PU:2016 spring]
The main advantage of using dynamic memory allocation is preventing the wastage of memory or efficient utilization of memory. Let us consider an example:
In static memory allocation the memory is allocated before the execution of program begins (During compilation).In this type of allocation the memory cannot be resized after initial allocation. So it has some limitations. Like
- Wastage of memory
- Overflow of memory

eg. int num[100];

Here, the size of an array has been fixed to 100.If we just enter to 10 elements only, then their will be wastage of 90 memory location and if we need to store more than 100 elements there will be memory overflow.

But, in dynamic memory allocation memory is allocated during runtime.so it helps us to allocate and de-allocate memory during the period of program execution as per requirement. So there will be proper utilization of memory.

In C++ dynamic memory allocation can be achieved by using new and delete operator. **new** is used to allocate memory blocks at run time (dynamically). While, **delete** is used to de-allocate the memory which has been allocated dynamically.

Let us consider an example:

```
#include <iostream>
using namespace std;
int main ()
{
int i,n;
int *ptr;
int sum=0;
cout << "How many numbers would you like to Enter? ";
cin >>n;
ptr= new int[n];
for (i=0; i<n; i++)
{
cout << "Enter number:"<<i+1<<endl;
cin >> ptr[i];
}
for (i=0; i<n; i++)
{
sum=sum+ptr[i];
}
cout<<"sum of numbers="<<sum<<endl;
delete[] ptr;
return 0;
}
```

Here, in above example performs the sum of n numbers .But the value of n will be provided during runtime.so that memory allocation to store n numbers is done during program execution which results proper utilization of memory and preventing from wastage of memory or overflow of memory.

17.Constructor is called in derived class but it can't be inherited. Support your answer with suitable example.

In object-oriented programming, a constructor is a special method that is called when an object is created. The constructor initializes the object's data members and sets the object's initial state. In C++, constructors are not inherited by derived classes. However, the derived class can call the constructor of the base class to initialize the base class members.

Here is an example that demonstrates this concept:

```cpp
#include <iostream>
using namespace std;
class Base
{
private:
int a;
public:
   Base(int x)
   {
                a=x;
     cout << "Base constructor called "<<endl;
   }
   void showbase()
   {
       cout<<"value initialzied in base constructor="<<a<<endl;
       }
};

class Derived : public Base
{
private:
int b;
public:
   Derived(int x,int y) : Base(x)
       {
     b=y;
     cout << "Derived constructor called"<<endl;
   }
   void showderived()
   {
       cout<<"value initialized in derived constructor="<<b<<endl;
       }
};
```

```
int main()
 {
    Derived d(5,10);
    d.showbase();
    d.showderived();
    return 0;
}
```
Output:

Base constructor called
Derived constructor called
value initialized in base constructor=5
value initialized in derived constructor=10

In this example, we have a base class 'Base' and a derived class 'Derived'. The Base class has a constructor that takes one integer parameter, and the Derived class has a constructor that takes two integer parameter and calls the Base constructor by passing one parameter.

When we create an object of the Derived class in the main function and pass the integer value of 5 and 10 to its constructor i.e. Derived d(5,10).

 Now, derived class called the base class constructor by supplying first value .i.e 5 is initialized to base class data member 'a' ,followed by the constructor of the Derived class constructor executed and initialized the derived class data member with value 10.

Thus, even though the Derived class does not inherit the constructor of the Base class, it can still call the constructor of the Base class explicitly in its own constructor to initialize the base class members.

UNIT-3 (THEORY SOLUTION)

1. **When base class and derived class have the same function name what happens when derived class object calls the function?[PU 2017 fall]**

When the base class and derived class have the same function name, the derived class function overrides the function that is inherited from base class, which is known as function overriding. Now, when derived class object calls that overridden function, the derived class member function is accessed.

For example:
```cpp
#include<iostream>
using namespace std;
class Base
{
public:
void display()
{
cout<<"This is base class"<<endl;
}
};
class Derived:public Base
{
public:
void display()
{
cout<<"This is derived class"<<endl;
}
};
int main()
{
Derived d;
d.display();
return 0;
}
```
Output:

This is derived class

In this example, the class derived inherits the public member function display() .When we redefine this function in derived class then the inherited members from base are overridden.so when derived class object calls the function display() it actually refers the function in derived class but not the inherited member function in base class.

2. **How inheritance support reusability features of OOP? Explain with example.**
   [PU:2010 spring]

   Using the concept of inheritance the data member and member function of classes can be reused. When once a class has been written and tested, it can be adapted by another programmer to suit their requirements. This is basically done by creating new classes, reusing the properties of the existing ones. This mechanism of deriving a new class from an old one is called inheritance.

   A derived class includes all features of the base class and then it can add additional features specific to derived class.

   Example:

```cpp
#include<iostream>
using namespace std;
class Sum
{
protected:
int a,b;
public:
void getdata()
{
cout<<"Enter two numbers"<<endl;
cin>>a>>b;
}
void display()
{
cout<<"a="<<a<<endl;
cout<<"b="<<b<<endl;
}
};
class Result:public Sum
{
private:
int total;
public:
void disp_result()
{
total=a+b;
cout<<"sum="<<total<<endl;
}
};
```

```
int main()
{
Result r;
r.getdata();
r.display();
r.disp_result();
return 0;
}
```
Here, In above example **Sum** is base class and **Result** is derived class. The data member **named a and b** and member function display () of base class are inherited and they are used by using the object of derived class named **Result.** Hence, we can see that inheritance provides reusability.

3.How are arguments are sent to base constructors in multiple inheritance? Who is responsible for it.[PU:2013 spring]

In Multiple Inheritance arguments are sent to base class in the order in which they appear in the declaration of the derived class**.**
For example:
Class gamma: public beta, public alpha
{

}
Order of execution
beta(); base constructor (first)
alpha(); base constructor(second)
gamma();derived (last)
The constructor of derived class is responsible to supply values to the base class constructor.

Program:


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

class alpha
{
int x;
public:
alpha(int a)
{
x=a;
cout<<"Alpha is initialized"<<endl;
}
```

```cpp
void showa()
{
cout<<"x="<<x<<endl;
}
};
class beta
{
int y;
public:
beta(int b)
{
y=b;
cout<<"Beta is initialized"<<endl;
}
void showb()
{
cout<<"y="<<y<<endl;
}
};
class gamma:public beta,public alpha
{
int z;
public:
gamma(int a,int b,int c):alpha(a),beta(b)
{
z=c;
cout<<"Gamma is initialized"<<endl;
}
void showg()
{
cout<<"z="<<z<<endl;
}
};

int main()
{
gamma g(5,10,15);
g.showa();
g.showb();
g.showg();
return 0;
}
```

Output:

Beta is initialized
Alpha is initialized
Gamma is initialized
x=5
y=10
z=15

Here, **beta** is initialized first although it appears second in the derived constructor *as it has been* declared first in the derived class header line
class gamma: public beta, public alpha
{ }
If we change the order *to*
class gamma: public alpha, public beta
{ }
then alpha will be initialized first

4.Class 'Y' has been derived from class 'X' .The class 'Y' does not contain any data members of its own. Does the class 'Y' require constructors? If yes why.[PU:2013 spring]

When Class 'Y' has been derived from class 'X' and class 'Y' does not contain any data members of its own then,

• It is mandatory have constructor in derived class 'Y', whether it has data members to be initialized or not, if there is constructor with arguments(parameterized constructor) in base class 'X'.

• It not necessary to have constructor in derived class 'Y' if base class 'X' does not contain a constructor with arguments (parameterized constructor).

Derived class constructor is used to provide the values to the base class constructor.

Example
```
#include<iostream>
using namespace std;
class X
{
private:
int a;
public:
X (int m)
{
a=m;
}
```

```cpp
void display()
{
cout<<"a="<<a<<endl;
}
};
class Y:public X
{
public:
Y(int b):X(b)
{ }
};
int main()
{
Y obj(5);
obj.display();
return 0;
}
```
Here, in above example, class 'Y' does not have any data members but there is a parameterized constructor in class 'X' so, there is necessary to have constructor in class 'Y'.

5. **Write a short notes on:**

Generalization[PU:2013 spring]
Generalization is the process of taking out common properties and functionalities from two or more classes and combining them together into another class which acts as the parent class of those classes or what we may say the generalized class of those specialized classes. All the subclasses are a type of superclass. So we can say that subclass "is-A" superclass. Therefore Generalization is termed as "is-A relationship".
Here is an example of generalization:
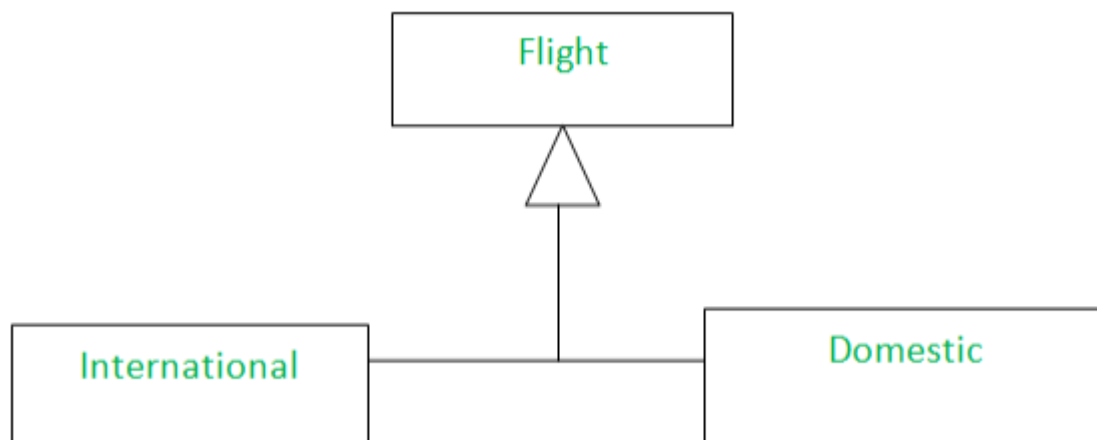
In this figure, we see that there are two types of flights so we made a flight class which will contain common properties and then we has an international and domestic class which are an extension of flight class and will have properties of flight as well as their own. Here flight is the parent/superclass and the other two are child/subclass. International Flight "is-a" flight as well as the domestic flight.

6. **Explain how composition provide reusability.**

In composition one class contains object of another class as its member data, which means members of one class is available in another class. Composition exhibits "has-a" relationship. For example, Company **has an** employee.
So, properties of one class can be used by another class independently.

Let us consider an example
```cpp
#include<iostream>
using namespace std;
class Employee
{
int eid;
float salary;
public:
void getdata()
{
cout<<"Enter id and salary of employee"<<endl;
cin>>eid>>salary;
}
void display()
{
cout<<"Emp ID:"<<eid<<endl;
cout<<"Salary:"<<salary<<endl;
}
};
class Company
{
char cname[20];
char department[20];
Employee e;
public:
void getdata()
{
e.getdata();
cout<<"Enter company name and Deparment:"<<endl;
cin>>cname>>department;
}
```

