UNITIII

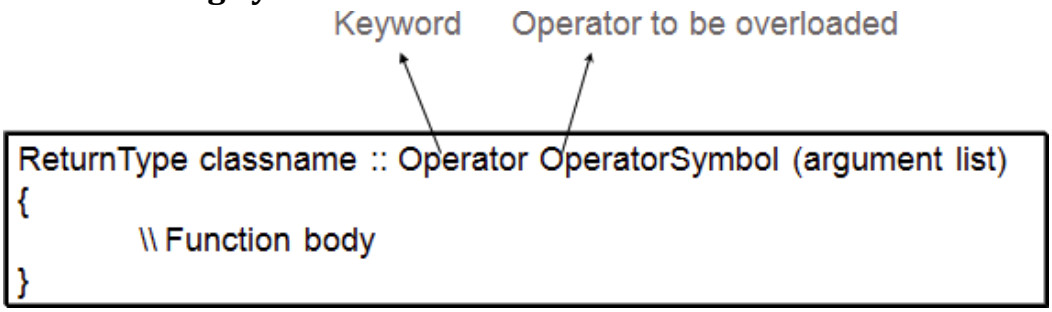
Operator Overloading and Type Conversion & Inheritance

The Keyword Operator, Overloading Unary Operator, Operator Return Type, Overloading Assignment Operator (=), Rules for Overloading Operators, Inheritance, Reusability, Types of Inheritance, Virtual Base Classes, Object as a Class Member, Abstract Classes, Advantages of Inheritance, Disadvantages of Inheritance.

Operator overloading: The single operator is used to exhibit the different behaviors at different instances is known as operator overloading.

Following is the list of operators, which cannot be overloaded:
scope operator - :: , sizeof ,member selector : "." , member pointer selector : "*.", ternary operator : " ? :"

Operator Overloading Syntax:



Common operators to be overload are

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Following is the list of operators, which cannot be overloaded –

::	.*	.	?:

Operator overloading can be done by using either member function or by using friend function. The differences are as given below.

Member Function	Friend Function
Number of parameters to be passed is reduced by one, as the calling object is implicitly supplied as an operand.	Number of parameters to be passed is more.
Unary operators takes no explicit parameters.	Unary operators takes one explicit parameter.
Binary operators takes only one explicit parameter.	Binary operators takes two explicit parameters.
Left-hand operand has to be the calling object.	Left-hand operand need not be an object of the class.
Writing Obj2 = Obj1 + 10 is allowed but Obj2 = 10 + Obj1 is not allowed	Writing either Obj2 = Obj + 10 or Obj2 = 10 + Obj1 is allowed.

// C++ program to add two complex numbers using operator overloading

```
#include<iostream.h>

class comp
{
            int r,i;
 public:
          void input()
           {
            cout<< "Enter real & imaginary parts : ";
            cin>>r>>i;
            }

        void display()
         {
           cout<<"Real = "<<r<<endl;
            cout<<"Imaginary = "<<i<<endl;
         }

         comp operator  +(comp c2)
         {
         comp c3;
         c3.r=r+c2.r;
         c3.i=i+c2.i;
         return c3;
         }
         };
```

```
int main()
{
comp b1,b2,b3;
clrscr();
b1.input ();
b2.input();
b3=b1+b2;
b3.display();
return 0;
}
```

Output :

```
Enter real & imaginary parts : 2 3
Enter real & imaginary parts :  4 5
Real = 6
Imaginary = 8 i
```

/* C++ program uses operator overloading to perform Addition, Subtraction, Multiplication and Division of two complex numbers. */

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<stdio.h>

class complex
{
int i,r;
public:
void read()
{
cout<<"\nEnter Real Part:";
cin>>r;
cout<<"Enter Imaginary Part:";
cin>>i;
}

void display()
{
cout<<r<<"+"<<i<<"i";
}

complex operator+(complex a2)
 {
complex a;
a.r=r+a2.r;
a.i=i+a2.i;
return a;
}
complex operator  - (complex  a2)
 {
complex a;
a.r=r-a2.r;
a.i=i-a2.i;
return a;
}
complex  operator *(complex a2)
complex a;
a.r=(r*a2.r)-(i*a2.i);
```

