# C++ exercises Please complete the below half-done programs

```
/*
   Program to illustrate default arguments in C++
*/
# include<iostream>
using namespace std;

void dummy (int , int);
void dummy (int   ix=10 ,  int=200,   float=777);
main()
{
         int iNum1 = 5,iNum2 = 10, iNum3 = 100;
         float fNum1 = 0.999999f,fNum2 = 100.99999f;
         double  dNum1 = 9999.9999,dNum2 = 68795.55555;

         dummy(iNum1);
         dummy(iNum1,iNum2);
         dummy(iNum1,iNum2,iNum3);

}
void dummy(int   ix, int   iy, float  iz)
{ cout <<ix <<"   "<<iy<<"   "<<  iz <<endl; }

void  dummy(int  ix , int   iy)
{ cout <<ix <<"   "<<iy<<endl ;}

/* Example Program to demonstrate function overloading */

#include<iostream>
using namespace std;

inline int    max (int  ,int );
inline float max (float , float);
inline double max (double , double );

int main()
{
  int ival1,ival2;
  float fval1,fval2;
  double dval1,dval2;

  cout <<"Enter integers vals:" <<endl;
  cin >> ival1>>ival2;

  cout <<"Enter float vals:" <<endl;
  cin >> fval1>>fval2;

  cout <<"Enter double vals:" <<endl;
  cin >> dval1>>dval2;
```

```
   cout << max(ival1,ival2)<<endl;
   cout <<max (fval1,fval2)<<endl;
   cout <<fixed<<max (dval1,dval2)<<endl;
}
int    max(int x ,int y){
     return(x>y?x:y);
}
float  max (float x, float y){
     return(x>y?x:y);
}
double max (double x, double y){
     return(x>y?x:y);
}
```

```
/* Complete the below program. This exercise is to demonstrate constructure setter, getter methods
*/


#include<iostream>
using namespace std;
const float  pi=3.14156;

class Circle{
    private:
       float radius, circumference;
       float area, diameter;
     public:
        Circle(){

        }
        Circle(float   r){

        }
        void set_radius(float  r){
           radius=r;
        }
        void fn_area(void){

        }
        void fn_circumference(void){

        }
        void fn_diameter(void){
          diameter = 2*radius;
        }

        float get_radius(){
          return radius;
        }
        float get_diameter(){
         return diameter;
        }
        float get_area(){
          return area;
        }
       float get_circumference(){
          return circumference;
       }
};
int main()
{
   Circle  c1(10.0);
   //c1.set_radius(25.5);
   c1.fn_area();
   c1.fn_circumference();
   c1.fn_diameter();

   cout<<"radius is "<<c1.get_radius()<<endl;
   cout<<"diameter is "<<c1.get_diameter()<<endl;
   cout<<"Area is "<<c1.get_area()<<endl;
   cout<<"Circumference is "<<c1.get_circumference()<<endl;
 }
```

```
 /* Complete the below program. This exercise is to demonstrate constructure setter, getter
methods */

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

class RTriangle{
    private:
       float   base ;
       float   height;
       float   area;
       float   perimeter;
    public:
       static float  pi;
       RTriangle(){
          cout <<"Indefault constructor\n" ;
       }
       ~RTriangle(){cout<<"In destructor\n"; }
```

```
        RTriangle(float b, float h){

        }
        void setBase(float   b){
            base =b;
         }
        void setHeight(float  h){
             height =h;
        }
        float getBase(void){
            return base;
         }
        float getHeight(void){
             return height;
        }
        float getArea(void){
             return(area);
         }
        float getPerimeter(void){
             return (perimeter);
        }
        float  fnArea(void);
        float  fnPerimeter(void);
};

float   RTriangle::pi=3.14156f;

float  RTriangle::fnArea(void){

}
float  RTriangle::fnPerimeter(void){

}
int main()
{

    RTriangle   *ptr=NULL;

    ptr = new  RTriangle(123.24,65.7213);

    cout <<"Area is : "<<ptr->fnArea()<<endl;
    cout <<"Perimeter is : "<<ptr->fnPerimeter()<<endl;

    cout <<"Pi is : "<<RTriangle::pi<<endl;
    delete  ptr;
}
```

```
/* Complete the below program. This exercise is to demonstrate constructure setter, getter methods
*/


#include<iostream>
using namespace std;

class Box{
   float  height;
   float  width;
   float  length;
   float  volume;
   float surfacearea;

   public:
     Box(){}
     Box(float h, float w, float l){

     }
     Box(Box & b){

     }
     void setHeight(float h){
        height = h;
     }
     void setLength(float l){

     }
     void setWidth(float w){

     }
     double fnVolume(){

     }
     double  fnsurfacearea(){

     }
     float  getVolume(){

     }
     float getSurfaceArea(){

     }
     void Print(){
        cout <<"volume is : "<<volume<<endl;
        cout <<"surfacearea is:"<<surfacearea<<endl;
     }
     float getHeight(void){

     }
     float getLength(void){
       }
       float getWidth(void){
          return width;
       }

};
int main()
{
   Box   obj(12.6,21.3,41.5);
   obj.fnVolume();
   obj.fnsurfacearea();
   obj.Print();

   cout <<hex<<obj.getHeight()<<endl;
   cout <<obj.getLength()<<endl;
   cout <<obj.getVolume()<<endl;
}
```

```
/* Complete the below program to illustrate private method*/

class Book{
   public:
   class chapter {
      char   chap_name[25];
      int number;
      int pages;
      public:
        chapter(){}
        chapter(char *ch,int  n,int  p){

        }
        void display(){
           cout << chap_name<<endl;
           cout <<number <endl;
           cout << pages<<endl;
        }
   };
   Book(char *bn,int amt,char  *name,int n,int p){

      }
      void display(){
        cout << name<<endl;