```
void display()
{
e.display();
cout<<"Company name:"<<cname<<endl;
cout<<"Department:"<<department<<endl;
}
};

int main()
{
Company c;
c.getdata();
c.display();
return 0;
}
```
In above example class company contains the object of another class employee. As we know "company **has a** employee" sounds logical .so there exits has a relationship between company and employee. Class company contains the object of class employee and members of class employee are used in class company which provides reusability.

UNIT-4 (THEORY SOLUTION)

1) How can polymorphism be achieved during compile time and during runtime? Explain with examples in C++.

Compile time polymorphism can be achieved in two ways:

- Function overloading
- Operator Overloading

Runtime polymorphism can be achieved by using virtual function.

Now, let us discuss them briefly.

Compile time polymorphism

Compile time polymorphism means that an object is bound to its function call at the compile time. That means, there is no ambiguity at the compile time about which a function is linked to a particular function call. This mechanism is called early binding or static binding or static linking.

Function overloading

- The method of using same function name but with different parameter list along with different data type to perform the variety of different tasks is known as function overloading.
- The correct function is to be invoked is determined by checking the number and type of arguments but not function return type.
  ```
  #include<iostream>
  using namespace std;
  class calcarea
  {
  public:
  int area(int s)
  {
  return (s*s);
  }
  int area(int l,int b)
  {
  return(l*b);
  }
  ```

```cpp
float area(float r)
{
return(3.14*r*r);
}
};
int main()
{
calcarea c1;
cout<<"Area of square="<<c1.area(5)<<endl;
cout<<"Area of Rectangle="<<c1.area(5,10)<<endl;
cout<<"Area of Circle="<<c1.area(2.5f)<<endl;
return 0;
}
```

Operator overloading

```cpp
#include<iostream>
using namespace std;
class space
{
private:
int x,y,z;
public:
space(int a,int b,int c)
{
x=a;
y=b;
z=c;
}
void display()
{
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
cout<<"z="<<z<<endl;
}
void operator -()
{
x=-x;
y=-y;
z=-z;
}
}
```

```cpp
int main()
{
space s(5,10,15);
cout<<"s:"<<endl;
s.display();
-s;
cout<<"-s:"<<endl;
s.display();
return 0;
}
```

Runtime polymorphism

We should use virtual functions and pointers to objects to achieve run time
polymorphism. For this, we use functions having same name, same number of
parameters, and similar type of parameters in both base and derived classes. The
function in the base class is declared as virtual using the keyword virtual.
A virtual function uses a single pointer to base class pointer to refer to all the derived objects.
When a function in the base class is made virtual, C++ determines which function to use at run
time based on the type of object pointed by the base class pointer, rather than the type
of the pointer.

```cpp
#include<iostream>
using namespace std;
class base
{
public:
virtual void show()
{
cout<<"Base Class Show function"<<endl;
}
};
class derived:public base
{
public:
void show()
{
cout<<"Derived Class Show function "<<endl;
}
};
```

```
int main()
{
base b1,*bptr;
derived d1;
bptr=&b1;
bptr->show();
bptr=&d1;
bptr->show();
return 0;
}
```
Output:
```
Base Class Show function
Derived Class Show function
```

2) **Explain importance of polymorphism with a suitable example.**

The word polymorphism means having many forms. Real life example of polymorphism, a person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations. This is called polymorphism.

Polymorphism (function overloading) allows one to write more than one method with the same name but with different number of arguments. Hence same method name can be used to perform operations on different data types.

Suppose we want to calculate the area of square, rectangle and circle. We can use the same function named area for all of these by just changing the number and type of arguments.

```
#include<iostream>
using namespace std;
class calcarea
{
public:
int area(int s)
{
return (s*s);
}
int area(int l,int b)
{
return(l*b);
}

float area(float r)
{
return(3.14*r*r);
}
};
```

```
int main()
{
calcarea c1;
cout<<"Area of square="<<c1.area(5)<<endl;
cout<<"Area of Rectangle="<<c1.area(5,10)<<endl;
cout<<"Area of Circle="<<c1.area(2.5f)<<endl;
return 0;
}
```

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function

```
#include<iostream>
using namespace std;
class shape
{
public:
virtual void draw()=0;
};
class square:public shape
{
public:
void draw()
{
cout<<"Implementing method to draw square"<<endl;
}
};
class circle:public shape
{
public:
void draw()
{
cout<<"Implementing method to draw circle"<<endl;
}
};
```

```
int main()
{
shape *bptr;
square s;
circle c;
bptr=&s;
bptr->draw();
bptr=&c;
bptr->draw();
return 0;
}
```

Similarly, let us consider an example, operator symbol '+' is used for arithmetic operation between two numbers, however by overloading (means given additional job) it can be used over Complex Object like currency that has Rs and Paisa as its attributes, complex number that has real part and imaginary part as attributes. By overloading same operator '+' can be used for different purpose like concatenation of strings. When same function name is used in defining different function to operate on different data (type or number of data) then this feature of polymorphism is function overloading.

```
#include<iostream>
using namespace std;
class complex
{
private:
int real,imag;
public:
complex()
{ }
complex(int r,int i)
{
real=r;
imag=i;
}
void display()
{
cout<<real<<"+i"<<imag<<endl;
}
complex operator +(complex);
};
```

```cpp
complex complex::operator +(complex c2)
{
complex temp;
temp.real=real+c2.real;
temp.imag=imag+c2.imag;
return temp;
}

int main()
{
complex c1(2,3);
complex c2(2,4);
complex c3;
c3=c1+c2;
cout<<"c1=";
c1.display();
cout<<"c2=";
c2.display();
cout<<"c3=";
c3.display();
return 0;
}
```

3) **How do we make use of a virtual destructor when we need to make sure that the different destructors in inheritance chain are called in order? Explain with an example in C++.**

Let's first see what happens when we do not have a virtual Base class destructor.

```cpp
class Base
{
public:
~Base()
{
cout << "Base class destructor"<<endl;
}
};
class Derived: public Base
{
public:
~Derived()
{
cout<< "Derived class destructor"<<endl;
}
};
```

```
int main()
{
Base* ptr = new Derived; //Base class pointer points to derived class object
delete ptr;
}
```
Output:
Base class Destructor

In the above example, delete ptr will only call the Base class destructor, which is undesirable because, then the object of Derived class remains un-destructed, because its destructor is never called. Which leads to memory leak situation.

To make sure that the derived class destructor is mandatory called, we make base class destructor as virtual
```
class Base
{
public:
virtual ~Base()
{
cout << "Base class destructor"<<endl;
}
};
```
Output:
Derived class Destructor
Base class Destructor

When we have Virtual destructor inside the base class, then first Derived class's destructor is called and then Base class's destructor is called, which is the desired behavior.

4. Virtual function vs Pure virtual function [PU:2016 spring]

Virtual function	Pure virtual function
1. A virtual function is a member function of base class which can be redefined by derived class.	1. A pure virtual function is a member function of base class whose only declaration is provided in base class and should be defined in derived class otherwise derived class also becomes abstract.
2. Classes having virtual functions are not abstract.	2. Base class containing pure virtual function becomes abstract.

3. Virtual return_type   function_name(arglist)
 	
{
 	
    // code
 	
}
	3. Virtual return_type function_name(arglist)=0;

4. Base class having virtual function can be instantiated i.e. its object can be made.	4. Base class having pure virtual function becomes abstract i.e. it cannot be instantiated
5. All derived class may or may not redefine virtual function of base class.	5. All derived class must redefine pure virtual function of base class otherwise derived class also becomes abstract just like base class.

5.Function Overloading Vs Function overriding

Function Overloading	Function Overriding
1. It allows us to declare two or more function having same name with different number of parameters or different datatypes of argument.	1. It allow us to declare a function in parent class and child class with same name and signature.
2. Overloading is utilized to have the same number of various functions which act distinctively relying	2. Overriding is required when a determined class function needs to perform some additional or unexpected job in comparison to the base class function.
3. It can occur without inheritance concept.	3. Overriding can only be done when one class is inherited by another classes.
4. It is an example of compile time polymorphism.	4. It is an example of runtime polymorphism.
5. The overloaded functions are always in the same scope.	5. Functions are always in the different scope.
6. we can have any number of overloaded functions	6. We can have only one overriding function in the child class.

6.Friend function vs Virtual function

Friend function	Virtual Function
1. It is non-member functions that usually have private access to class representation.	1. It is a base class function that can be overridden by a derived class.
2. It is used to access private and protected classes.	2. It is used to ensure that the correct function is called for an object no matter what expression is used to make a function class.
3. It is declared outside the class scope. It is declared using the 'friend' keyword.	3. It is declared within the base class and is usually redefined by a derived class. It is declared using a 'virtual' keyword.
4. It is generally used to give non-member function access to hidden members of a class.	4. It is generally required to tell the compiler to execute dynamic linkage of late binding on function.
5. They support sharing information of class that was previously hidden, provides method of escaping data hiding restrictions of C++, can access members without inheriting class, etc.	5. They support object-oriented programming, ensures that function is overridden, can be friend of other function, etc.
6. It can access private members of the class even while not being a member of that class.	6. It is used so that polymorphism can work.

7.Operator overloading [PU: 2016 fall]

The mechanism of adding special meaning to an operator is called operator overloading.
It provides a flexibility for the creation of new definitions for most C++ operators. Using operator overloading we can give additional meaning to normal C++ operations such as (+,-,=,<=,+= etc.) when they are applied to user defined data types.

let us consider an example, operator symbol '+' is used for arithmetic operation between two numbers, however by overloading (means given additional job) it can be used over Complex Object like currency that has Rs and Paisa as its attributes, complex number that has real part and imaginary part as attributes.

By overloading same operator '+' can be used for different purpose like concatenation of strings, Addition of complex numbers, Addition of two vectors , Addition of two times etc.

General form:

```
return_type operator op(arglist)
{
function body //task defined
}
```

Where,

- return_type is the type of value returned by the specified operation.
- op is the operator being overloaded.
- Operator op is function name, where operator is keyword.

8.Why does this pointer is widely used than object pointer? Write a program to implement pure polymorphism.

In C++, the this pointer is a special pointer that points to the object that the member function belongs to. It is used to access the member variables and methods of the current object.

The this pointer is widely used in C++ for several reasons:

i)This pointer can be used to refer current class instance variable

When a member variable or method has the same name as a local variable or parameter, the this pointer can be used to disambiguate between them. For example, consider the following code:

```
class MyClass {
   private:
      int x;

   public:
      void setX(int x) {
            this->x = x;
      }
};
```

In this code, the setX method takes a parameter x that has the same name as the member variable x. The this pointer is used to refer to the member variable x, so that it can be assigned the value of the parameter.

ii)this pointer return the objects it points to.

For example, the statement
return *this;
inside a member function will return the object that invoked the function.

Example:
```cpp
#include<iostream>
#include<string.h>
using namespace std;

class person
{
char name[20];
float age;
public:
person()
{ }
person (char n[],float a)
{
strcpy(name,n);
age=a;
}
person& greater(person &x)
{
if(x.age>=age)
{
return x;
}
else
{
return *this;
}
}
void display()
{
cout<<"Name:"<<name<<endl;
cout<<"Age:"<<age<<endl;
}
};
```

```cpp
int main()
{
person p1("Ram",52);
person p2("Hari",24);
person p3;
p3=p1.greater(p2);
cout<<"Elder person is:"<<endl;
p3.display();
return 0;
}
```

UNIT 2(Program Solution)

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

```cpp
#include<iostream>
using namespace std;
struct car
{
char name[50];
char model[50];
long int price;
int miles;
}
  int main()
    {
        car c;
        cout<<"Enter manufacturer of car"<<endl;
        cin.getline(c.name,50);
        cout<<"Enter the model name of car"<<endl;
        cin.getline(c.model,50);
        cout<<"Enter the price of car"<<endl;
        cin>>c.price;
        cout<<"Enter number of miles on odometer"<<endl;
        cin>>c.miles;
        cout<<"Details of car:"<<endl;
        cout<<"Manufacturer name:"<<c.name<<endl;
        cout<<"Model name:"<<c.model<<endl;
        cout<<"Price :"<<c.price<<endl;
        cout<<"Number of miles on odometer:"<<c.miles<<endl;
        return 0;
    }
```

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]**

