SVKM'S NMIMS Deemed-to-be UNIVERSITY

SVKM's NMIMS

Mukesh Patel School of Technology Management and Engineering, Vile Parle, Mumbai- 400056

Department of Mechatronics

OOPS LAB FILE

Name: Jay Menon
Roll No: H041
BTECH MECHATRONICS 3RD YEAR

H2 BATCH

LIST OF EXPERIMENTS

SR. NO.	NAME OF THE EXPERIMENTS
1	TOWERS OF HANOI (RECURSION)
2	CLASS (RECTANGLE) CONSTRUCTOR OVERLOADING
3	CARDS GAME
4	FUNCTION OVERLOADING (VECTORS)
5	FRIEND FUNCTION
6	OPERATOR OVERLOADING(CN)
7	INHERITANCE
8	PURE VIRTUAL FUNCTION
9	2D ARRAY INTIALIAZATION DYNAMIC DETERMINANT
10	CLASS TEMPLATE FOR SORTING (BUBBLE,INSERTION)

EXP - 1

AIM: To learn the concept of recursion using Towers of Hanoi problem

PROGRAM CODE:

```
#include<iostream>
using namespace std;
void t_o_h(int n, char n1, char n2, char n3)
{
if(n==1)
{
cout<<"\nShift top disk from tower"<<n1<<" to tower"<<n2;
return;
}
t_o_h (n-1,n1,n3,n2);
cout<<"\nShift top disk from tower"<<t1<<" to tower"<<t2;
t_o_h (n-1,n3,n2,n1);
}
int main()
{
int disk;
cout<<"Enter the number of disks:"; cin>>disk;
if(disk<1){
cout<<"There are no disks to shift";}
else
cout<<"There are "<<disk<<"disks in tower 1\n";
t_o_h (disk, '1','2','3');

cout<<"\n\n"<<disk<<"disks in tower 1 are shifted to tower 2";


return 0;

}
```

CONCLUSION: In this experiment we learnt how to call a function within the same function (i.e Recursive function) and also implemented it in the above program.

EXP 2

Aim: WAP to find the centroid of the rectangle from a given point.

PROGRAM CODE:

```
#include <iostream>

using namespace std;

class point

{

   private:

   double x;

   double y;

   public:

   point()

   {

      x=y=0;

   }

   point(double x, double y)

   {

      this->x=x;

      this->y=y;

   }

   void display()

   {

      cout<<"("<<this->x<<","<<this->y<<")"<<endl;

   }
```

```
    void set_x(double x)
    {
        this->x=x;
    }
    void set_y(double y)
    {
        this->y=y;
    }
    double get_x()
    {
        return x;
    }
    double get_y()
    {
        return y;
    }
};

class rectangle
{
    private:
    point tl;
    double len,wid;

    public:
    rectangle()
    {
```

```
    tl.set_x(0);

    tl.set_y(0);

    len=0;

    wid=0;

}

rectangle(point p,double len,double wid)

{

    tl.set_x(p.get_x());

    tl.set_y(p.get_y());

    this->len=len;

    this->wid=wid;

}

rectangle(double tlx, double tly, double len, double wid)

{

    tl.set_x(tlx);

    tl.set_y(tly);

    this->len=len;

    this->wid=wid;

}

point calc_centroid()

{

    point p;

    p.set_x(tl.get_x()+len/2);

    p.set_y(tl.get_y()+wid/2);

    return p;

}

point calc_diagonally_opposite()
```

```
    {
        point p;
        p.set_x(tl.get_x()+len);
        p.set_y(tl.get_y()+wid);
        return p;
    }
};
int main()
{
  point p(20,50);
  rectangle r(p,30,40);
  point ctd=r.calc_centroid();
  ctd.display();
  point br=r.calc_diagonally_opposite();
  br.display();
}
```

CONCLUSION: In this experiment we learnt about the different types of constructors, concept of constructor overloading and how to call the constructors in the main. The purpose of constructor is to initialize the object of a class while the purpose of a method.

EXP 3

Aim: WAP to create a cards game in which a random card will be generated. If the user's card matches with the random card it is a draw else the user wins or looses according to the no and the colour of the card.

PROGRAM CODE:

```
#include <random>

#include <iostream>

using namespace std;

//char suit[]={'S','H','C','D'};

class cards

{

  private:

  int suit;

  int rank;

  public:

  cards()

  {

    suit=rand()%4;

    rank=(rand()%13+1);

  }

  cards(int c, int r)

  {

    suit=c;

    rank=r;

  }
```

```
    void display()
    {
        cout<<rank<<" "<<suit<<endl;
    }
    void play(cards o)
    {
        if (this->suit!=o.suit)
        {
            cout<<"draw"<<endl;
        }
        else if (this->rank>o.rank)
        {
            cout<<"Player 1 Wins"<<endl;
        }
        else if(this->rank<o.rank)
        {
            cout<<"Player 2 wins"<<endl    ;
        }
        else
        {
            cout<<"draw"<<endl;
        }
    }
};
int main()
{
    cards c1(1,5),c2(1,5);
    c1.display();

    c2.display();

    c1.play(c2);

    return 0;

}
```

CONCLUSION: In this experiment we learnt how to apply the concepts of c++ to build a cards game. Also we learnt how to generate a random no.