        cout <<price<endl;
        cout << "******chapter info ****" <<endl;
```

```
          cout << pages<<endl;
       }
   private:
      chapter   sub(name,n,p);
      char   name[25];
      int price ;
};

/* Complete the below program.
    This program is to illustrate dynamic memory allocation and copy constructor
  */

#include<iostream>
using namespace std;

class Array {
   int *data;
   int size;
   int capacity;
   public:
       Array(){
          data=nullptr;
          size=capacity=0;
       }
       Array(int sz){

       }
       Array(Array & ref ){

       }
       ~Array(){
          delete [] data;
          data=nullptr;
       }
       void addbeg(int val){

       }
       void append(int val){

       }
       void insert(int pos,int val){

          }
          data[pos]=val;
       }
       int Size(){ return size;}
       int deletebeg(){

          return temp;
       }
```

```
        int deleteend(){
           --size;
           return data[size];
        }
        int delete_pos(int pos){
           int  i, temp;

           return temp;

        }
        void Print(){
           int  i, temp;
           for(i=0; i < size;i++)
              cout <<data[i]<<endl;
        }
        int getVal(int pos){
           return(data[pos-1]);
        }
};


int main()
{
   Array   array(100);
   array.addbeg(10);
   array.append(20);
   array.insert(2,30);
   array.addbeg(100);
   array.append(200);
   array.insert(5,300);

   cout <<"Size of Array is"<<array.Size();
   cout <<"Data at "<<3<<array.getVal(3)<<endl;

   array.Print();
   cout <<"**********************************\n";
   cout <<array.deletebeg()<<endl;
   cout <<array.deleteend()<<endl;
   cout <<array.delete_pos(2)<<endl;
   cout <<"**********************************\n";
   array.Print();
   cout <<"Size of Array is"<<array.Size()<<endl;


   Array sec_obj(array);

   sec_obj.addbeg(10);
   sec_obj.append(20);
   sec_obj.insert(2,30);
   sec_obj.addbeg(100);
   sec_obj.append(200);
```

```
    sec_obj.Print();
    cout <<"Size of Array is"<<sec_obj.Size();
}
```

```
/* Complete the below program.
     This program is to illustrate dynamic memory allocation and copy constructor
   */


#include<iostream>

using namespace std;

class Stack{
    int *data;
    int sz;
    int itop;
   public:
    Stack(){
       data = NULL;
       sz=0;
       itop=0;
    }
    Stack(int size): itop(0), sz(size){
       data = new int[size];
    }
    Stack(Stack &stk){

    }
    ~Stack(){
    }
    void push(int val){

    }
    int pop(void){

    }
    int top(){return itop;}
    int isStackEmpty(){
       if(itop == 0) return 1;
       else   return 0;
    }
    int isStackFull(){
       if(itop >=sz) return 1;
       else return 0;
    }
};

int main()
{
    Stack o(10);
    o.push(2);
    o.push(4);
    o.push(5);
    o.push(6);
    o.push(7);
    o.push(20);

    Stack obj(o);

    cout <<obj.pop()<<endl;
    cout <<obj.pop()<<endl;
    cout <<obj.pop()<<endl;
    cout <<"POPing from o"<<endl;
    cout <<o.pop()<<endl;
    cout <<o.pop()<<endl;
    cout <<o.pop()<<endl;
    cout <<o.pop()<<endl;
}

/* Complete the below program */

/***************header.h**********************/

#ifndef __HEADER_H__
#define __HEADER_H__


#include<iostream>
#include<cstring>
```

```
using namespace std;
class Person
{
            char *name;
            char *address;
            char *phone;
     public:
            void init(void);
            void clear(void);
            void setname(char const *str);
            void setaddress(char const *str);
            void setphone(char const *str);
            char const *getname(void)        const;
            char const *getaddress(void)     const;
            char const *getphone(void)       const;
     void printperson(Person const &);
};
#endif
```

```
/************************Person.cpp*************/

// example to constant parameter
#include"header.h"
int num;
void Person :: init()
{
     name=address=phone=0;
}
void Person :: clear()
{
     delete name;
     delete address;
     delete phone;
}
void Person :: setname(char const *str)
{

}
void Person :: setaddress(char const *str)
{

}
void Person :: setphone(char const *str)
{

}
```

```
char const *Person :: getname() const
{
          num = 10;
    return name;
}
char const *Person :: getaddress() const
{
    return address;
}
char const *Person :: getphone() const
{
    return phone;
}
void Person::printperson(Person const &p)
{
    if(p.getname())
                cout<<"name="<<p.getname()<<endl;
    if(p.getaddress())
                cout<<"address"<<p.getaddress()<<endl;
    if(p.getname())
                cout<<"phone="<<p.getphone()<<endl;
}


/* Complete the below program */
/**********************************************
Inheritance
***********************************************/
#include<iostream>
using namespace std;

class Box{
   float  height;
   float  width;
   float  length;
   float  volume;
   float surfacearea;

   public:
     Box(){}
     Box(float h, float w, float l){

     }
     Box(Box & b){

     }
     void setHeight(float h){
        height = h;
     }
     void setLength(float l){
        length = l;
    }
    void setWidth(float w){
        width = w;
    }
    double fnVolume(){
        volume = length * width * height;
        return  volume;
    }
    double  fnsurfacearea(){

    }
    float  getVolume(){
        return volume;
    }
    float getSurfaceArea(){
        return surfacearea;
    }