```cpp
#include<iostream>
using namespace std;
class Employee
{
private:
int empno;
char name[20];
char address[20];
public:
void readdata();
void displaydata();
};
void Employee::readdata()
{
cout<<"Enter the Employee number"<<endl;
cin>>empno;
cout<<"Enter the Employee name"<<endl;
cin>>name;
cout<<"Enter the address"<<endl;
cin>>address;
}
void Employee::displaydata()
{
cout<<"Employee number="<<empno<<endl;
cout<<"Employee name="<<name<<endl;
cout<<"Employee address="<<address<<endl;
}
int main()
{
Employee e1,e2;
cout<<"Enter the information of first employee"<<endl;
e1.readdata();
cout<<"Enter the information of second employee"<<endl;
e2.readdata();
cout<<"Information of first employee is"<<endl;
e1.displaydata();
cout<<"Information of second employee is"<<endl;
e2.displaydata();
return 0;
}
```

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]**

```cpp
#include<iostream>
using namespace std;
class Student
{
private:
char stdnt_name[20];
char faculty[20];
int roll_no;
public:
void readdata();
void displaydata();
};
void Student::readdata()
{
cout<<"Enter student name"<<endl;
cin>>stdnt_name;
cout<<"Enter student faculty"<<endl;
cin>>faculty;
cout<<"Enter student roll number"<<endl;
cin>>roll_no;
}
void Student::displaydata()
{
cout<<"Name="<<stdnt_name<<endl;
cout<<"Roll no="<<roll_no<<endl;
cout<<"Faculty="<<faculty<<endl;
}
int main()
{
Student st1,st2;
cout<<"Enter the information of first student"<<endl;
st1.readdata();
cout<<"Enter the information of second student"<<endl;
st2.readdata();
cout<<"Informaton of first student is:"<<endl;
st1.displaydata();
cout<<"Information of second student is:"<<endl;
st2.displaydata();
return 0;
}
```

4. Modify the Que.no 3 **for 20 students using array of object.**

```cpp
#include<iostream>
using namespace std;
class Student
{
private:
char stdnt_name[20];
char faculty[20];
int roll_no;
public:
void readdata();
void displaydata();
};
void Student::readdata()
{
cout<<"Enter student name"<<endl;
cin>>stdnt_name;
cout<<"Enter student faculty"<<endl;
cin>>faculty;
cout<<"Enter student roll number"<<endl;
cin>>roll_no;
}
void Student::displaydata()
{
cout<<"Name="<<stdnt_name<<endl;
cout<<"Roll no="<<roll_no<<endl;
cout<<"Faculty="<<faculty<<endl;
}
int main()
{
Student st[20];
int i;
for(i=0;i<20;i++)
{
cout<<"Enter the information of student:"<<i+1<<endl;
st[i].readdata();
}
for(i=0;i<20;i++)
{
cout<<"Information of student:"<<i+1<<endl;
st[i].displaydata();
}
return 0;
}
```

5. WAP to perform the addition of time in hours, minutes and seconds by passing object as an argument.

```cpp
#include<iostream>
using namespace std;
class Time
{
private:
int hours,minutes,seconds;
public:
void gettime(int h,int m,int s)
{
hours=h;
minutes=m;
seconds=s;
}
void display()
{
cout<<hours<<":"<<minutes<<":"<<seconds<<endl;
}
void sum(Time t1,Time t2)
{
seconds=t1.seconds+t2.seconds;
minutes=seconds/60;
seconds=seconds%60;
minutes=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(2,45,35);
t2.gettime(3,30,40);
t3.sum(t1,t2);
cout<<"First time=";
t1.display();
cout<<"Second time=";
t2.display();
cout<<"sum of two times=";
t3.display();
return 0;
}
```

6. WAP to perform the addition of time by passing object as an argument and returning object.

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

class Time
{
private:
int hours,minutes,seconds;
public:
void gettime(int h,int m,int s)
{
hours=h;
minutes=m;
seconds=s;
}
void display()
{
cout<<hours<<":"<<minutes<<":"<<seconds<<endl;
}
Time sum(Time t1,Time t2)
{
Time temp;
temp.seconds=t1.seconds+t2.seconds;
temp.minutes=temp.seconds/60;
temp.seconds=temp.seconds%60;
temp.minutes=temp.minutes+t1.minutes+t2.minutes;
temp.hours=temp.minutes/60;
temp.minutes=temp.minutes%60;
temp.hours=temp.hours+t1.hours+t2.hours;
return temp;
}
};
int main()
{
Time t1,t2,t3,result;
t1.gettime(2,45,35);
t2.gettime(3,30,40);
result=t3.sum(t1,t2);
cout<<"First time=";
t1.display();
cout<<"Second time=";
t2.display();
cout<<"Sum of two times=";
result.display();
return 0;
}
```

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);*

```cpp
#include<iostream>
using namespace std;
class Time
{
private:
int hours,minutes,seconds;
public:
void gettime(int h,int m,int s)
{
hours=h;
minutes=m;
seconds=s;
}
void display()
{
cout<<hours<<":"<<minutes<<":"<<seconds<<endl;
}
Time sum(Time t2)
{
Time temp;
temp.seconds=seconds+t2.seconds;
temp.minutes=temp.seconds/60;
temp.seconds=temp.seconds%60;
temp.minutes=temp.minutes+minutes+t2.minutes;
temp.hours=temp.minutes/60;
temp.minutes=temp.minutes%60;
temp.hours=temp.hours+hours+t2.hours;
return temp;
}
};
int main()
{
Time t1,t2,t3;
t1.gettime(2,45,35);
t2.gettime(3,30,40);
t3=t1.sum(t2);
cout<<"First time=";
t1.display();
cout<<"Second time=";
t2.display();
cout<<"Sum of two times=";
t3.display();
return 0;
}
```

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);*

```cpp
#include<iostream>
using namespace std;
class Distance
{
private:
int feet;
int inches;
public:
void getdata();
Distance add(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 d2)
{
Distance sum;
sum.inches=inches+d2.inches;
sum.feet=sum.inches/12;
sum.inches=sum.inches%12;
sum.feet=sum.feet+feet+d2.feet;
return (sum);
}
int main()
{
Distance d1,d2,d3;
cout<<"Enter first Distance"<<endl;
d1.getdata();
cout<<"Enter second Distance"<<endl;
d2.getdata();
d3=d1.add(d2);
cout<<"The sum of    two distances=" ;
d3.display();
return 0;}
```

9. Create a class called Rational having data members nume and deno and using friend function find which one is greater.

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

class Rational
{
private:
int nume,deno;
public:
void getdata()
{
cout<<"Enter the value of numerator"<<endl;
cin>>nume;
cout<<"Enter the value of denominator"<<endl;
cin>>deno;
}
friend void max(Rational r);
};

void max(Rational r)
{
if(r.nume>r.deno)
{
cout<<"Maximum value="<<r.nume<<endl;
}
else
{
cout<<"Maximum value="<<r.deno<<endl;
}
}

int main()
{
Rational r;
r.getdata();
max(r);
return 0;
}
```

10. WAP to add the private data of three different classes using friend function.

```cpp
#include<iostream>
using namespace std;
class B;
class C;
class A
{
private:
int x;
public:
void setdata(int num)
{
x=num;
}
friend void sum(A,B,C);
};

class B
{
private:
int y;
public:
void setdata(int num)
{
y=num;
}
friend void sum(A,B,C);
};

class C
{
private:
int z;
public:
void setdata(int num)
{
z=num;
}
friend void sum(A,B,C);
};

void sum(A m,B n,C o)
{
cout<<"sum="<<(m.x+n.y+o.z)<<endl;
}
```

```cpp
int main()
{
A p;
B q;
C r;
p.setdata(5);
q.setdata(10);
r.setdata(15);
sum(p,q,r);
return 0;
}
```

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**]

```cpp
#include<iostream>
using namespace std;
class Y;
class Z;

class X
{
private:
int a;
public:
void getdata()
{
cout<<"Enter first number"<<endl;
cin>>a;
}
friend void max(X,Y,Z);
};

class Y
{
private:
int b;
public:
void getdata()
{
cout<<"Enter second number"<<endl;
cin>>b;
}
friend void max(X,Y,Z);
};
```

```cpp
class Z
{
private:
int c;

public:

void getdata()
{
cout<<"Enter third number"<<endl;
cin>>c;
}
friend void max(X,Y,Z);
};




void max(X m,Y n,Z o)
{
int d;
cout<<"Enter fourth number"<<endl;
cin>>d;
if(m.a>n.b&&m.a>o.c&&m.a>d)
{
cout<<"Largest number="<<m.a<<endl;
}
else if(n.b>o.c&&n.b>d)
{
cout<<"Largest number="<<n.b<<endl;
}
else if(o.c>d)
{
cout<<"Largest number="<<o.c<<endl;
}
else
{
cout<<"Largest number="<<d<<endl;
}
}
```

```cpp
int main()
{
X p;
Y q;
Z r;
p.getdata();
q.getdata();
r.getdata();
max(p,q,r);
return 0;
}
```

12. WAP to swap the contents of two variables of 2 different classes using friend function.

```cpp
#include <iostream>
using namespace std;
class ABC;
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;
}
```

```cpp
void display()
{
cout<<"value2="<<y<<endl;
}
friend void swap(XYZ &,ABC &) ;
};

void swap (XYZ &m, ABC &n)
{
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;
}
```

13. WAP to add two complex numbers of two different classes using friend function.

```cpp
#include<iostream>
using namespace std;
class complex2;
class complex1
{
private:
int real,imag;
```

```cpp
public:
void getdata()
{
cout<<"Enter real and imaginary part"<<endl;
cin>>real>>imag;
}
void display()
{
cout<<real<<"+"<<imag<<"i"<<endl;
}
friend void addcomplex(complex1,complex2);
};

class complex2
{
private:
int real,imag;
public:
void getdata()
{
cout<<"Enter real and imaginary part"<<endl;
cin>>real>>imag;
}
void display()
{
cout<<real<<"+"<<imag<<"i"<<endl;
}
friend void addcomplex(complex1,complex2);
};

void addcomplex(complex1 c1,complex2 c2)
{
int real,imag;
real=c1.real+c2.real;
imag=c1.imag+c2.imag;
cout<<"sum of complex number="<<real<<"+"<<imag<<"i"<<endl;
}
```

```cpp
int main()
{
complex1 c1;
complex2 c2;
c1.getdata();
c2.getdata();
cout<<"first complex number"<<endl;
c1.display();
cout<<"second complex number"<<endl;
c2.display();
addcomplex(c1,c2);
return 0;
}
```

14. WAP to add complex numbers of two different classes using friend class.