```
do {
cout<<"\n1.Addition\n";
cout<<"\n2.Substraction\n";
cout<<"\n3.Mulitplication\n
cout<<"\n4.Division\n";
cout<<"\n5.Exit\n";

cout<<"\nEnter the choice :";
cin>>ch;

cout<<"\nEnter First Complex
Number:";
a.read();
a.display();

cout<<"\nEnter Second Complex
Number:";
b.read();
b.display();

switch(ch)
{
case 1:
c=a+b;
c.display();
break;
case 2:
c=b-a;
c.display();
break;
case 3:
c=a*b;
c.display();
break;
case 4:
c=a/b;
c.display();
break;
}
```

```
a.i=(r*a2.i)+(i*a2.r);
return a;
}

complex operator/(complex a2)
{
complex a;
a.r=((r*a2.r)+(i*a2.i))/((a2.r*a2.r)+ (a2.i*a2.i));
a.i=((i*a2.r)-(r*a2.i))/((a2.r*a2.r)+ (a2.i*a2.i));
return a;
}
};

int main()
{
int ch;
clrscr();
complex a,b,c;
```

```
}while(ch!=5);
getch();
}
```

Output:

```
1.Addition
2. Substraction
3.Mulitplication
4.Division 5.Exit
Enter the choice : 1

Enter The First Complex Number:
2 3
Enter The Second Complex
Number: 4 5
6+8 i
Enter the choice : 5
```

//Write C++ Program to overload + operator to add two matrices.

```
#include<iostream.h>
#include<conio.h>

class matrix
{
        int m, n, x[30][30];
public:
matrix(int a, int b)
{
m=a; n=b;
}
matrix()
{

}
void get();
void put();
matrix operator +(matrix);
};
```

```
matrix matrix::operator +(matrix b)
{
matrix c(m,n);
for(int i=0; i<m; i++)
for(int j=0; j<n; j++)
c.x[i][j]= x[i][j] + b.x[i][j];
return c;
}

int main( )
{
int m,n;
clrscr();
cout<<"\n Enter the size of the Matrix";
cin>>m>>n;
matrix a(m,n) , b(m,n) , c;
a.get();
b.get();
c= a+b;
c.put();
 return 0;
```

```
void matrix:: get()
{
cout<<"\n Enter values into the matrix";
for(int i=0; i<m; i++)
for(int j=0; j<n;j++)
cin>>x[i][j];
}

void matrix:: put()
{
cout<<"\n Sum of the matrix is :\n";
for(int i=0; i<m; i++)
{
for(int j=0; j<n;j++)
cout<<x[i][j]<<"\t";
cout<<endl;
}
}
```

```
}
```

Output:

```
Enter the size of the Matrix 3 3 Enter values into the matrix:
1 2 3
4 5 6
3 4 5

Enter values into the matrix:
1 1 1
2 2 2
3 3 3

Sum of the matrix is :
2 3 4
6 7 8
6 7 8
```

//C++ program for unary increment (++) and decrement (--) operator overloading.

```
#include<iostream.h>
class NUM
{
private:
int n;
public:
//function to get number
void getNum(int x)
{
n=x;
}

//function to display number
void dispNum(void)
{
cout << "value of n is: " << n;
}

//unary ++ operator overloading
void operator ++ (void)
{
++n;
}
```

```
int main()
{
NUM num;
num.getNum(10);

++num;
cout << "After increment - ";
num.dispNum();
cout << endl;

--num;
cout << "After decrement - ";
num.dispNum();
cout << endl;
return 0;
}
```

Output:

```
After increment - value of n is: 11

After decrement - value of n is: 10
```

}
//unary -- operator overloading
void operator -- (void)
{
--n;
}
};	

Inheritance:

C++ strongly supports the concept of **reusability**. The C++ classes can be reused in several ways. 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. The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as the base class and the new one is called the derived class or subclass. A derived class includes all features of the generic base class and then adds qualities specific to the derived class.

Inheritance is the process of creating new classes from the existing class or classes. Existing class is known as Base class. New class is known as Derived class. A class derivation list names one or more base classes and has the form:

Syntax : class derived-class: access-specifier base-class

Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

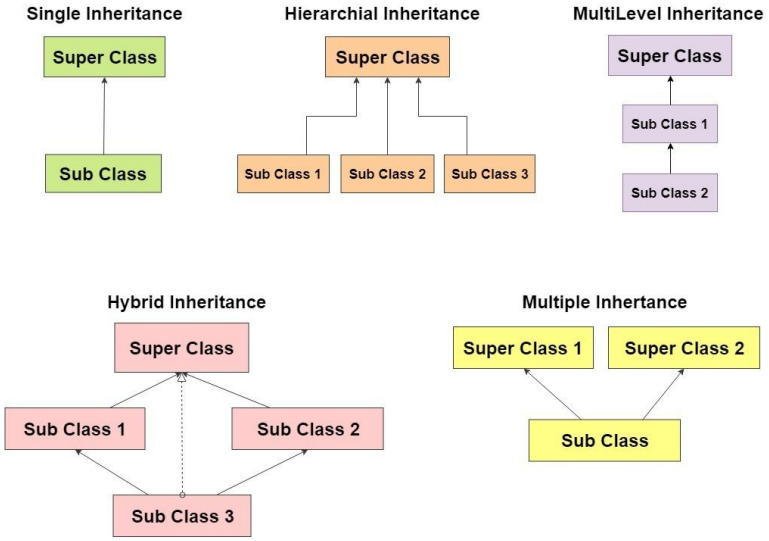
Access Control and Inheritance:

A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class. A derived class can access all the non-private members of its base class. Thus base- class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

Different access types according to who can access them in the following way:

Access Specifier	Private	Protected	Public
In Same Class	YES	YES	YES
In Derived Class	NO	YES	YES
In Outside Class	NO	NO	YES

Types of inheritance:



Single Inheritance - If a class is derived from a single base class, it is called as single inheritance.

Multiple Inheritance - If a class is derived from more than one base class, it is known as multiple inheritance.

Multilevel Inheritance - The classes can also be derived from the classes that are already derived. This type of inheritance is called multilevel inheritance.

Hierarchical Inheritance - If a number of classes are derived from a single base class, it is called as hierarchical inheritance.

Hybrid Inheritance - Hybrid Inheritance is a method where one or more types of inheritance are combined together and used.

```
class A
{
public:
int x;
protected:
int y;
private:
int z;
};
class B : public A
{
// x is public
// y is protected
// z is not accessible from B
};
```

```
class C : protected A
{
// x is protected
// y is protected
// z is not accessible from C
};

class D : private A // 'private' is default for
 {                                  classes
// x is private
// y is private
// z is not accessible from D
};
```

//C++ Program for single inheritance

```
#include<iostream.h>
#include<conio.h>
class B
{
public:
int a,b;
void get();
void show();
};

class D:public B
{
int c;
public:
void mul();
void disp();
};

void B::get()
{
a=5;b=10;
}

void B::show()
{
cout<<"\t"<<a;
}

void D::mul()
{
c=b*a;
}

void D::disp()
{
cout<<"\n"<<a<<"\t"<<b<<"\ta*b:"<<c;
}

int main()
{
D d;
d.get();
d.mul();
d.show();
d.disp();
d.b=20;
d.mul();
d.disp();
getch();
return 0;
}
```

Output:

```
5
5 10 50
5 20 100
```

Multiple Inheritance:

```
#include <iostream.h>
// Base class Shape
class Shape
{
public:
void setWidth(int w)
{ width = w;
}

void setHeight(int h)
{ height = h;

// Derived class
class Rectangle :public shape, public PaintCost
{
public:
int getArea()
{
return (width * height);
}
};
```

```
}
protected:
int width; int height;
};
// Base class PaintCost
class PaintCost
{
public:
int getCost(int area)
{
return area * 70;
}
};
```

```
int main(void)
{
Rectangle Rect;
int area;
Rect.setWidth(5);
Rect.setHeight(7);
area = Rect.getArea();
// Print the area of the object.
cout << "Total area: " <<area<< endl;
// Print the total cost of painting
cout << "Total paint cost: $" <<
Rect.getCost(area) << endl;
return 0;
}
```

Output:

Total area: 35

Total paint cost 2450

Multilevel Inheritance:

```
#include<iostream.h>
class Student
{ protected:
int marks;
public:
void accept(){
cout<<" Enter marks"; cin>>marks;
}
};
class Test : public Student
{ protected:
int p=0;
public:
void check()
```

```
class Result :public Test{ public:
void print(){ if(p==1)
cout<<"\n You have passed";
else
cout<<"\n You have not passed";
}
};