    void Print(){
        cout <<"volume is : "<<volume<<endl;
        cout <<"surfacearea is:"<<surfacearea<<endl;
    }
    float getHeight(void){
        return height;
    }
    float getLength(void){
        return length;
    }
    float getWidth(void){
        return width;
    }

};
class Boxcolor:public Box{
     float color;
     public:
    Boxcolor(){}
    Boxcolor(float h,float  w,float l,float c):Box(h,w,l){
        color = c;
    }
    float setColor(float c){
        color = c;
    }
    float getColor(){
        return color;
    }

};
int main()
{
   Boxcolor   obj(12.6,21.3,41.5,0x45);
```

```
   obj.fnVolume();
   obj.fnsurfacearea();
   obj.Print();

   cout <<hex<<"color:"<<obj.getColor()<<endl;
   cout <<"Lengthis :"<<obj.getLength()<<endl;
   cout <<"Volume is :"<<obj.getVolume()<<endl;
}
```

```
/************************************************
Inheritance
Complete the below program

*************************************************/

#include<iostream>
using namespace std;

class Cube{
   float  height;
   float  width;
   float  length;
   float  volume;
   float surfacearea;

   public:
     Cube(){

     }
     Cube(float h, float w, float l){

     }
     Cube(Cube & b){

     }
     virtual ~Cube(){}
```

```
    void setHeight(float h){
      height = h;
    }
    void setLength(float l){
      length = l;
    }
    void setWidth(float w){
      width = w;
    }
    double fnVolume(){

    }
    double  fnsurfacearea(){

    }
    float  getVolume(){
      return volume;
    }
    float getSurfaceArea(){
      return surfacearea;
    }
    void Print(){
      cout <<"volume is : "<<volume<<endl;
      cout <<"surfacearea is:"<<surfacearea<<endl;
    }
    float getHeight(void){
      return height;
    }
    float getLength(void){
      return length;
    }
    float getWidth(void){
      return width;
    }

};
class Boxweight:public  Cube{
   float weight;
   public:
    Boxweight(){}
    Boxweight(float h,float  w,float l,float m):Cube(h,w,l){
      weight = m;
    }
    float setWeight(float w){
      weight = w;
    }
    float getWeight(void){
      return weight;
    }

};
```

```
int main()
{
   Boxweight   obj(12.6,21.3,41.5,45);
   obj.fnVolume();
   obj.fnsurfacearea();
   obj.Print();

   cout <<hex<<obj.getWeight()<<endl;
   cout <<obj.getLength()<<endl;
   cout <<obj.getVolume()<<endl;
}
```

```
/************************************************
This exercise is to illustrate Inheritance
Complete the below program

*************************************************/
#include<iostream>
using namespace std;

class Box{
   float  height;
   float  width;
   float  length;
   float  volume;
   float surfacearea;

   public:
     Box(){}
     ~Box(){cout <<"Box";}
     Box(float h, float w, float l){

     }
     Box(Box & b){

     }
     void setHeight(float h){
        height = h;
     }
```

```
    void setLength(float l){
       length = l;
    }
    void setWidth(float w){
       width = w;
    }
    double fnVolume(){

       return  volume;
    }
    double  fnsurfacearea(){

       return surfacearea;
    }
    float  getVolume(){
       return volume;
    }
    float getSurfaceArea(){
       return surfacearea;
    }
    void Print(){
       cout <<"volume is : "<<volume<<endl;
       cout <<"surfacearea is:"<<surfacearea<<endl;
    }
    float getHeight(void){
       return height;
    }
    float getLength(void){
       return length;
    }
    float getWidth(void){
       return width;
    }

};
class Boxweight:public virtual Box{
   float weight;
   public:
    Boxweight(){}
    ~Boxweight(){cout <<"Boxweight"<<endl;}
    Boxweight(float  w){
       weight = w;
    }
    Boxweight(float h,float  w,float l,float m):Box(h,w,l){
       weight = m;
    }
    float setWeight(float w){
       weight = w;
    }
    float getWeight(void){
       return weight;
```

```
      }

};
class Boxcolor:public virtual Box{
    float color;
    public:
      Boxcolor(){}
      ~Boxcolor(){cout <<"Boxcolor"<<endl;}
      Boxcolor(float  c){
         color  = c;
      }
      Boxcolor(float h,float  w,float l,float c):Box(h,w,l){
         color = c;
      }
      float setColor(float c){
         color = c;
      }
      float getColor(){
         return color;
      }

};
class Shipment: public Boxcolor,public Boxweight{
    float cost;
    public:
      Shipment(){}
      ~Shipment(){cout <<"Shipment"<<endl;}
      Shipment(float h,float  w,float l,float c,float m ,float cost):Box(h,w,l),Boxweight(m),Boxcolor(c){
         this->cost = cost;
      }
      float setColor(float c){
         cost = c;
      }
      float getColor(){
         return cost;
      }

};
int main()
{
   Shipment   *obj = new Shipment(12.6,21.3,41.5,99.9,0x45,1000.00);
   obj->fnVolume();
   obj->fnsurfacearea();
   obj->Print();

   cout <<hex<<"color:"<<obj->getColor()<<endl;
   cout <<"Lengthis :"<<obj->getLength()<<endl;
   cout <<"Volume is :"<<obj->getVolume()<<endl;
   delete obj;

}
```

```
//: C14:Car.cpp
// Public composition
class Engine {
public:
void start() const {}
void rev() const {}
void stop() const {}
};
class Wheel {
public:
void inflate(int psi) const {}
};
class Window {
public:
void rollup() const {}
void rolldown() const {}
};
class Door {
public:
Window window;
void open() const {}
void close() const {}
};
class Car {
public:
Engine engine;
Wheel wheel[4];
Door left, right; // 2-door
};
int main() {
Car car;
car.left.window.rollup();
car.wheel[0].inflate(72);
} ///:~
```