```cpp
#include<iostream>
using namespace std;
class complex2;
class complex1
{
private:
int real,imag;
public:
void getdata()
{
cout<<"Enter real and imaginary part"<<endl;
cin>>real>>imag;
}
void display()
{
cout<<real<<"+i"<<imag<<endl;
}
friend class complex2;
};
class complex2
{
private:
int real,imag;
```

```cpp
public:
void getdata()
{
cout<<"Enter real and imaginary part"<<endl;
cin>>real>>imag;
}
void display()
{
cout<<real<<"+i"<<imag<<endl;
}
void addcomplex(complex1);
};
void complex2::addcomplex(complex1 c1)
{
real=c1.real+real;
imag=c1.imag+imag;
cout<<"sum of complex number="<<real<<"+i"<<imag<<endl;
}
int main()
{
complex1 c1;
complex2 c2;
c1.getdata();
c2.getdata();
cout<<"first complex number"<<endl;
c1.display();
cout<<"second complex number"<<endl;
c2.display();
cout<<"sum="<<endl;
c2.addcomplex(c1);
return 0;
}
```

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]**

```cpp
#include<iostream>
using namespace std;
class customer
{
private:
float principle,rate,si;
int time;
```

```cpp
public:
void setdata(float p,int t,float r=8);
void display();
};
void customer::setdata(float p,int t,float r)
{
principle=p;
time=t;
rate=r;
}
void customer::display()
{
si=(principle*time*rate)/100;
cout<<"Simple Interest="<<si<<endl;
}
int main()
{
float p;
int t,i;
customer c[3];
for(i=0;i<3;i++)
{
cout<<"Enter principle and time for customer"<<i+1<<endl;
cin>>p>>t;
c[i].setdata(p,t);
c[i].display();
}
return 0;
};
```

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]**

```cpp
#include<iostream>
#include<string.h>
using namespace std;
```

```cpp
class City
{
private:
char CityName[20];
float DistFromKtm;
public:
void setdata(char cname[],float d)
{
strcpy(CityName,cname);
DistFromKtm=d;
}
void display()
{
cout<<"City name="<<CityName<<endl;
cout<<"Distance from ktm="<<DistFromKtm<<endl;
}
float AddDistance(City c1,City c2)
{
return (c1.DistFromKtm+c2.DistFromKtm);
}
};
int main()
{
City c1,c2,c3;
c1.setdata("Pokhara",250);
c2.setdata("Dhangadi",150);
cout<<"Information of first city"<<endl;
c1.display();
cout<<"Information of second city"<<endl;
c2.display();
cout<<"sum of DistFromKtm of Pokhara and Dhangadi="<<c3.AddDistance(c1,c2);
return 0;
}
```

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]

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

```cpp
class volume
{
private:
int lf,wf,hf,li,wi,hi;
float vol,length,width,height;
public:
void getdata();
void convertdim();
void display();
};
void volume::getdata()
{
cout<<"Enter the length of room in feet and inches"<<endl;
cin>>lf>>li;
cout<<"Enter the width of room in feet and inches"<<endl;
cin>>wf>>wi;
cout<<"Enter the Height of room in feet and inches"<<endl;
cin>>hf>>hi;
}
void volume::convertdim()
{
length=lf+(float)li/12;
width=wf+(float)wi/12;
height=hf+(float)hi/12;
cout<<"The length of room ="<<length<<endl;
cout<<"The width of room ="<<width<<endl;
cout<<"The Height of room="<<height<<endl;
}
void volume::display()
{
vol=length*width*height;
cout<<"Volume of Room ="<<vol<<endl;
}
int main()
{
volume v;
v.getdata();
v.convertdim();
v.display();
return 0;
}
```

18. 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.

```cpp
#include<iostream>
using namespace std;
float calculate(float principle,int time,float rate=15);
int main()
{
float p,r;
int t;
char ch;
cout<<"Enter principle and time"<<endl;
cin>>p>>t;
cout<<"Do you want to enter rate?"<<endl;
cin>>ch;
if(ch=='Y'||ch=='y')
{
cout<<"Enter rate"<<endl;
cin>>r;
cout<<"Interest="<<calculate(p,t,r)<<endl;
}
else
{
cout<<"Interest="<<calculate(p,t)<<endl;
}
return 0;
}
float calculate(float principle,int time,float rate)
{
return(principle*time*rate)/100.0;
}
```

19. Create a class comparison with data members x and y and max and member function getdata() to read the value of x, y, largest() to find greater of two and print() to the greater number.

```cpp
#include<iostream>
using namespace std;
class Comparison
{
private:
int x,y,max;
public:
void getdata()
{
cout<<"Enter two numbers"<<endl;
cin>>x>>y;
}
int largest()
{
max=(x>y)?x:y;
return max;
}
void print()
{
cout<<"Greater number="<<max<<endl;
}
};
int main()
{
Comparison c1;
c1.getdata();
c1.largest();
c1.print();
return 0;
}
```

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)

```cpp
#include<iostream>
using namespace std;
class Student
{
private:
char name[20];
int roll;
float me,mm,ms,total;
public:
void init()
{
cout<<"Enter roll number"<<endl;
cin>>roll;
cout<<"Enter name"<<endl;
cin>>name;
cout<<"Enter Marks in English,Math and science"<<endl;
cin>>me>>mm>>ms;
}
void calctotal()
{
total=me+mm+ms;
}
void display()
{
cout<<"Name:"<<name<<endl;
cout<<"RollNo:"<<roll<<endl;
cout<<"Marks in English:"<<me<<endl;
cout<<"Marks in Math:"<<mm<<endl;
cout<<"Marks in science:"<<ms<<endl;
cout<<"Total:"<<total<<endl;
}
};
int main()
{
Student st;
st.init();
st.calctotal();
st.display();
return 0;
}
```

21. Create classes called Vechile1, Vechile2 and Vechile3 with each of having one private member named price. Add member function to set a price(say setPrice()) one each class. Add one more function max() that is friendly to all classes. max() function should compare private member named price of three classes and show maximum among them. Create one-one object of each class and then set a value on them. Display the maximum price among them.

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

class Vechile2;
class Vechile3;

class Vechile1
{
    private:
    long int price;public:
      void setPrice(long int p)
      {
        price=p;
      }
    friend void max(Vechile1,Vechile2,Vechile3);
};
class Vechile2
{
     private:
    long int price;
    public:
    void setPrice(long int p)
    {
      price=p;
     }
     friend void max(Vechile1,Vechile2,Vechile3);
};
```

```cpp
class Vechile3
{
private:
long int price;
public:
void setPrice(long int p)
{
price=p;
}
friend void max(Vechile1,Vechile2,Vechile3);
};

void max(Vechile1 v1,Vechile2 v2,Vechile3 v3)
{
        if(v1.price>v2.price&&v1.price>v3.price)
        {
        cout<<"Maximum price="<<v1.price<<endl;
        }
        else if(v2.price>v3.price)
        {
        cout<<"Maximum price="<<v2.price<<endl;
        }
        else
        {
        cout<<"Maximum price="<<v3.price<<endl;
        }
}
int main()
{
Vechile1 v1;
Vechile2 v2;
Vechile3 v3;
v1.setPrice(5032323);
v2.setPrice(4543043);
v3.setPrice(1232343);
max(v1,v2,v3);
return 0;
}
```

22. Define a class to represent a bank account .Include the following

Data members

• Name of the depositor

• Account number

• Type of account

• Balance amount in the account

Member functions

• To assign Initial values

• To deposit an amount

• To withdraw an amount after checking the balance

• To display name and balance

Write main program to test the program

```cpp
#include<iostream>
#include<string.h>
using namespace std;
class Bank
{
  char name[20];
long int acno;
char acctype[20];
float bal;
public:
Bank(long int ano, char n[], char at[], float b) //Parameterized Constructor
{
acno=ano;
strcpy(name,n);
strcpy(acctype, at);
bal=b;
}

void deposit();
void withdraw();
void display();
};
void Bank::deposit()
{
int damt;
cout<<"Enter Deposit Amount"<<endl;
cin>>damt;
bal=bal+damt;
}
```

```cpp
void Bank::withdraw()
{
int wamt;
cout<<"Enter Withdraw Amount"<<endl;
cin>>wamt;
if(wamt>bal)
{
cout<<"Cannot Withdraw Amount";
}
else
{
bal=bal-wamt;
}
}

void Bank::display()
{
cout<<"Account Number="<<acno<<endl;
cout<<"Name="<<name<<endl;
cout<<"Account Type:"<<acctype<<endl;
cout<<"Balance ="<<bal<<endl;
}
int main()
{
int acc_no;char name[100], acc_type[100];
float balance;
cout<<"Enter Details:"<<endl;
cout<<"Enter Account Number"<<endl;
cin>>acc_no;
cout<<"Enter Name"<<endl;
cin>>name;
cout<<"Enter Account Type"<<endl;
cin>>acc_type;
cout<<"Enter Balance"<<endl;
cin>>balance;
Bank b1(acc_no,name,acc_type,balance);
b1.deposit();
b1.withdraw();
b1.display();
return 0;
}
```

23. Create a class Employee with data members (id, name, Post, address, salary) and read information of 20 Employees and display the name and post of employee whose salary is greater than 50000.

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

class Employee
{
private:
int eid;
char name[20];
char post[20];
char address[20];
float salary;
public:
void getdata()
{
cout<<"Enter the Employee id"<<endl;
cin>>eid;
cout<<"Enter the Employee name"<<endl;
cin>>name;
cout<<"Enter the post "<<endl;
cin>>post;
cout<<"Enter the address"<<endl;
cin>>address;
cout<<"Enter the salary"<<endl;
cin>>salary;
}
void display()
{
        if(salary>50000)
        {
        cout<<"Employee id="<<eid<<endl;
        cout<<"Employee name="<<name<<endl;
        cout<<"Post="<<post<<endl;
        cout<<"Address="<<address<<endl;
        cout<<"Salary="<<salary<<endl;
        }
}
};
```

```cpp
int main()
{
Employee e[20];
int i;
for(i=0;i<20;i++)
{
cout<<"Enter the information of Employee:"<<i+1<<endl;
e[i].getdata();
}
cout<<"Information of Employee who has salary greater than 50000 are:"<<endl;
for(i=0;i<20;i++)
{
e[i].display();
}
return 0;
}
```

24.Using class write a program that receives inputs principle amount, time and rate. Keeping rate 12% as the default argument , calculate the simple interest for five customers.

```cpp
#include<iostream>
using namespace std;
class customer
{
private:
float principle,rate,si;
int time;
public:
void setdata(float p,int t,float r=12);
void display();
};

void customer::setdata(float p,int t,float r)
{
principle=p;
time=t;
rate=r;
}
void customer::display()
{
si=(principle*time*rate)/100;
cout<<"Simple Interest="<<si<<endl;
}
```

```cpp
int main()
{
float p;
int t,i;
customer c[5];
for(i=0;i<5;i++)
{
cout<<"Enter principle and time for customer"<<i+1<<endl;
cin>>p>>t;
c[i].setdata(p,t);
}
for(i=0;i<5;i++)
{
cout<<"for customer"<<i+1<<endl;
c[i].display();
}
return 0;
};
```

25. WAP to input marks of 35 students and find the highest marks using Dynamic memory allocation.