int main()
{ Result r;
r.accept();
r.check();
r.print();
return 0;
```

{ if(marks>60)
{
p=1;
}
}
};
	}
	
	Output:-
	Enter marks 70
	You have passed

Heirarchical Inheritance:

/*C++ program to demonstrate example of hierarchical inheritance to get square and cube of a number.*/

#include <iostream.h>
class Number
{ private:
int num;
public:
void getNumber(void)
{ cout << "Enter an integer number: ";
cin>> num;
}
//to return num
int returnNumber(void)
{ return num;
}
};
//Base Class 1, to calculate square of a //number
class Square:public Number
{ public:
int getSquare(void)
{ int num,sqr;
num=returnNumber(); //get number from //class Number
sqr=num*num;
return sqr;
	//Base Class 2, to calculate cube of a //number
	class Cube:public Number
	{ public:
	int getCube(void)
	{ int num,cube;
	num=returnNumber();
	//get number from class Number cube=num*num*num;
	return cube;
	}
	};
	
	int main()
	{ Square objS;
	Cube objC;
	int sqr,cube;
	objS.getNumber();
	sqr =objS.getSquare();
	cout << "Square of "<<
	objS.returnNumber() << " is: " << sqr; objC.getNumber();
	cube=objC.getCube();
	cout << "Cube of "<< objS.returnNumber() << " is: " << cube;

} };	return 0; } **Output:** Enter an integer number: 10 Square of 10 is: 100 Enter an integer number: 20 Cube of 10 is: 8000

Hybrid Inheritance:
// C++ program to implement Hybrid Inheritance

#include<iostream.h>
#include<conio.h>
class arithmetic
{
protected:
int num1, num2;
public:
void getdata(){
cout<<"For Addition:";
cout<<"\nEnter the first number: ";
cin>>num1;
cout<<"\nEnter the second number: ";
cin>>num2;
}
};
 	
class plus: public arithmetic{
protected:
int sum;
public:
void add(){
sum=num1+num2;
}
};
	class result: public plus, public minus{
	public:
	void display(){
	cout<<"\nSum of "<<num1<<" and "<<num2<<"= "<<sum;
	cout<<"\nDifference of "<<n1<<" and "<<n2<<"= "<<diff;
	}
	};
	
	int main(){
	clrscr();
	result z;
	z.getdata();
	z.add();
	z.sub();
	z.display();
	getch();
	return 0;
	}
	
	Output:
	For Addition:
	Enter the first number: 10

class minus{
protected:
int n1,n2,diff;
public:
void sub(){
cout<<"\nFor Subtraction:";
cout<<"\nEnter the first number: ";
cin>>n1;
cout<<"\nEnter the second number: ";
cin>>n2;
diff=n1-n2;
}
};
	Enter the sencond number: 5
	For Subtraction:
	Enter the first number: 15
	Enter the sencond number: 5
	Sum of 10 and 5 =15
	Difference of 15 and 5 is 10

Inheritance Advantages and Disadvantages:

Advantages :

1. Inheritance promotes reusability. When a class inherits or derives another class, it can access all the functionality of inherited class.

2. Reusability enhanced reliability. The base class code will be already tested and debugged.

3. As the existing code is reused, it leads to less development and maintenance costs.

4. Inheritance makes the sub classes follow a standard interface.

5. Inheritance helps to reduce code redundancy and supports code extensibility.

6. Inheritance facilitates creation of class libraries.

Disadvantages:-

1. Inherited functions work slower than normal function as there is indirection.

2. Improper use of inheritance may lead to wrong solutions.

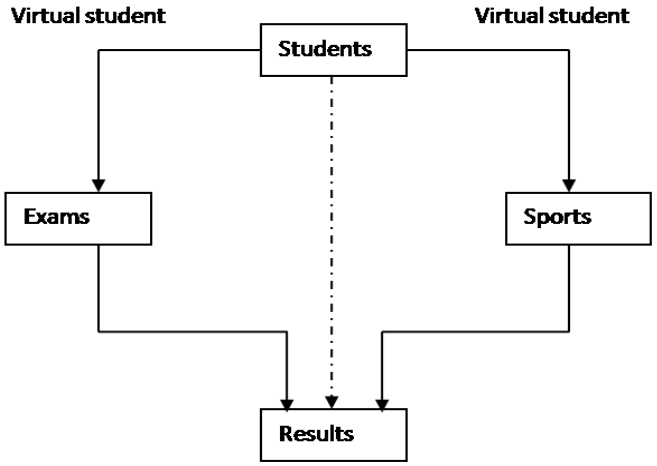
3. Often, data members in the base class are left unused which may lead to memory wastage.

4. Inheritance increases the coupling between base class and derived class. A change in base class will affect all the child classes.

Virtual base class:
An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.

C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.

What is Virtual base class? Explain its uses.
When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class' name with the word virtual.



Simple Program for Virtual Base Class:

```
#include<iostream.h>
#include<conio.h>
class student {
int rno;
public:
void getnumber() {
cout << "Enter Roll No:";
cin>>rno;
}
void putnumber() {
cout << "\nRoll No:" << rno << "\n";
}
class result : public test, public sports
{  int total;
public:
void display() {
total = part1 + part2 + score;
putnumber();
putmarks();
putscore();
cout << "\n\tTotal Score:" << total;
}
};
int main() {
```