EXP 4

Aim: To find the dot and cross product of vectors using function overloading.

PROGRAM CODE:

```
#include <iostream>
#include<cmath>
using namespace std;
class vector
{
   double i,j,k;
   public:
   vector()
   {
      i=j=k=0;
      }
   vector(double x,double y,double z)
   {
   i=x;
   j=y;
   k=z;
   }
   vector add(vector v1,vector v2)
   {
      vector v;
```

```
                    v.i=v1.i+v2.i;

                    v.j=v1.j+v2.j;

                    v.k=v1.k+v2.k;

                    return v;

                }

                double dot(vector v1,vector v2, double theta)

                {

                    double d;

                    double a,b;

                    a=(v1.i)*(v1.i)+(v1.j)*(v1.j)+(v1.k)*(v1.k);

                    b=(v2.i)*(v2.i)+(v2.j)*(v2.j)+(v2.k)*(v2.k);

                    d=sqrt(a)*sqrt(b)*cos(theta);

                    cout<<"dot product="<<d<<"\n";

                    return d;


                }

                vector cross(vector v1,vector v2)

                {

                    vector c;

                    cout<<"\n"<<((v1.j*v2.k)-(v1.k*v2.j))<<"i -"<<((v1.i*v2.k)-(v1.k*v2.i))<<"j +"<<((v1.i*v2.j)-
        (v2.i*v1.j))<<"k ";

                }

                void display1()

                {

                    cout<<"i="<<i <<" j="<<j<<" k="<<k;

                }

            };
```

```
int main()

{

   vector v5;

   vector v1(4,5,6),v2(7,10,20);

   vector v3=v1.add(v1,v2);

      double v4=v1.dot(v1,v2,3.14);

   v3.display1();

   v5.cross(v1,v2);


}
```

CONCLUSION: In this experiment we learnt the concept of function overloading. Function overloading is a feature in C++ where two or more functions can have the same name but different parameters. Function overloading can be considered as an example of polymorphism feature in C++. We also learnt how to call the functions with different argument list in the main.

EXP 5

Aim: WAP for implementing class as global friend, class as a friend of another class and function in class as friend

PROGRAM CODE:

Global Friend:-

```
#include <iostream>

#include <string>

using namespace std;


class B;

class A

{

   int a;


   public:

   A(int x)

   {

      a=x;

   }


   friend int add(A,B);


};
```

```
class B
{
  int b;
  public:
  B(int y)
    {
      b=y;
    }

    friend  int add(A,B);

};



int add (A aref, B bref)
{
    return (aref.a+bref.b);
}



int main()
{
    A a1(10);
    B b1(20);
    cout<<add(a1,b1);
    return 0;
}
```

Class as friend:-

```
#include <iostream>

#include <string>

using namespace std;


class B;

class A

{

   int a;


   public:

   A()

   {

      a=10;

   }


   friend class B;


};


class B

{

  int b;

  public:

  B()
   {
     b=20;
   }

   int add (A aref)
   {
      return (b+aref.a);
   }

};
int main()
{
   A a1;
   B b1;
   cout<<b1.add(a1);
   return 0;
}
```

Function in class as a friend:-

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

class B;
class A
{
  int a;
```

```
    public:
    A(int x)
    {
        a=x;
    }

    int add(B);

};



class B
{
  int b;
  public:
 B(int y)
    {
      b=y;
    }

    friend  int A:: add(B);

};
```

```
int A::add (B ref)
{
   return (ref.b+a);
}
int main()
{
   A a1(10);
   B b1(20);
   cout<<a1.add(b1);
   return 0;
}
```

CONCLUSION: In this experiment we understood the concept of friend functions and how to use it in different ways ie. Global friend, class as a friend of another class and function in class as friend.
A friend function in C++ is a function that is preceded by the keyword “friend”. When the function is declared as a friend, then it can access the private and protected data members of the class.

EXP 6

Aim: WAP to perform of Addition of two Complex Numbers using operator overloading.

PROGRAM CODE:

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

class Complex
 {
private:
int real, imag;
public:
Complex(int r = 0, int i =0)
 {
real = r;
 imag = i;
}
void print()
 {
 cout << real << " + i" << imag << endl;
 }
friend Complex operator + (Complex const &, Complex const &);
};
Complex operator + (Complex const &c1, Complex const &c2)
{
```

```
return Complex(c1.real + c2.real, c1.imag + c2.imag);

}

int main()

{

Complex c1(10, 5), c2(2, 4);

Complex c3 = c1 + c2; // An example call to "operator+"

c3.print();

return 0;

}
```

CONCLUSION: In this experiment we understood the concept of operator overloading. Operator overloading is used to give special meaning to most of the operators available in C++. It is used to perform the operation on the user-defined data type. In this program we overloaded + operator to add two complex nos. We also learnt its syntax and how to call these functions in main.