```cpp
#include <iostream>
using namespace std;
int main ()
{
int i;
float *ptr,high=0;
ptr= new float[35];
for (i=0; i<35; i++)
{
cout <<"Enter the marks of student"<<i+1<<endl;
cin >> ptr[i];
}
for (i=0; i<35; i++)
{
        if(ptr[i]>high)
        {
        high=ptr[i];
        }
}
cout<<"Highest marks="<<high<<endl;
delete[] ptr;
return 0;
}
```

Unit-3 (program solution)

1) **WAP to enter information of n students and then display is using the concept multiple inheritance[PU: 2015 fall]**

```cpp
#include <iostream>
using namespace std;
class Student_info
{
private:
char name[25];
int age;
public:
void get_info()
{
cout<<"Enter name and age of student"<<endl;
cin>>name>>age;
}
void disp_info()
{
cout<<"Name of student:"<<name<<endl;
cout<<"Age of student:"<<age<<endl;
}
};
class Faculty
{
private:
char faculty[20];
public:
void get_faculty()
{
cout<<"Enter Faculty of student"<<endl;
cin>>faculty;
}
void disp_faculty()
{
cout<<"Faculty of student:"<<faculty<<endl;
}
};
```
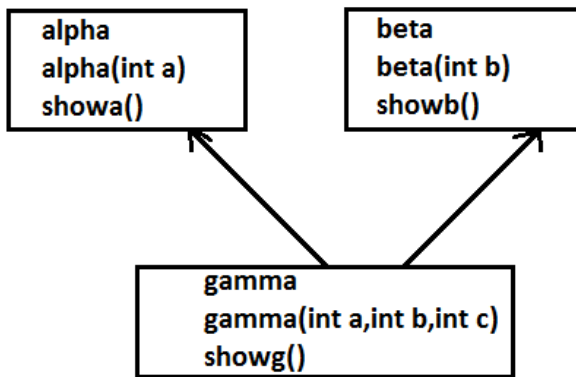
```cpp
class Attendance:public Faculty,public Student_info
{
private:
int attendance;
public:
void get_attendance()
{
cout<<"Enter student's attendance percentage"<<endl;
cin>>attendance;
}
void disp_attendance()
{
cout<<"Student's attendance percentage ="<<attendance<<endl;
}
};

int main()
{
int i,n;
Attendance a[50];
cout<<"Enter Number of students"<<endl;
cin>>n;
for(i=0;i<n;i++)
{
cout<<"Enter information of student:"<<i+1<<endl;
a[i].get_info();
a[i].get_faculty();
a[i].get_attendance();
}
for(i=0;i<n;i++)
{
cout<<"Information of student:"<<i+1<<endl;
a[i].disp_info();
a[i].disp_faculty();
a[i].disp_attendance();
}
return 0;
}
```

2) **WAP to enter information of n students and then display is using the concept multilevel inheritance.**

```cpp
#include <iostream>
using namespace std;
class Student_info
{
private:
char name[25];
int age;
public:
void get_info()
{
cout<<"Enter name and age of student"<<endl;
cin>>name>>age;
}
void disp_info()
{
cout<<"Name of student:"<<name<<endl;
cout<<"Age of student:"<<age<<endl;
}
};

class Faculty:public Student_info
{
private:
char faculty[20];
public:
void get_faculty()
{
cout<<"Enter Faculty of student"<<endl;
cin>>faculty;
}
void disp_faculty()
{
cout<<"Faculty of student:"<<faculty<<endl;
}
};
```

```cpp
class Attendance:public Faculty
{
private:
int attendance;
public:
void get_attendance()
{
cout<<"Enter student's attendance percentage"<<endl;
cin>>attendance;
}
void disp_attendance()
{
cout<<"Student's attendance percentage ="<<attendance<<endl;
}
};

int main()
{
int i,n;
Attendance a[50];
cout<<"Enter Number of students"<<endl;
cin>>n;
for(i=0;i<n;i++)
{
cout<<"Enter information of student:"<<i+1<<endl;
a[i].get_info();
a[i].get_faculty();
a[i].get_attendance();
}
for(i=0;i<n;i++)
{
cout<<"Information of student:"<<i+1<<endl;
a[i].disp_info();
a[i].disp_faculty();
a[i].disp_attendance();
}
return 0;
}
```

3) Write a complete program with reference to the given figure.

```
┌─────────────────┐        ┌─────────────────┐
│ alpha           │        │ beta            │
│ alpha(int a)    │        │ beta(int b)     │
│ showa()         │        │ showb()         │
└─────────────────┘        └─────────────────┘
         ↖                         ↗
          ┌────────────────────────┐
          │ gamma                  │
          │ gamma(int a,int b,int c)│
          │ showg()                │
          └────────────────────────┘
```

```cpp
#include<iostream>
using namespace std;
class alpha
{
private:
int x;
public:
alpha(int a)
{
x=a;
}
void showa()
{
cout<<"x="<<x<<endl;
}
};

class beta
{
private:
int y;
public:
beta(int b)
{
y=b;
}
void showb()
{
cout<<"y="<<y<<endl;
}
};
```

```
class gamma:public beta,public alpha
{
private:
int z;
public:
gamma(int a,int b,int c):alpha(a),beta(b)
{
z=c;
}
void showg()
{
cout<<"z="<<z<<endl;
}
};

int main()
{
gamma g(5,10,15);
g.showa();
g.showb();
g.showg();
return 0;
}
```

4) The following figure shows the minimum information required for each class. Write a progam by realizing the necessary member functions to read and display information of individual object. Every class should contain at least one constructor and should be inherited from other classes as well.[PU:2019 fall]


```
#include<iostream>
#include<string.h>
using namespace std;
```
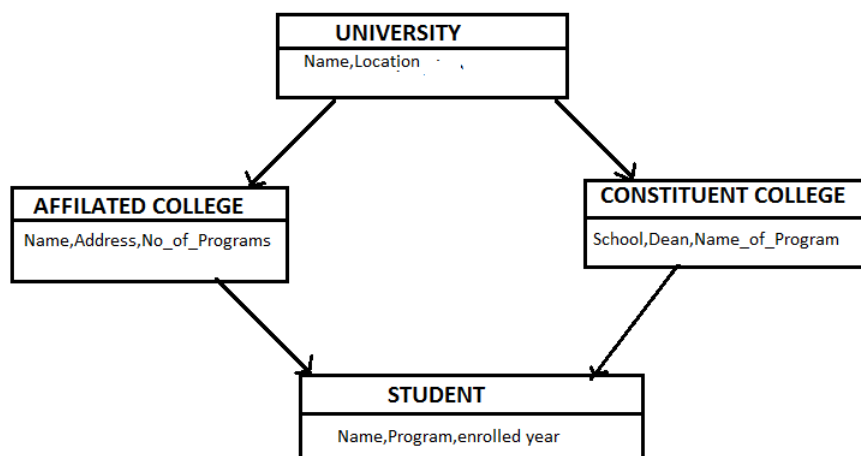
```cpp
class University
{
private:
char uname[20];
char location[20];
public:
University(char un[],char loc[])
{
strcpy(uname,un);
strcpy(location,loc);
}

void display()
{
cout<<"University name="<<uname<<endl;
cout<<"University location="<<location<<endl;
}
};

class Affiliated_College:virtual public University
{
private:
char cname[20];
char address[20];
int no_of_programs;
public:
Affiliated_College(char un[],char loc[],char cn[],char addr[],int nop):University(un,loc)
{
strcpy(cname,cn);
strcpy(address,addr);
no_of_programs=nop;
}
void display()
{
cout<<"Affiliated College Name="<<cname<<endl;
cout<<"Address="<<address<<endl;
cout<<"Number of programs="<<no_of_programs<<endl;
}
};
```

```cpp
class Constituent_College:virtual public University
{
private:
char school[20];
char dean[20];
char name_of_program[20];
public:
Constituent_College(char un[],char loc[],char sn[],char d[],char npro[]):University(un,loc)
{
strcpy(school,sn);
strcpy(dean,d);
strcpy (name_of_program,npro);
}

void display()
{
cout<<"School name="<<school<<endl;
cout<<"Dean name="<<dean<<endl;
cout<<"Name of Program="<<name_of_program<<endl;
}
};

class Student:public Affiliated_College,public Constituent_College
{
private:
char student_name[20];
char program[20];
int enrolled_year;
public:
Student(char un[],char loc[],char cn[],char addr[],int nop, char sn[],
char d[], char npro[],char stn[],char pro[],int year):
Affiliated_College(un,loc,cn,addr,nop),Constituent_College(un,loc,sn,d,npro),University(un,loc)
{
strcpy(student_name,stn);
strcpy(program,pro);
enrolled_year=year;
}
void display()
{
cout<<"Student name="<<student_name<<endl;
cout<<"Program="<<program<<endl;
cout<<"Enrolled year="<<enrolled_year<<endl;
}
};
```

```cpp
int main()
{
Student st("Pokhara","Dhungepatan","EEC","Sanepa",3,
"Engineering","Bharat","Civil","Pradip","Civil",2018);
st.University::display();
st.Affiliated_College::display();
st.Constituent_College::display();
st.display();
return 0;
}
```

5) An educational institution wishes to maintain a database of its employees. The database is divided into a number of classes whose hierarchical relationship are shown below. The figure also shows minimum information requires for each class. Specify all the classes and define functions to create database and retrieve individual information when required.
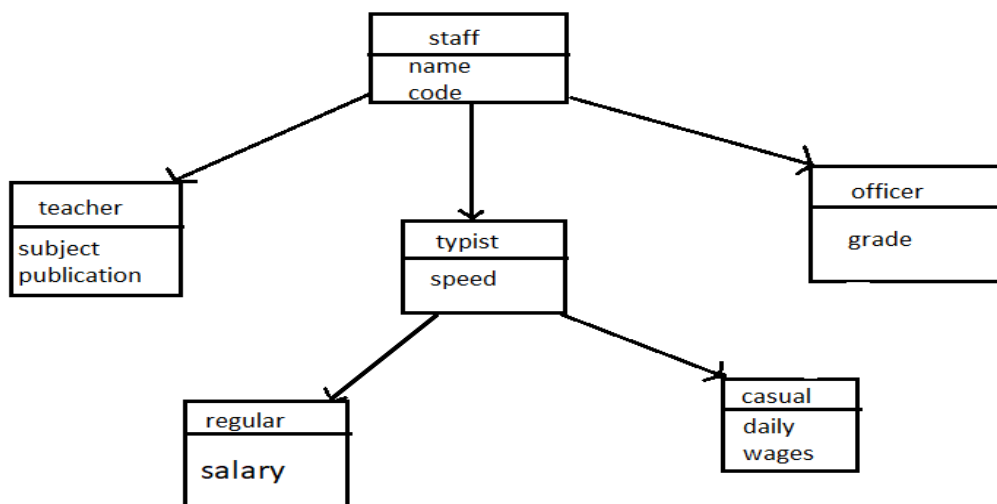

```cpp
#include<iostream>
using namespace std;
class Staff
{
private:
char name[20];
int code;
public:
void get_staff()
{
cout<<"Enter Name and code of staff"<<endl;
cin>>name>>code;
}
```

```cpp
void disp_staff()
{
cout<<"Name="<<name<<endl;
cout<<"Code="<<code<<endl;
}
};


class Teacher:public Staff
{
private:
char subject[20],publication[20];
public:
void get_teacher()
{
cout<<"Enter Subject and Publication"<<endl ;
cin>>subject>>publication;
}
void disp_teacher()
{
cout<<"Subject="<<subject<<endl;
cout<<"Publication="<<publication<<endl;
}
};

class Officer:public Staff
{
private:
char grade;
public:
void get_officer()
{
cout<<"Enter the grade"<<endl;
cin>>grade;
}
void disp_officer()
{
cout<<"Grade="<<grade<<endl;
}
};
```

```cpp
class Typist:public Staff
{
private:
int speed;
public:
void get_typist()
{
cout<<"Enter Speed"<<endl;
cin>>speed;
}
void disp_typist()
{
cout<<"Speed="<<speed<<endl;
}
};

class Regular:public Typist
{
private:
int salary;
public:
void get_regular()
{
cout<<"Enter Salary"<<endl;
cin>>salary;
}
void disp_regular()
{
cout<<"Salary="<<salary<<endl;
}
};

class Casual:public Typist
{
private:
int wages;
public:
void get_casual()
{
cout<<"Enter Daily wages"<<endl;
cin>>wages;
}
void disp_casual()
{
cout<<"Daily wages="<<wages<<endl;
}
};
```

```cpp
int main()
{
Teacher t;
Officer o;
Regular r;
Casual c;
t.get_staff();
t.get_teacher();
t.disp_staff();
t.disp_teacher();
o.get_staff();
o.get_officer();
o.disp_staff();
o.disp_officer();
r.get_staff();
r.get_typist();
r.get_regular();
r.disp_staff();
r.disp_typist();
r.disp_regular();
c.get_staff();
c.get_typist();
c.get_casual();
c.disp_staff();
c.disp_typist();
c.disp_casual();
return 0;
}
```

6) The following figure shows minimum information required for each class.

i) Write a Program to realize the above program with necessary member functions to create the database and retrieve individual information
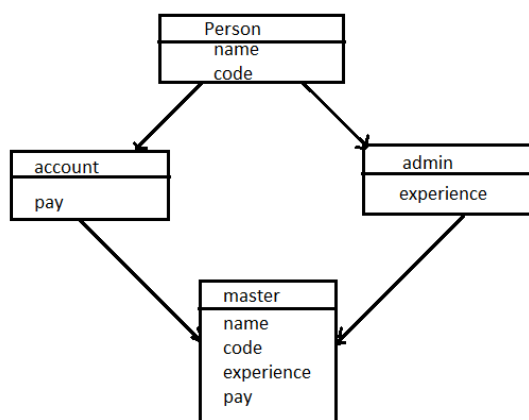
```cpp
#include<iostream>
using namespace std;
class Person
{
protected:
char name[20];
int code;
};

class Account:virtual public Person
{
protected:
float pay;
};

class Admin:virtual public Person
{
protected:
int experience;
};

class Master:public Account,public Admin
{
public:
void getinfo()
{
cout<<"Enter name of person"<<endl;
cin>>name;
cout<<"Enter code"<<endl;
cin>>code;
cout<<"Enter pay for person"<<endl;
cin>>pay;
cout<<"Enter experience"<<endl;
cin>>experience;
}
void display()
{
cout<<"Name:"<<name<<endl;
cout<<"Code:"<<code<<endl;
cout<<"Pay:"<<pay<<endl;
cout<<"Experience:"<<experience<<"years"<<endl;
}
};
```

```
int main()
{
Master m;
m.getinfo();
m.display();
return 0;
}
```

    ii) Rewrite the above program using constructor on each class to initialize the data
        members.