Modify Car.cpp so that it also inherits from a class called
Vehicle, placing appropriate member functions in Vehicle (that is, make up some member functions).

Add a nondefault constructor to Vehicle, which you must call
inside Car’s constructor.

Create a class with two static member functions. Inherit from this class and redefine one of the member functions.
Show that the other is hidden in the derived class.

```
/* Operator overloading */ Complete below programming
/********************************************************************************
*         File name: program5.cpp                                                       *
*         Version No:1.4                                                       *
*         Author  :                                          *
*      Description:  Strings comparison using Relational operator overloading *
********************************************************************************/

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

const int BUFF_SIZE=50;
enum boolean {FALSE,TRUE};
class String
{
          char str[BUFF_SIZE];
    public:
          String() { strcpy(str," "); }
          void read() { cin>>str; }
          void echo() { cout<<str; }
          boolean operator <(String s)
          {

          }
           boolean operator >(String s)
           {

           }
           boolean operator ==(char *MyStr)
           {

          }
};
main()
{
          String str1,str2;
```

```
        while(TRUE)
        {
                cout<<"\nEnter string1<'end' to stop>:";
                str1.read();
                if(str1=="end")
                        break;
                cout<<"Enter string2:";
                str2.read();
                cout<<"\n comparison Status:";
                str1.echo();
                if(str1<str2)
                        cout<<"<";
                else if(str1>str2)
                        cout<<">";
                else
                        cout<<"=";
                str2.echo();
        }
        cout<<"\n Bye!!Thats all folks........";
}
```