EXP 7

Aim: WAP to read input from 2 classes student and sports and display total marks and average along with student details using inheritance.

PROGRAM CODE:

```
#include<iostream.h>

using namespace std;


class student

{

protected:

   int rno, m1, m2;

public:

      void get()

 {

       cout << "Enter the Roll no :";

       cin>>rno;

       cout << "Enter the two marks   :";

       cin >> m1>>m2;

   }

};


class sports

 {

protected:
    int sm;
public:
    void getsm()
   {
        cout << "\nEnter the sports mark :";
        cin>>sm;


    }
};


class statement : public student, public sports
 {
    int tot, avg;
public:
    void display()
 {
        tot = (m1 + m2 + sm);
        avg = tot / 3;
        cout << "\n\n\tRoll No    : " << rno << "\n\tTotal      : " << tot;
        cout << "\n\tAverage    : " << avg;
    }
};
void main()
{   statement obj;
    obj.get();
    obj.getsm();
    obj.display();
```

```
}
```

CONCLUSION: In this experiment we understood the concept of inheritance. In the above program we have implemented multiple inheritance. In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object. We learnt how to inherit a base class and call the functions accordingly. Inheritance provides the feature of reusability.

EXP 8

Aim: WAP to implement pure virtual function

PROGRAM CODE:

```
#include <iostream>
#include <string>
using namespace std;
class Shape
{
   public:
   virtual double calcArea()=0;
   virtual double calcPeri()=0;
   virtual int noOfSides()=0;
   void display()
   {
      cout<<"Area:\t"<<calcArea()<<endl;
      cout<<"Peri:\t"<<calcPeri()<<endl;
      cout<<"Sides:\t"<<noOfSides()<<endl;
   }
};
class Quadrilateral: public Shape
{
   public:
   int noOfSides()
   {
      return 4;
```

```
    }
};
class Rectangle : public Quadrilateral
{
    protected:
    int len,wid;
    public:
    Rectangle()
    {
  len=wid=0;
  }
    Rectangle(int x, int y)
    {
        len=x;wid=y;
    }
    double calcArea()
    { return len*wid;}
    double calcPeri()
    { return 2*(len+wid);}
};
```

```
int main()

{

   Rectangle *r1=new Rectangle(4,5);

   r1->display();

}
```

CONCLUSION: In this experiment we understood the concept of pure virtual function. A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have implementation, we only declare it by assigning 0 in declaration. We also understood the difference between the virtual and pure virtual fn .

EXP 9

Aim: WAP to implement 2D array initialization for dynamic determinant

PROGRAM CODE:

```
#include <iostream>


int main()

{

  int dim1, dim2;

  std::cin >> dim1 >> dim2;


double* array_data = new double[dim1*dim2];

  double** array = new double*[dim1];

  for (int i = 0; i < dim1; ++i)

    array[i] = array_data + dim2*i;


  array[0][0] = 3.5;

  std::cout << array[0][0] << std::endl;

  delete[] array;

  delete[] array_data;


  return 0;

}
```

CONCLUSION: In this experiment we learnt how to dynamically initialize variables at run time during compilation process

EXP 10

Aim: WAP to implement class template for sorting (bubble,insertion)

PROGRAM CODE:

Bubble Sort

```
#include<conio.h>

using namespace std;

template<class bubble>

void bubble(bubble a[], int n)

{

   int i, j;

   for(i=0;i<n-1;i++)

   {

      for(j=i+1;j<n;j++)

      {

         if(a[i]>a[j])

         {

            bubble element;

            element = a[i];

            a[i] = a[j];

            a[j] = element;

         }

      }

   }

}
```

```
void main()

{

   int a[6]={1,2,3,4,4,3};

   char b[4]={'s','b','d','e'};

   clrscr();

   bubble(a,6);

   cout<<"\nSorted Order Integers: ";

   for(int i=0;i<6;i++)

      cout<<a[i]<<"\t";

   bubble(b,4);


   cout<<"\nSorted Order Characters: ";

   for(int j=0;j<4;j++)

      cout<<b[j]<<"\t";


}
```

CONCLUSION: In this experiment we learnt the concept of templates and implemented it for sorting purpose.