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]**

```
#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.]

```
#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]**

```
#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;
}
}
```

```
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.

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

```
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.

```
#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;
}
```

```
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]**

```
#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;
}
```

```
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);
}
};
```

```
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]**

```
#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;
}
```

```
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;
}
};
```

```
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]

```
#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;
}
};
```

```
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.

```
#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;
}
};
```

```
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

```
#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;
    }
```

```
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)

```
#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;
 }
```

```
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.

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

```
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.

```
#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.

```
#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;
      }
```

```
    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.

```
#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;
        }
```

```
    // 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.