Complete below programming

```
/*********************************************************************
*       File name: program7.cpp                                     *
*       Version No:1.3                                        *
*       Author  :                                   *
*  Description:  Conversion between objects and Basic types      */
/*********************************************************************/
#include<iostream>
using namespace std;

class metre
{
        float length;
   public:
        metre() {  length=0.0;    }
        explicit metre(float Initlength)   {  length=Initlength/100.0;  }
        //conversion from userdefined type to basic datatype
        operator float()
        {
        }
        void getlength()
        {
        }
        void showlength()
        {
                cout<<"\nlength in metres:"<<length;
```

```
            }
};

main()
{
            metre metre1;
            float length1;
            cout<<"\nEnter length in cms:";
            cin>>length1;
             metre1=length1;
            metre1.showlength();
            metre metre2;
            float length2;
            metre2.getlength();
            length2=metre2;
            cout<<"\nLength in cms="<<length2;
}
```

Complete below programming

```
/**********************************************************************
*           File name: program7.cpp                                           *
*           Version No:1.3                                               *
*           Author  :                                     *
*      Description:  Overloading of new and delete operators          */
/*******************************************************************/
#include<iostream>
# include<cstdio>
using namespace std;

const int ARRAY_SIZE=5;

void* operator new(size_t size)
{
            cout <<"hai there\n";
            fflush(stdout);
            return(malloc(size));
}

class Vector
{
            int i;
            int *array;
    public:
            void * operator new(size_t size)
            {

                return my_Vector;
            }
            void operator delete(void * vec)
            {

            }
            void read();
            int sum();
};

void Vector :: read()
{
            int x;
            for(int i=0;i<ARRAY_SIZE;i++){
                fflush(stdout);
                cout<<"Vector["<<i<<"]=?\n";
                cin>>array[i];
            }
}
int Vector :: sum()
{
            int total=0;
            for(int i=0;i<ARRAY_SIZE;i++)
            total+=array[i];
            return total;
}
main()
{
            Vector *my_Vector=new Vector;
            cout<<"Enter Vector data...."<<endl;
            my_Vector->read();
            cout<<"Sum of Vector="<<my_Vector->sum();
            delete my_Vector;

}

//Complete below programming

#include<iostream>
using namespace std;

class complex
{
    private:
            float real;
            float imag;
    public:
            complex() { real=imag=0; }
            void getdata()
```

```
            {
                        cout<<"Real part?";
                        cin>>real;
                        cout<<"\nImag part?";
                        cin>>imag;
            }
            void outdata(char *msg)
            {
                        cout<<endl<<msg;
                        cout<<"("<<real;
                        cout<<","<<imag<<")";
            }
            void operator +=(complex c2);
            void operator -=(complex c2);
            void operator *=(complex c2);
            void operator /=(complex c2);
};
```

```
//Complete below programming

 #include<iostream>
using namespace std;

class Degree{
   float  degree;
   public:
      Degree(){}
      ~Degree(){}
      Degree(float   d){
         degree = d;
      }
      Degree operator +(Degree  d){

      }
      Degree operator -(Degree  d){

      }
      Degree operator *(Degree  d){

      }
      Degree operator /(Degree  d){

         return temp;
      }
      void fndisplay(){
         cout <<"Degree is : "<< degree<<endl;
      }
};


int main()
{
   Degree d1(30.0),   d2(60.0),   d3;
   Degree d4(45.0), d5(90.0);
   d3 = d1 + d2 * d5 / d4; // d1.operator+(d2);
   d3.fndisplay();

  /* d3 = d1 - d2; // d1.operator-(d2);
   d3.fndisplay();
   d3 = d1 * d2; // d1.operator*(d2);
   d3.fndisplay();
   d3 = d1 / d2; // d1.operator/(d2);
   d3.fndisplay();
*/
}
```

```
/***********************************************************
*           File name: friendUnary.cpp                                  *
*           Version No:1.3                                              *
*           Complete below programming                       *
*           Description: Unary operator overloading using friend functions *
***********************************************************/
#include<iostream>
using namespace std;

class complex
{
     private:
             float real;
             float imag;
     public:
             complex(){real=imag=0.0; }
             void outdata(char *msg);
             friend complex operator -(complex c1)
             {

             }
            friend complex operator -(complex c1,complex c2)
            {


            }
            void readdata();
};
void complex :: outdata(char *msg)
