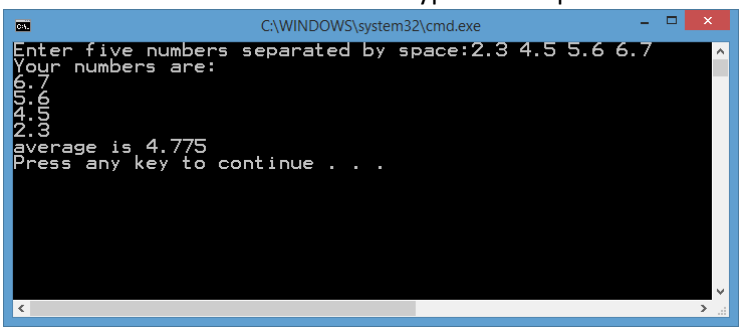
Final Review Fall 2015

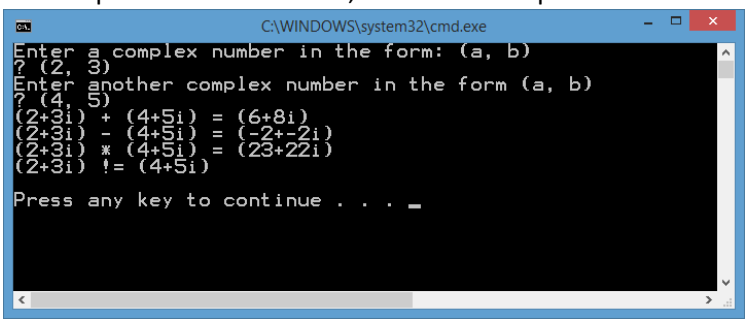
Programs

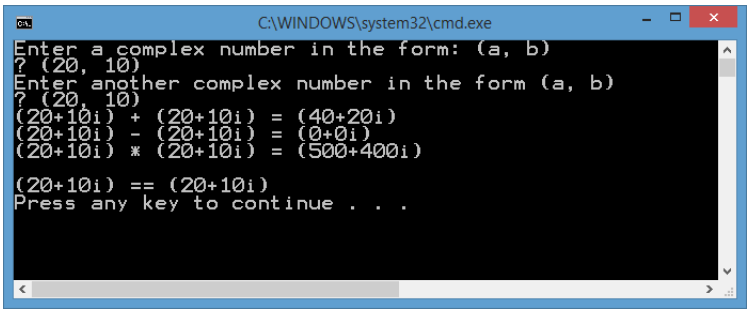
1. Write a program which will ask user to enter four numbers and then output in reverse order. Your program will also output the average of these numbers. You must declare an array of type double and then use a variable of type double pointer to access the array elements.




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


int main()
{
	double arr[4];
	int i;
	double *p = arr;
	double sum = 0;
	cout << "Enter four numbers separated by space:";
	cin >> *p >> *(p + 1) >> *(p + 2) >> *(p + 3);
	cout << "Your numbers are:\n";
	for (i = 3; i >= 0; i--)
	{
		cout << *(p + i) << endl;
		sum += *(p + i);
	}
	cout << "average is " << (sum / 4) << endl;
	return 0;

}
```

2. Construct a complex class with operators +, -, *, / overloaded. Also overload >> and << for input and output. Also overload ==, and != for comparison.






```
#ifndef COMPLEX_H
#define COMPLEX_H

#include <iostream>

class Complex
{
   friend std::ostream &operator<<( std::ostream &, const Complex & );
   friend std::istream &operator>>( std::istream &, Complex & );
public:
   explicit Complex( double = 0.0, double = 0.0 ); // constructor
   Complex operator+( const Complex& ) const; // addition
   Complex operator-( const Complex& ) const; // subtraction
   Complex operator*( const Complex& ) const; // multiplication
   Complex& operator=( const Complex& ); // assignment
   bool operator==( const Complex& ) const;
   bool operator!=( const Complex& ) const;
private:
   double real; // real part
   double imaginary; // imaginary part
}; // end class Complex

#endif
```

```
#include <iostream>
#include "Complex.h"
```

```
using namespace std;

// Constructor
Complex::Complex( double realPart, double imaginaryPart )
   : real( realPart ),
   imaginary( imaginaryPart )
{
   // empty body
} // end Complex constructor

// addition operator
Complex Complex::operator+( const Complex &operand2 ) const
{
   return Complex( real + operand2.real,
      imaginary + operand2.imaginary );
} // end function operator+

// subtraction operator
Complex Complex::operator-( const Complex &operand2 ) const
{
   return Complex( real - operand2.real,
      imaginary - operand2.imaginary );
} // end function operator-