```
#include<iostream>
#include<string.h>
using namespace std;

class Person
{
protected:
char name[20];
int code;

public:
Person(char n[],int c)
{
strcpy(name,n);
code=c;
}
};

class Account:virtual public Person
{
protected:
float pay;
public:
Account(char n[],int c,float p):Person(n,c)
{
pay=p;
}
};
```

```cpp
class Admin:virtual public Person
{
protected:
int experience;
Admin(char n[],int c,int e):Person(n,c)
{
experience=e;
}
};
class Master:public Account,public Admin
{
public:
Master(char n[],int c,float p,int e):Admin(n,c,e),Account(n,c,p),Person(n,c)
{
strcpy(name,n);
code=c;
pay=p;
experience=e;
}
void display()
{
cout<<"Name:"<<name<<endl;
cout<<"Code:"<<code<<endl;
cout<<"Pay:"<<pay<<endl;
cout<<"Experience:"<<experience<<"years"<<endl;
}
};
int main()
{
Master m("Pradip",121,23000.56,2);
m.display();
return 0;
}
```

7. Write a program that allow you to book a ticket for person and use two classes Person, Reservation. Class Reservation is composite class/ container class.
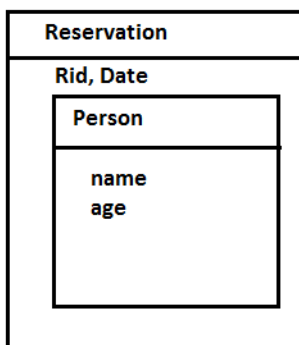
```cpp
#include<iostream>
using namespace std;
class Person
{
private:
char name[20];
int age;
public:
void getdata()
{
cout<<"Enter name and age of person"<<endl;
cin>>name>>age;
}
void display()
{
cout<<"Name="<<name<<endl;
cout<<"Age="<<age<<endl;
}
};

class Reservation
{
private:
int rid;
char date[10];
Person p;
public:
void getdata()
{
p.getdata();
cout<<"Enter Reservation ID"<<endl;
cin>>rid;
cout<<"Enter date in YY-MM-DD format "<<endl;
cin>>date;
}
void display()
{
p.display();
cout<<"Reservation ID="<<rid<<endl;
cout<<"Reservation date="<<date;
}
};
```

```
int main()
{
Reservation r;
r.getdata();
r.display();
return 0;
}
```

8. The following figure shows the minimum information required for each class. Write a program to realize the above program with necessary member functions to create the database and retrieve individual information .Every class should contain at least one constructor and should be inherited to other classes as well.[PU:2010 spring][PU 2009 fall]
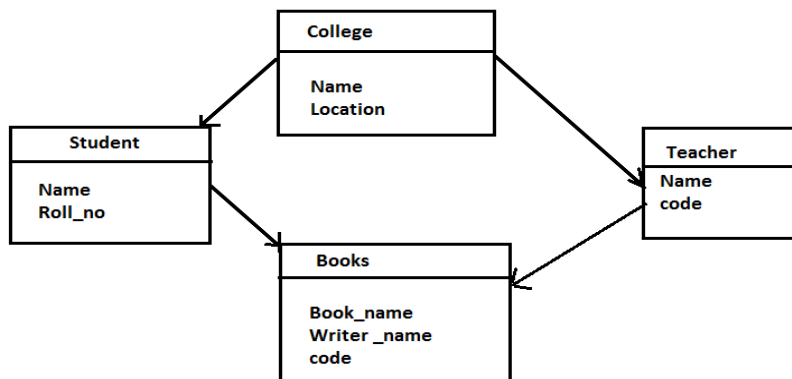

```
#include<iostream>
#include<string.h>
using namespace std;
class College
{
private:
char cname[20];
char location[20];
public:
College(char cn[],char loc[])
{
strcpy(cname,cn);
strcpy(location,loc);
}
void display()
{
cout<<"College name="<<cname<<endl;
cout<<"College location="<<location<<endl;
}
};
```

```cpp
class Student:virtual public College
{
private:
char sname[20];
int roll;
public:
Student(char cn[],char loc[],char sn[],int r):College(cn,loc)
{
strcpy(sname,sn);
roll=r;
}
void display()
{
cout<<"Student name="<<sname<<endl;
cout<<"Roll no="<<roll<<endl;
}
};

class Teacher:virtual public College
{
private:
char tname[20];
int tcode;
public:
Teacher(char cn[],char loc[],char tn[],int tc):College(cn,loc)
{
strcpy(tname,tn);
tcode=tc;
}
void display()
{
cout<<"Teacher name="<<tname<<endl;
cout<<"Teacher Code="<<tcode<<endl;
}
};
class Books:public Student,public Teacher
{
private:
char book_name[20];
char writer_name[20];
int book_code;
```

```cpp
public:
Books(char cn[],char loc[],char sn[],int r, char tn[],int tc,
char bn[],char wn[],intbc):Teacher(cn,loc,tn,tc),Student(cn,loc,sn,r),College(cn,loc)
{
strcpy(book_name,bn);
strcpy(writer_name,wn);
book_code=bc;
}
void display()
{
cout<<"Book name="<<book_name<<endl;
cout<<"Writer name="<<writer_name<<endl;
cout<<"Book code="<<book_code<<endl;
} };
int main()
{
Books b("EEC","sanepa","Ram",47,"Pradip",151,"OOPs","Timothy Budd",53421);
b.College::display();
b.Student::display();
b.Teacher::display();
b.display();
return 0;
}
```

9) Develop a complete program for an institution, which wishes to maintain a database of its staff. The database is divided into number of classes whose hierarchical relationship is shown in the following diagram. specify all classes and define constructors and functions to create database and retrieve the individual information as per requirements.


```cpp
#include<iostream>
#include<string.h>
using namespace std;



class STAFF
```
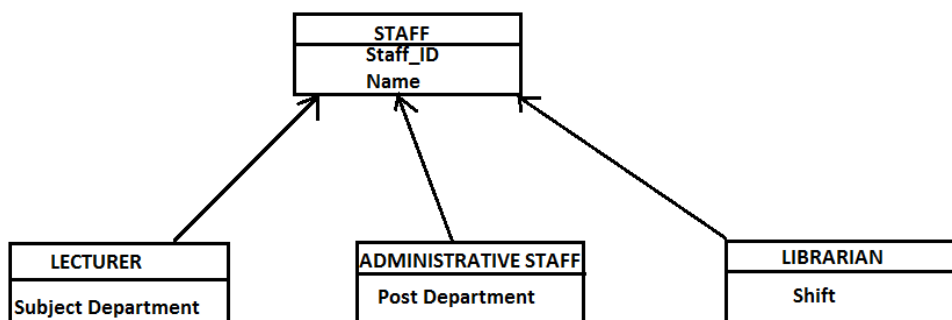
```cpp
{
private:
int staff_id;
char name[20];
public:
STAFF(int sid,char sn[])
{
staff_id=sid;
strcpy(name,sn);
}
void display()
{
cout<<"Staff ID:"<<staff_id<<endl;
cout<<"Name:"<<name<<endl;
}
};

class LECTURER:public STAFF
{
private:
char subject[20];
char department[20];
public:
LECTURER(int sid,char sn[],char sub[],char dept[]):STAFF(sid,sn)
{
strcpy(subject,sub);
strcpy(department,dept);
}
void display()
{
cout<<"Subject:"<<subject<<endl;
cout<<"Department:"<<department<<endl;
}
};
class ADMINISTRATIVE_STAFF:public STAFF
{
private:
char post[20];
char department[20];
public:
ADMINISTRATIVE_STAFF(int sid,char sn[],char p[],char dept[]):STAFF(sid,sn)
{
strcpy(post,p);
strcpy(department,dept);
}

void display()
```

```cpp
{
cout<<"Post:"<<post<<endl;
cout<<"Department:"<<department<<endl;
}
};

class LIBRARIAN:public STAFF
{
private:
char shift[20];
public:
LIBRARIAN(int sid,char sn[],char s[]):STAFF(sid,sn)
{
strcpy(shift,s);
}
void display()
{
cout<<"Shift:"<<shift<<endl;
}
};

int main()
{
LECTURER l(101,"Pradip Paudel","C programming","Computer");
l.STAFF::display();
l.display();
ADMINISTRATIVE_STAFF a(212,"Ekraj Acharaya","Admin","Administration");
a.STAFF::display();
a.display();
LIBRARIAN lib(314,"Sita sharma","Morning");
lib.STAFF::display();
lib.display();
return 0;
}
```

10) Develop a complete program for an institution which wishes to maintain a database of its staff. Declare a base class Staff which include staff_id and name.Now develop a records forthe following staffs with the given information below.
Lecturer(subject,department)
Administrative staff (Post,department)