{
}
void complex :: readdata()
{
}
main()
{
            complex c1,c2;
            cout<<"Enter the complex number c1"<<endl;
            c1.readdata();
            c2=-c1;
            c1.outdata("complex c1:");
            c2.outdata("complex c2=-complex c1:");
            c2 = c1 - c1;
            c1.outdata("complex c1:");
            c2.outdata("complex c2=-complex c1:");
}
```

```
// Complete below programming

#include<iostream>
using namespace std;

class metre   {
   float length;
            public:
      metre(){  length=0.0;   }
      metre(float Initlength)
             {  length=Initlength/100.0;  }

 //metre(float Initlength) {  length =I nitlength/100.0;  }
//conversion from userdefined type to basic datatype
    operator float()  {
          float lengthcms;
          lengthcms=length*100.0;
          return(lengthcms);
     }
     void getlength()    {
               cout<<"\nEnter length in metres:";
               cin>>length;
     }
     void showlength()    {
               cout<<"\nlength in metres:"<<length;
    }
};
```

```
int main()
{
   metre oMeter(1000.00);
   float  fdata;

   fdata = oMeter;//oMeter.operator float()
   cout << "fadata is : "<<fdata<<endl;
}
```

```
//Complete below programming

#include<iostream>
const float PI=3.141592654;
using namespace std;
class Degree {
      friend class Radian;
           float degree;
    public:
           Degree(float  d = 0.0){ degree=d;}

           float Getdegree(){ return(degree); }
           void output(){  cout<<"Degree="<<degree<<endl;}
};

class Radian {
           float rad;
    public:
           Radian() { rad=0;}

           Radian (float initrad)//constructor 1
           { rad=initrad; }

           float GetRadian() { return (rad); }
           void Input() {
                      cout<<"Enter radian:";
                      cin>>rad;
```

```
            }
            void output(){ cout<<"Radian="<<GetRadian(); }
           void Display(Degree *ref){
               cout <<"Degree is :"<<ref->degree<<endl;
            }
};
int  main(void)
{
      Degree deg1(55.0);
      Radian rad2(99.00);

      rad2.Display(&deg1);

}
```

```
/*************************************************************
*           File name: prog13.cpp                                              *
*           Version No:1.3                                                     *
*                                                            *
*           Description: Subscript operator overloading            *
**************************************************************/
#include<iostream>
#include<cstring>
using namespace std;

typedef struct AccountEntry
{
            int number;
            char name[25];
}AccountEntry;

class AccountBook
{
            int acct;
            AccountEntry account[10];
    public:
            AccountBook(int acctIn){  acct=acctIn;}
            void AccountEntry();
            int operator [](char *nameIn);
            char *operator [](int numberIn);
};
```

```
int AccountBook :: operator[](char *nameIn)
{
        for(int i=0;i<acct;i++)
                if(strcmp(nameIn,account[i].name)==0)
                        return account[i].number;
        return 0;
}
char *AccountBook :: operator [](int numberIn)
{

}
void AccountBook :: AccountEntry()
{

}

main()
{
   int accno;
   char name[25];
   AccountBook accounts(5);
   cout<<"\nBuilding 5 customers database"<<endl;
   accounts.AccountEntry();
   cout<<"\nAccessing Accounts Information";
   cout<<"\nTo access Name Enter Account Number:";
   cin>>accno;
   cout<<"Name:"<<accounts[accno];
   cout<<"\nTo access Account number, Enter name";
   cin>>name;
   cout<<"Account Number:"<<accounts[name];
}

/***********************************************
Polymorphism: This exercise is to illustrate virtual function and function overloading
Complete the below program

***********************************************/

#include<iostream>
#include<cmath>

using namespace std;
class Triangle
{
  protected:
    int side1, side2, side3;
    float  area;
    float perimeter;
  public:
    Triangle(){}
```

```
      Triangle(int s1, int s2, int s3){

      }
      virtual float fnarea()=0;
      virtual float fnperimeter()=0;

      void Print(){
        cout <<"side1 is: "<<side1  <<endl;
        cout <<"side2 is: "<<side2  <<endl;
        cout <<"side3 is: "<<side3  <<endl;
        cout <<"area is: "<< area <<endl;
        cout <<"perimeter is: "<<perimeter  <<endl;
      }
};

class Rtriangle: public Triangle{
   public:
      Rtriangle(int s1, int s2, int s3):Triangle(s1,s2,s3){}

      virtual float fnarea(){

         return area;
      }
      virtual float fnperimeter(){

         return perimeter;
      }
};
class Sctriangle: public Triangle{
   public:
      Sctriangle(int s1, int s2, int s3):Triangle(s1,s2,s3){}
      virtual float fnarea(){
          float s;

          return area;
      }
      virtual float fnperimeter(){

         return perimeter;
      }
};

/***********************************************
Polymorphism: This exercise is to illustrate virtual function and function overloading
Complete the below program

***********************************************/


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