```
class test : virtual public student
{ public:
int part1, part2;
void getmarks() {
cout << "Enter Marks\n"; cout << "Part1:";
cin>>part1;
cout << "Part2:"; cin>>part2;
}
void putmarks() {
cout << "\tMarks Obtained\n";
cout << "\n\tPart1:" << part1;
cout << "\n\tPart2:" << part2;
}
};
class sports : public virtual student
 { public:
int score;
void getscore() {
cout << "Enter Sports Score:";
cin>>score;
}
void putscore() {
cout <<"\n\tSports Score is:" << score;
}
};
```

```
result obj;
clrscr();
obj.getnumber();
obj.getmarks();
obj.getscore();
obj.display();
getch();
return 0;
}
```

Output :

```
Enter Roll No: 200
Enter Marks Part1: 90
Part2: 80
Enter Sports Score: 80


Roll No: 200
Marks Obtained Part1: 90
Part2: 80
Sports Score is: 80
Total Score is: 250
```

Virtual functions:

A virtual function is a member function that is declared within a base class and redefined by a derived class. To create virtual function, precede the function's declaration in the base class with the keyword virtual. When a class containing virtual function is inherited, the derived class redefines the virtual function to suit its own needs.

Base class pointer can point to derived class object. In this case, using base class pointer if we call some function which is in both classes, then base class function is invoked. But if we want to invoke derived class function using base class pointer, it can be achieved

by defining the function as virtual in base class, this is how virtual functions support runtime polymorphism.

- Consider the following program code:

```
Class A {
int a;
public:
A() {
a = 1;
}
virtual void show() {
cout <<a;
}
};

Class B: public A {
int b;
public:
B() {
b = 2;
}
virtual void show() {
cout <<b;
}
};

int main() {
A *pA;
B oB;
A oA;
pA = &oB;
pA→show(); // Derived version will be called
pA = &oA;
pA→show(); // Base version will be called
return 0;
}
```

Output:

2

1

Abstract class:

The C++ interfaces are implemented using **abstract classes** .A class is made abstract by declaring at least one of its functions as **pure virtual** function. A pure virtual function is specified by placing "= 0" in its declaration as follows:

```
class Box
{
public:
virtual double getVolume() = 0;   // pure virtual function
private:
double length; // Length of a box
double  breadth; // Breadth of a box
double height; // Height of a box
};
```

The purpose of an **abstract class** (often referred to as an ABC) is to provide an appropriate base class from which other classes can inherit. Abstract classes cannot be

used to instantiate objects and serves only as an **interface**. Attempting to instantiate an object of an abstract class causes a Compilation error.

Thus, if a subclass of an ABC needs to be instantiated, it has to implement each of the virtual functions, which means that it supports the interface declared by the ABC. Failure to override a pure virtual function in a derived class, then attempting to instantiate objects of that class, is a compilation error.

Classes that can be used to instantiate objects are called **concrete classes**.

Abstract Class Example:
Consider the following example where parent class provides an interface to the base class to implement a function called **getArea()**:

```
#include <iostream.h>
// Base class
 class Shape
{ public:
// pure virtual function providing
interface framework.
virtual int getArea() = 0;
void setWidth(int w)
{
width = w;
}
void setHeight(int h)
{
height = h;
}
protected: int width;
int height;
};
// Derived classes
class Rectangle:  public Shape
{ public:
int getArea()
{
return (width * height);
}
class Triangle:  public Shape
{
public:
int getArea()
{
return (width * height)/2;
}
};

int main()
{ Rectangle Rect;
Triangle Tri;
Rect.setWidth(5);
Rect.setHeight(7);
// Print the area of the object.
cout << "Total Rectangle area: " <<
Rect.getArea() << endl;
Tri.setWidth(5);
Tri.setHeight(7);
// Print the area of the object.
cout << "Total Triangle area: " <<
Tri.getArea() << endl;
return 0;
}
```

} };	**Output:** Total Rectangle area: 35 Total Triangle area: 17

Assignment Questions

1. What are different types of inheritance supported by C++? Give an example for each.
2. What is inheritance? Present the advantages and disadvantages of inheritance.
3. Write about operator overloading in C++ with an example.
4. Write C++ Program to overload + operator to add two matrices.
5. Explain about Virtual Base classes and Virtual Functions in C++.