```cpp
#include<iostream>
#include<string.h>
using namespace std;


class Staff
```

```cpp
{
private:
int staff_id;
char name[20];
public:
void get_staff()
{
cout<<"Enter staff id"<<endl;
cin>>staff_id;
cout<<"Enter name"<<endl;
cin>>name;
}
void display_staff()
{
cout<<"Staff ID:"<<staff_id<<endl;
cout<<"Name:"<<name<<endl;
}
};

class Lecturer:public Staff
{
char subject[20];
char department[20];
public:
void get_lecturer()
{
cout<<"Enter subject"<<endl;
cin>>subject;
cout<<"Enter department"<<endl;
cin>>department;
}
void display_lecturer()
{
cout<<"Subject:"<<subject<<endl;
cout<<"Department:"<<department<<endl;
}
};

class Administrative_staff:public Staff
```

```cpp
{
char post[20];
char department[20];
public:
void get_administrative()
{
cout<<"Enter post"<<endl;
cin>>post;
cout<<"Enter department"<<endl;
cin>>department;
}
void display_administrative()
{
cout<<"Post:"<<post<<endl;
cout<<"Department:"<<department<<endl;
}
};
int main()
{
Lecturer l;
l.get_staff();
l.get_lecturer();
l.display_staff();
l.display_lecturer();
Administrative_staff a;
a.get_staff();
a.get_administrative();
a.display_staff();
a.display_administrative();
return 0;
}
```

12. Implement the below given in class diagram in C++.Assume necessary function   yourself.
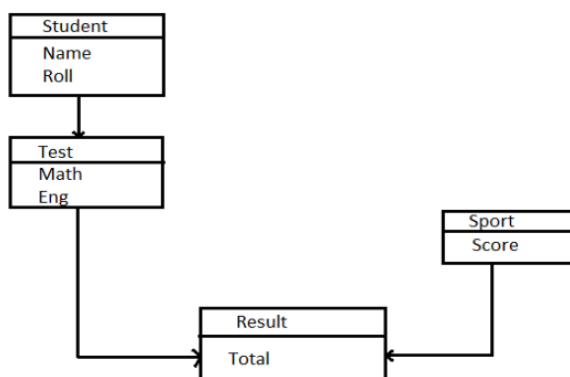
```cpp
#include<iostream>
using namespace std;
class Student
{
private:
char name[20];
int roll;
public:
void get_student()
{
cout<<"Enter name and roll no"<<endl;
cin>>name>>roll;
}
void disp_student()
{
cout<<"Name="<<name<<endl;
cout<<"Rollno="<<roll<<endl;
}
};

class Test:public Student
{
protected:
float math,eng;
public:
void get_test()
{
cout<<"Enter marks in math and english"<<endl;
cin>>math>>eng;
}
void disp_test()
{
cout<<"marks in math="<<math<<endl;
cout<<"marks in english="<<eng<<endl;
}
};
class Sport
{
protected:
float score;
public:
void get_sport()
{
cout<<"Enter score"<<endl;
cin>>score;
}
```

```
void disp_sport()
{
cout<<"Score="<<score<<endl;
}
};

class Result:public Test,public Sport
{
private:
float total;
public:
void disp_total()
{
total=eng+math+score;
cout<<"Total="<<total<<endl;
}
};

int main()
{
Result r;
r.get_student();
r.get_test();
r.get_sport();
r.disp_student();
r.disp_test();
r.disp_sport();
r.disp_total();
return 0;
}
```
`

13. Create a class student with two data members represent name and roll. Use appropriate member function to read and print these data members name and roll. Derive a class marks from student that has additional data member sessional1, sessional2 to store sessional marks. Derive another class result from marks and add the sessional marks. Use appropriate member function to read and display data in the class.

```
#include<iostream>
using namespace std;
class Student
{
protected:
char name[20];
int roll;
```

```cpp
public:
void getdata()
{
cout<<"Enter the name"<<endl;
cin>>name;
cout<<"Enter the rollno"<<endl;
cin>>roll;
}
void putdata()
{
cout<<"Name="<<name<<endl;
cout<<"Roll="<<roll<<endl;
}
};
class Marks:public Student
{
protected:
float sessional1;
float sessional2;public:
void getmarks()
{
cout<<"Enter sessional1 marks"<<endl;
cin>>sessional1;
cout<<"Enter sessional2 marks"<<endl;
cin>>sessional2;
}
void putmarks()
{
cout<<"sessional1="<<sessional1<<endl;
cout<<"sessional2="<<sessional2<<endl;
}
};
class Result:public Marks
{
private:
float total;
public:
void display()
{
total=sessional1+sessional2;
cout<<"Total marks="<<total<<endl;
}
};
```

```cpp
int main()
{
Result r;
r.getdata();
r.getmarks();
r.putdata();
r.putmarks();
r.display();
return 0;
}
```

14. Create a class student with two data members to represent name and age. Use member function to read and print those data. From this derive a class called boarder with member data to represent room number .Derive another class called day-scholar from class student with member data to represent address of student. In both derived class use function to read and print respective data.

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

class Student
{
protected:
char name[20];
int age;
public:
void read()
{
cout<<"Enter name"<<endl;
cin>>name;
cout<<"Enter age"<<endl;
cin>>age;
}
void print()
{
cout<<"Name="<<name<<endl;
cout<<"Age="<<age<<endl;
}
};
```

```cpp
class Boarder:public Student
```

```cpp
{
private:
int roomnumber;
public:
void getroomno()
{
cout<<"Enter room number"<<endl;
cin>>roomnumber;
}
void putroomno()
{
cout<<"Room number="<<roomnumber<<endl;
}
};
class Day_scholar:public Student
{
private:
char address[20];
public:
void getaddress()
{
cout<<"Enter address"<<endl;
cin>>address;
}
void putaddress()
{
cout<<"Address="<<address<<endl;
}
};
int main()
{
Boarder b;
Day_scholar d;
b.read();
b.getroomno();
d.read();
d.getaddress();
b.print();
b.putroomno();
d.print();
d.putaddress();
return 0;   }
```

UNIT 4 (Program solution)

1. **Write a program finding area of square, rectangle, triangle. Use function overloading technique**.

```
#include<iostream>
using namespace std;
class calcarea
{
public:
int area(int x)
{
return (x*x);
}
int area(int x,int y)
{
return (x*y);
}
float area(float x,float y)
{
return (0.5*x*y);
}
};
int main()
{
int length,breadth,side;
float base,height;
calcarea c1,c2,c3;
cout<<"Enter side of square:"<<endl;
cin>>side;
cout<<"Area of square="<<c1.area(side)<<endl;
cout<<"Enter length and breadth of rectangle:"<<endl;
cin>>length>>breadth;
cout<<"Area of rectangle="<<c2.area(length,breadth)<<endl;
cout<<"Enter base and height of triangle:"<<endl;
cin>>base>>height;
cout<<"Area of Triangle="<<c3.area(base,height)<<endl;
return 0;
}
```

2. WAP to overload multiplication operator showing (*) showing the multiplication of two objects.**[PU:2020 fall]**

```cpp
#include<iostream>
using namespace std;
class Arithmetic
{
private:
float num;
public:
void getdata()
{
cout<<"Enter the number"<<endl;
cin>>num;
}
void display()
{
cout<<num<<endl;
}
Arithmetic operator*(Arithmetic b);
};
Arithmetic Arithmetic::operator*(Arithmetic b)
{
Arithmetic temp;
temp.num=num*b.num;
return temp;
}

int main()
{
Arithmetic a,b,c;
a.getdata();
b.getdata();
c=a*b;
cout<<"Multiplication of two objects="<<endl;
c.display();
return 0;
}
```

3. Write a program   with class fibo to realize the following code snippet. **[PU: 2014 fall]**
fibo f=1;
for (i=1;i<=10;i++)
{
++f;
f.display();
}
[Hint: overload ++ operator and conversion technique.]

```cpp
#include<iostream>
using namespace std;
class fibo
{
private:
int a,b,c;
public:
fibo(int x)
{
a=-x;
b=x;
c=a+b;
}
void display()
{
cout<<c<<",";
}
void operator++()
{
a=b;
b=c;
c=a+b;
}
};
int main()
{
fibo f=1;
int i;
for(i=1;i<=10;i++)
{
++f;
f.display();
}
return 0;
}
```

4. Design a soccer player class that includes three ineger fields:a player's jersey number,number of goals,number of assists and necessary constructors to initialize the data members.Overload the > operator (Greater than) .One player is considered greater than another if the sum of goals plus assists is greater than that of others. Create an array of 11 soccer players, then use the overloaded > operator to find the greater total of goal plus assists.**[PU:2015 fall]**

```cpp
#include<string.h>
#include<iostream>
using namespace std;
class Soccerplayer
{
private:
int jerseyno;
int goals;
int assists;
public:
Soccerplayer()
{ }
Soccerplayer(int jn,int g,int a)
{
jerseyno=jn;
goals=g;
assists=a;
}
void display()
{
cout<<"Jersey number="<<jerseyno<<endl;
cout<<"Number of Goals="<<goals<<endl;
cout<<"Number of assists="<<assists<<endl;
}
friend int operator >(Soccerplayer &s1,Soccerplayer &s2);
};
int operator >(Soccerplayer &s1,Soccerplayer &s2)
{
int sum1 =s1.goals +s1.assists;
int sum2 =s2.goals +s2.assists;
if(sum1 >sum2 )
{
return 1;
}
else
{
return 0;
}
}
```

```cpp
int main ()
{
int ngoals,jno,nassists,max;
Soccerplayer *s[11];
for(int i=0;i<11;i++)
{
cout<<"Enter player jersey number"<<endl;
cin>>jno;
cout<<"Enter number of goals"<<endl;
cin>>ngoals;
cout<<"Enter number of assists"<<endl;
cin>>nassists;
s[i]= new Soccerplayer(jno,ngoals,nassists);
}
cout<<"The details of players are:"<<endl;
for(int i=0;i<11;i++)
{
s[i]->display();
}
max=0;
for(int i=0;i<10;i++)
{
if(*s[i+1]>*s[max])
{
max=i+1;
}
}
cout<<"The details of player with highest points(goals+assists) :" <<endl;
s[max]->display();
return 0;
}
```

5. Make a class called memory with member data to represent bytes, kilobytes and megabytes .Create an object of memory and initialize data members and then convert them into number of   bytes using suitable conversion method.

```cpp
#include<iostream>
using namespace std;
class memory
{
private:
int mb;
int kb;
int byte;
```

```cpp
public:
memory(int m,int k,int b)
{
mb=m;
kb=k;
byte=b;
}
void display()
{
cout<<mb<<"megabytes"<<endl;
cout<<kb<<"kilobytes"<<endl;
cout<<byte<<"bytes"<<endl;
}
operator long int()
{
        return(mb*1024*1024+1024*kb+byte);
}
};
int main()
{
memory m1(1,38,177);
long int total;
total=m1;
cout<<"Given memory is"<<endl;
m1.display();
cout<<"After type conversion total number of bytes="<<total<<endl;
return 0;
}
```

6. Write a program to create a class age with attributes YY, MM, DD and convert the object of class age to basic data type int days.

```cpp
#include<iostream>
using namespace std;
class Age
{
private:
int yy,mm,dd;
public:
Age(int y,int m,int d)
{
yy=y;
mm=m;
dd=d;
}
```

```cpp
void display()
{
cout<<"year="<<yy<<endl;
cout<<"month="<<mm<<endl;
cout<<"Day="<<dd<<endl;
}
operator int()
{
return(365*yy+30*mm+dd);
}
};
int main()
{
Age a1(25,6,15);
int days;
a1.display();
days=a1;
cout<<"Number of days="<<days<<endl;
return 0;
}
```

7. Define two classes named 'Polar' and 'Rectangle' to represent points in polar and rectangle systems. Use conversion routines to convert from one system to another system.**[PU:2005 fall][PU:2014 fall]**

```cpp
#include <iostream>
#include <math.h>
using namespace std;
class Polar
{
private:
float radius;
float angle;
public:
Polar()
{
radius=0.0;
angle=0.0;
}
Polar(float r, float a)
{
radius=r;
angle=a;
}
```

```cpp
void display()
{
cout<<"("<<radius<<","<<angle<<")"<<endl;
}
float getr()
{
return radius;
}
float geta()
{
return angle;
}
};
class Rectangle
{
private:
float xco;
float yco;
public:
Rectangle()
{
xco=0.0;
yco=0.0;
}
Rectangle(float x,float y)
{
    xco=x;
    yco=y;
}
void display()
{
cout<<"("<<xco<<","<<yco<<")"<<endl;
}
operator Polar()
{
float a=atan(yco/xco);
float r=sqrt(xco*xco+yco*yco);
return Polar(r,a);
}
Rectangle(Polar p)
{
float r=p.getr();
float a=p.geta();
xco=r*cos(a);
yco=r*sin(a);
}
};
```

```cpp
int main()
{
cout<<"Conversion from Polar to Rectangular coordinates"<<endl;
Polar p1(10.0,0.785398);
Rectangle r1;
r1=p1;
cout<<"Polar coordinates:"<<endl;
p1.display();
cout<<"Rectangular coordinates:"<<endl;
r1.display();
cout<<"Conversion from Rectagular to Polar coordinates"<<endl;
Rectangle r2(7.07107,7.07107);
Polar p2;
p2=r2;
cout<<"Rectangular coordinates:"<<endl;
r2.display();
cout<<"Polar coordinates:"<<endl;
p2.display();
return 0;
}
```