```
class Shape
{
   protected:
     float  side1, side2;
     float  area, perimeter;
   public:
      Shape(){}
       ~Shape(){cout << "Shape"<<endl;}
      Shape(float s1,float s2){

      }
      virtual float fnarea()=0;//Pure virtual function
      virtual float fnperimeter()=0;

};
class rTriangle:public Shape
{
   float  side3;
   public:
     rTriangle(){}
     ~rTriangle(){cout << " Triangle"<<endl;}
     rTriangle(float s1,float  s2,float s3):Shape(s1,s2)      {
        side3 = s3;
     }
     float fnarea(){

     }
     float fnperimeter() {

             }
};
class Rectangle:public Shape
{
   public:
     Rectangle(){}
     ~Rectangle(){cout <<"Rectangle" <<endl;}
     Rectangle(float s1,float s2):Shape(s1,s2){ }
     float fnarea(){

     }
     float fnperimeter() {

     }
};

int main()
{
   Shape *ptr=new rTriangle(4,6,9);
   cout<<ptr->fnarea()<<endl;
   cout<<ptr->fnperimeter()<<endl;
```

```
    delete  ptr;

    Shape *ptr1=new Rectangle(7,9);
    cout<<ptr1->fnarea()<<endl;
    cout<<ptr1->fnperimeter()<<endl;

    delete  ptr1;
}
```

```
/***********************************************
Polymorphism: This exercise is to illustrate virtual function and function overloading
Complete the below program

************************************************/

#include<iostream>

using namespace std;

class Number{
    protected:
        int value;
    public:
        Number(){}
        Number(int  val){
            value = val;
        }
        virtual void Print(){ }

        virtual ~Number(){  cout <<"Number"<<  endl; }
};
class Hex:public Number  {
    public:
        Hex(){}
        Hex(int v) :Number(v){  }
        virtual void Print(){
```

```
        cout <<"Value is : "<<hex<<value ;
      }
      void bar(){cout <<"In bar function\n";}
      ~Hex(){ cout <<"Hex destructor"  <<endl;}
};
class Oct:public Number  {
   public:
      Oct() {  }
      Oct(int v) :Number(v){  }
      virtual void Print(){
        cout <<"Value is : "<<oct<<value ;
      }
      void foo(){cout <<"foo in Oct\n";}
      ~Oct(){ cout <<"Octal destructor" <<endl;}
};
class Decimal:public Number  {
   public:
      Decimal(){}
      Decimal(int v) :Number(v){  }
      virtual void Print(){
        cout <<"Value is : "<<dec<<value ;
      }
};

void Select(Number &ref){
   ref.Print();
}
int main()
{
   Number   *ptr = new Oct(55);
   ptr->Print();
   delete  ptr;

   ptr = new Decimal(100);
   ptr->Print();
   delete  ptr;

   ptr = new Hex(100);
   ptr->Print();
   delete  ptr;

   //Select(oct_obj);

   Hex   hex_obj(55);
   Select(hex_obj);
   //hex_obj.Print();