8. Create a base class student. Use the class to store name, dob, rollno and includes the member function getdata(),discount().Derive two classes PG and UG   from student. make dispresult() as virtual function in the derived class to suit the requirement.**[PU:2013 spring]**

```cpp
#include<iostream>
using namespace std;
class Student
{
protected:
char name[20],dob[10];
int roll;
float dis,fees;
public:
void getdata()
{
cout<<"Enter the name and roll.no of student"<<endl;
cin>>name>>roll;
cout<<"Enter the date of birth format in yy-mm-dd"<<endl;
cin>>dob;
cout<<"Enter the fees"<<endl;
cin>>fees;
}
```

```cpp
void discount()
{
cout<<"Enter discount given to student"<<endl;
cin>>dis;
}
virtual void dispresult()//Empty virtual function
{ }
};

class UG:public Student
{
private:
float paidamount;
public:

void dispresult()
{
paidamount=fees-dis;
cout<<"Name:"<<name<<endl;
cout<<"Roll no:"<<roll<<endl;
cout<<"Date of Birth:"<<dob<<endl;
cout<<"Discount:"<<dis<<endl;
cout<<"Fees:"<<fees<<endl;
cout<<"Fees to be paid="<<paidamount<<endl;
}
};

class PG:public Student
{
private:
float paidamount;
public:
void dispresult()
{
paidamount=fees-dis;
cout<<"Name:"<<name<<endl;
cout<<"Roll no:"<<roll<<endl;
cout<<"Date of Birth:"<<dob<<endl;
cout<<"Discount:"<<dis<<endl;
cout<<"Fees:"<<fees<<endl;
cout<<"Fees to be paid="<<paidamount<<endl;
}
};
```

```cpp
int main()
{
Student *st1,*st2;
PG p;
UG u;
st1=&p;
st2=&u;
cout<<"Enter PG student details:"<<endl;
st1->getdata();
st1->discount();
cout<<"Enter UG student details:"<<endl;
st2->getdata();
st2->discount();
cout<<"Details of PG student:"<<endl;
st1->dispresult();
cout<<"Details of UG student:"<<endl;
st2->dispresult();
return 0;
}
```

9. Create a abstract class shape with two members base and height, a member function for initialization and a pure virtual function to compute area().Derive two specific classes, Triangle and Rectangle which override the function area ().use these classes in main function and display the area of triangle and rectangle.[PU:2009 spring]

```cpp
#include<iostream>
using namespace std;
class Shape
{
protected:
float base, height;
public:
void setdimension(float x, float y)
{
base=x;
height=y;
}
virtual void area() =0;
};

class Triangle : public Shape
{
public:
void area()
{
cout<<"Area of triangle="<<(0.5*base*height)<<endl;
}
};
```

```cpp
class Rectangle : public Shape
{
public:
void area()
{
cout<<"Area of Rectangle="<<(base*height)<<endl;
}
};
int main()
{
Shape *bptr;
Triangle t;
Rectangle r;
t.setdimension(10.0, 5.0);
r.setdimension(20.0, 10.0);
bptr= &t;
bptr->area();
bptr = &r;
bptr->area();
return 0;
}
```

10. Write a program to convert Celsius into Fahrenheit using the concept of conversion from one class type to another class type.

```cpp
#include<iostream>
using namespace std;
class Fahrenheit
{
private:
float f;
public:
Fahrenheit()
{
f=0;
}
Fahrenheit(float x)
{
f=x;
}
void display()
{
cout<<"Temperature in fahrenheit="<<f<<endl;
}
};
```

```cpp
class Celsius
{
private:
float c;
public:
Celsius()
{
c=0;
}
Celsius(float x)
{
c=x;
}
void display()
{
cout<<"Temprature in celius="<<c<<endl;
}
operator Fahrenheit()
{
float temp;
temp=c*9/5+32;
return Fahrenheit(temp);
}
};
int main()
{
Celsius c1(37);
Fahrenheit f1;
f1=c1;
c1.display();
f1.display();
return 0;
}
```

11. **Write a program to implement vector addition using operator overloading**

      i)   **Using Friend Function**
      ii)   **Without using Friend Function(using member function)**

Note:

Addition of vectors: Addition of vectors is done by adding the corresponding X, Y and Z magnitudes of the two vectors to get the resultant vector.
Example:
v1 = 1i + 2j + 3k
v2 = 3i + 2j + 1k
Therefore the resultant vector,
v3 = v1 + v2 = 4i + 4j + 4k

i) Using friend function

```cpp
#include<iostream>
using namespace std;
class Vector
{
private:
    int x, y, z;
public:
    Vector()
    {

    }


    Vector(int a, int b, int c)
    {
        x = a;
        y = b;
        z = c;
    }
    void display()
    {
        cout<<x<<"i+"<<y<<"j+"<<z<<"k"<<endl;
    }
    friend Vector operator +(Vector v1,Vector v2);
};
Vector operator +(Vector v1,Vector v2)
    {
        Vector temp;
        temp.x = v1.x + v2.x;
        temp.y= v1.y + v2.y;
        temp.z = v1.z + v2.z;
        return temp;
    }
```

```cpp
int main()
{
    Vector v1(3,4,2);
    Vector v2(6,3,9);
    Vector v3;
    v3=v1+v2;
    cout<<"v1= " ;
    v1.display();
    cout<<"v2= " ;
    v2.display();
    cout<<"v1+v2= ";
    v3.display();
    return 0;
}
```

ii) Without using friend function(using member function)

```cpp
#include<iostream>
using namespace std;
class Vector
{
private:
    int x, y, z;
public:
    Vector()
    {

    }
    Vector(int a, int b, int c)
    {
        x = a;
        y = b;
        z = c;
    }

Vector operator +(Vector v)

    {
        Vector temp;
        temp.x = x + v.x;
        temp.y= y + v.y;
        temp.z = z + v.z;
        return temp;
    }
```

```cpp
        void display()
          {
          cout<<x<<"i+"<<y<<"j+"<<z<<"k"<<endl;
          }
        };

        int main()
        {
              Vector v1(3,4,2);
              Vector v2(6,3,9);
              Vector v3;
              v3=v1+v2;
              cout<<"v1=" ;
              v1.display();
              cout<<"v2=" ;
              v2.display();
              cout <<"v1+v2=";
              v3.display();
              return 0;
        }
```

Output:

v1=3i+4j+2k

v2=6i+3j+9k

v1+v2=9i+7j+11k

13. **Write a program showing ++ and - - operator overloading.**

```cpp
#include <iostream>
using namespace std;
class counter
{
private:
int count ;
public:
void getdata (int x)
{
count=x ;
}
void showdata()
{
cout<<"count="<<count<<endl;
}
```

```cpp
counter operator ++()
{
counter temp;
temp.count=++count;
return temp;
}
counter operator --()
{
counter temp;
temp.count=--count;
return temp;
}

};

int main()
{
counter c1,c2;
c1.getdata(3) ;
c1.showdata() ;
cout<<"After overloading ++ operator"<<endl;
c2=++c1 ;
c1.showdata();
c2.showdata();
cout<<"After overloading -- operator"<<endl;
c2=--c1 ;
c1.showdata();
c2.showdata();
return 0;
}
```

14. **How operator overloading support polymorphism? Explain it by overloading '+'
   Operator to concatenate two strings.**

As we know Polymorphism means 'One name-multiple forms'.
let us consider an example, operator symbol '+' is used for arithmetic operation
between two numbers, however by overloading (means given additional job) it can be
used for difference purposes such as addition of   currency that has Rs and Paisa as its
attributes, addition of complex number that has real part and imaginary part as
attribute etc.
By overloading same operator '+' can be used for different purpose like concatenation
of strings too.

Here we illustrate the program that concatenates two strings by overloading '+' operator.

```cpp
#include<iostream>
#include<string.h>
using namespace std;
class stringc
{
private:
char str[50];
public:
stringc()
{ }
stringc(char s[])
{
strcpy(str,s);
}
void display()
{
cout<<str<<endl;
}
stringc operator +(stringc s2)
{
stringc s3;
strcpy(s3.str,str);
strcat(s3.str,s2.str);
return s3;
}
};
int main()
{
stringc s1("pokhara");
stringc s2("university");
stringc s3;
cout<<"s1=";
s1.display();
cout<<"s2=";
s2.display();
s3=s1+s2;
cout<<"s1+s2=";
s3.display();
return 0;
}
```

Output:
s1=pokhara
s2=university
s1+s2=pokharauniversity

In this way operator overloading supports polymorphism.

15. **We have a class name complex which have a data member real and imaginary .Default and parameterized constructor are there to initialize data member .WAP to overloading unary operator '++' where real and imaginary data member will be incremented and '+' operator to add two complex number.**

```cpp
#include<iostream>
using namespace std;
class Complex
{
    int real, imag;
  public:
    Complex()
      {
              real=0;
              imag=0;
      }
    Complex(int r, int i)
    {
      real=r;
      imag=i;
    }
    Complex operator++()
  {
      real++;
      imag++;
      return *this;
    }
    Complex operator +(Complex c2)
  {
      Complex temp;
      temp.real = real + c2.real;
      temp.imag = imag + c2.imag;
      return temp;
    }
```

```cpp
        void display()
        {
            cout << real << "+" << imag <<"i"<<endl;
        }
};


int main()

{
    Complex c1(10, 5);
    Complex c2(2, 4);
    Complex c3;
    ++c1;
    ++c2;
    c3= c1 + c2;
    c3.display();
    return 0;
}
```

16. **Can you have more than one constructor in a program? Write a program to find area of triangle (when sides are given) using the concept of overloaded constructor.**

Yes, it is possible to have more than one constructor in a program. In C++, constructor overloading is implemented by defining multiple constructors with different parameter lists. Each constructor has a unique signature that is determined by the number, types, and order of its parameters. When you create an object of the class, the compiler determines which constructor to call based on the arguments that you pass to it.

```cpp
#include <iostream>
#include <cmath>
using namespace std;
class Triangle {
    private:
        float a,b,c,s,area;

    public:
        // Default constructor
        Triangle()
    {
            a = 6;
            b = 8;
            c = 10;
    }
```

```cpp
        // Constructor with three arguments
        Triangle(float s1, float s2, float s3)
         {
             a = s1;
             b = s2;
             c= s3;
         }

        // Method to calculate area of triangle
        float calculate()
          {
             s=(a+b+c)/2;
             area=sqrt(s*(s-a)*(s-b)*(s-c));
             return area;
          }
};

int main()
 {
     Triangle t1(3, 4, 5);
     cout << "Area of triangle with sides 3, 4, 5 ="<<t1.calculate()<<endl;
     Triangle t2;
     cout << "Area of triangle with sides 6, 8, 10 = "<<t2.calculate()<<endl;
     return 0;
}
```

In this program, the Triangle class has two constructors - a default constructor with no arguments, and a constructor with three arguments for the lengths of the sides. The calcualte() method calculates the area of the triangle using the formula sqrt(s(s-a)(s-b)(s-c)), where s is the semiperimeter of the triangle, and a, b, and c are the lengths of the sides.

In the main() function, two Triangle objects are created using the two different constructors. The first triangle has sides 3, 4, and 5, and is created using the three-argument constructor. The second triangle has sides 6, 8, and 10, and is created using the default constructor. The output of the program is the areas of both triangles, which are calculated using the calculate() method.