}

/***********************************************************
Polymorphism: This exercise is to illustrate virtual function and function overloading
```

Complete the below program

*************************************************************/

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

class Triangle
{
   protected:
      int side1, side2, side3;
      float  area;
      float perimeter;
   public:
      Triangle(){}
      Triangle(int s1, int s2, int s3){

      }
      virtual float fnarea()=0;
      virtual float fnperimeter()=0;

      void Print(){
        cout <<"side1 is: "<<side1  <<endl;
        cout <<"side2 is: "<<side2  <<endl;
        cout <<"side3 is: "<<side3  <<endl;
        cout <<"area is: "<< area <<endl;
        cout <<"perimeter is: "<<perimeter  <<endl;
      }
};

class Rtriangle: public Triangle{
   public:
      Rtriangle(int s1, int s2, int s3):Triangle(s1,s2,s3){}

      virtual float fnarea(){

      }
      virtual float fnperimeter(){

      }
};
class Sctriangle: public Triangle{
   public:
      Sctriangle(int s1, int s2, int s3):Triangle(s1,s2,s3){}
      virtual float fnarea(){
          float s;

      }
      virtual float fnperimeter(){
          return perimeter;
       }
};
int main()
{
     Triangle   *Bptr=NULL;
     Sctriangle   o(3,5,8);
     Rtriangle   oR(4,7,6);

     Bptr = &o;
     cout <<"Area of triangle is "<<Bptr->fnarea()<<endl;
     cout <<"Primeter is :"<<Bptr->fnperimeter()<<endl;
     Bptr->Print();


     Bptr = &oR;
     cout <<"Area of triangle is "<<Bptr->fnarea()<<endl;
     cout <<"Primeter is :"<<Bptr->fnperimeter()<<endl;
     Bptr->Print();
}
```

```
/***********************************************
Polymorphism: This exercise is to illustrate virtual function and function overloading
************************************************/

#include<iostream>
using namespace std;

class temp{
protected:
   float  val;
   char  symbol;
   float cel, far, kel;
   public:
     temp(){}
     virtual ~temp(){cout <<"temp"<<endl;}
     temp(float v, char s){
        val = v;
        symbol = s;
     }
     virtual void makeHot() = 0;
     virtual void makeCold() = 0;
     virtual void freeze() = 0;
     virtual void boil() = 0;
     virtual float display() = 0;
};

class celcius:public virtual temp{
    float cel;
    public:
     celcius(){}
     ~celcius(){cout <<"celcius"<<endl;}
     celcius(float v, char s):temp(v, s){
        if(s == 'F')
        {
           cel = (v - 32) * 5/9;
        }
        else if(s == 'K')
        {
           cel = v - 273.15;
        }
        else
        {
           cel = v;
        }
     }

     void makeHot(){cel += 5;}
     void makeCold(){cel -= 5;}
     void freeze(){cel = 0;}
     void boil(){cel = 100;}
     float display(){
       return cel;
     }
};

class Fahrenheit:public virtual temp{
    float far;
    public:
     Fahrenheit(){}
     ~Fahrenheit(){cout <<"Fahrenheit"<<endl;}
     Fahrenheit(float v, char s):temp(v, s){
        if(s == 'C')
        {
           far = (v * 9/5) + 32;
        }
        else if(s == 'K')
        {
           far = (v - 273.15) * 9/5 + 32;
        }
        else
        {
           far = v;
        }
     }

     void makeHot(){far += 5;}
     void makeCold(){far -= 5;}
     void freeze(){far = 32;}
     void boil(){far = 212;}
     float display(){
       return far;
     }
};

class Kelvin:public virtual temp{
    float kel;
    public:
     Kelvin(){}
     ~Kelvin(){cout <<"Kelvin"<<endl;}
     Kelvin(float v, char s):temp(v, s){
        if(s == 'C')
        {
           kel = v + 273.15;
        }
        else if(s == 'F')
        {
           kel = (v - 32) * 5/9 + 273.15;
        }
        else
        {
           kel = v;
        }
     }

     void makeHot(){kel += 5;}
     void makeCold(){kel -= 5;}
     void freeze(){kel = 32;}
     void boil(){kel = 373.15;}
     float display(){
       return kel;
     }
};

int main()
{
   temp *ptrTemp = new celcius(2,'F');
   cout<<ptrTemp->display()<<endl;
   ptrTemp->makeHot();
   cout<<ptrTemp->display()<<endl;
   ptrTemp->makeCold();
   cout<<ptrTemp->display()<<endl;
   ptrTemp->freeze();
   cout<<ptrTemp->display()<<endl;
   ptrTemp->boil();
   cout<<ptrTemp->display()<<endl;

   temp *ptrTemp1 = new Fahrenheit(56,'K');
   cout<<ptrTemp1->display()<<endl;
   ptrTemp1->makeHot();
```

```
    cout<<ptrTemp1->display()<<endl;
    ptrTemp1->makeCold();
    cout<<ptrTemp1->display()<<endl;
    ptrTemp1->freeze();
    cout<<ptrTemp1->display()<<endl;
    ptrTemp1->boil();
    cout<<ptrTemp1->display()<<endl;

    temp *ptrTemp2 = new Kelvin(22,'C');
    cout<<ptrTemp2->display()<<endl;
    ptrTemp2->makeHot();
    cout<<ptrTemp2->display()<<endl;
    ptrTemp2->makeCold();
    cout<<ptrTemp2->display()<<endl;
    ptrTemp2->freeze();
    cout<<ptrTemp2->display()<<endl;
    ptrTemp2->boil();
    cout<<ptrTemp2->display()<<endl;

}
```

```
/*   This program to illustrate Polymorphism using interface  */

#include<iostream>
using namespace std;

class iAdd{
    public: virtual int add()=0;
    virtual ~iAdd(){}
};
class iMul{
    public: virtual int mul()=0;
    virtual ~iMul(){}
};
class iSub{
    public: virtual int sub()=0;
    virtual ~iSub(){}
};
class iDiv{
    public: virtual int div()=0;
    virtual ~iDiv(){}
};

class IntNumber:public iAdd,iMul,iSub,iDiv{
    int inum1, inum2;
    public:
    IntNumber(){}
    IntNumber(int n1, int n2){inum1=n1;inum2=n2;}
    virtual int add(){return inum1 + inum2;}
    virtual int mul(){return inum1 * inum2;}
    virtual int sub(){return inum1 - inum2;}
    virtual int div(){return inum1 / inum2;}
};
class FloatNumber:public iAdd,iMul,iSub,iDiv{
    float fnum1, fnum2;
    public:
    FloatNumber(){}
    FloatNumber(float n1, float n2){fnum1=n1;fnum2=n2;}
    virtual int add(){return fnum1 + fnum2;}
    virtual int mul(){return fnum1 * fnum2;}
    virtual int sub(){return fnum1 - fnum2;}
    virtual int div(){return fnum1 / fnum2;}
};

int main()
{
    IntNumber   int_o(213,2334);
    FloatNumber float_o(676.75,821.368);

    cout <<int_o.add()<<endl;
    cout <<int_o.mul()<<endl;
    cout <<float_o.add()<<endl;
    cout <<float_o.mul()<<endl;
    return 0;
}
```