// Overloaded multiplication operator
Complex Complex::operator*( const Complex &operand2 ) const
{
   return Complex(
      ( real * operand2.real ) - ( imaginary * operand2.imaginary ),
      ( real * operand2.imaginary ) + ( imaginary * operand2.real ) );
} // end function operator*

// Overloaded = operator
Complex& Complex::operator=( const Complex &right )
{
   real = right.real;
   imaginary = right.imaginary;
   return *this;   // enables concatenation
} // end function operator=

bool Complex::operator==( const Complex &right ) const
{
   return ( right.real == real ) && ( right.imaginary == imaginary );
} // end function operator==

bool Complex::operator!=( const Complex &right ) const
{
   return !( *this == right );
} // end function operator!=

ostream& operator<<( ostream &output, const Complex &complex )
```

```
{
   output << "(" << complex.real << "+" << complex.imaginary << "i)";
   return output;
} // end function operator<<

istream& operator>>( istream &input, Complex &complex )
{
   char rest[256];
   input.ignore(); // skip (
   input >> complex.real;
   input.ignore( 2 ); // skip ',' and space
   input >> complex.imaginary;
   input.getline(rest, 256); // skip )
   return input;
} // end function operator>>
```

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

int main()
{
   Complex x, y, z;

   cout << "Enter a complex number in the form: (a, b)\n? ";
   cin >> x; // demonstrating overloaded >>
   cout << "Enter another complex number in the form (a, b)\n? ";
   cin >> y;
   z = x + y;
   cout << x << " + " << y << " = " << z << endl;
   z = x - y;
   cout << x << " - " << y << " = " << z << endl;
   z = x * y;
   cout << x << " * " << y << " = " << z << endl;

   if ( x != y ) // demonstrating overloaded !=
      cout << x << " != " << y << '\n';

   cout << '\n';

   if ( x == y ) // demonstrating overloaded ==
      cout << x << " == " << y << '\n';
} // end main
```

3. Write a class call it CDAccount. It has three data members: balance, interest rate and term of the CD in year. Assuming the CD get interest monthly.

This class contains two constructors. One default and the other one has three parameters

CDAccount() /*do nothing */

CDAccount(double bal, double intRate, int T)

There are three getters for these three data members. Let's assume they are called

InterestRate() which will return interest rate

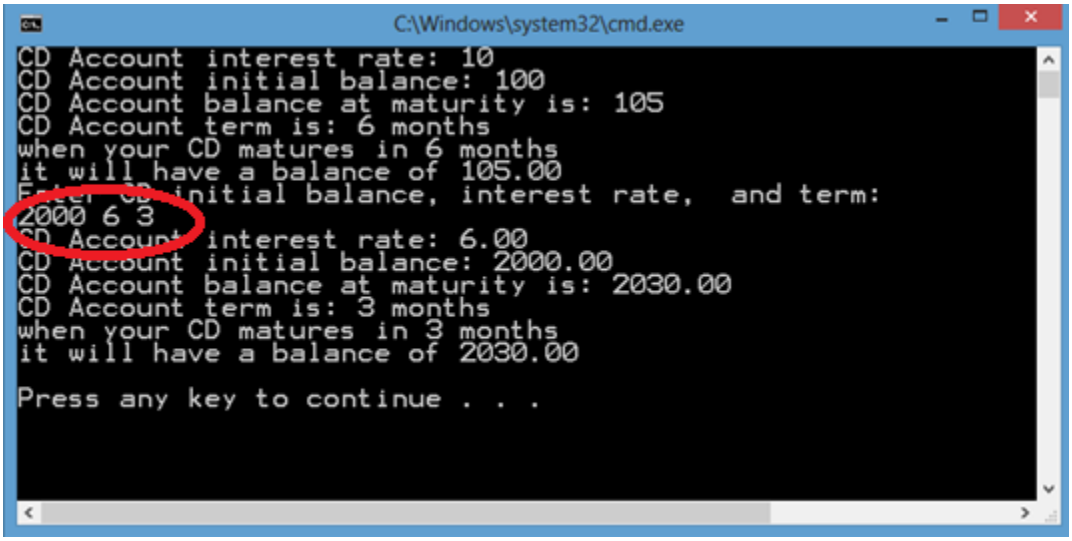
InitialBalance() which will return balance in our case here

Term() which will return the term of CD in year

BalanceAtMaturity() is a method that compute the balance at maturity

Write your driver and define your class member functions so that the output would like the following

The three numbers circled in red was entered by the user.



If you have difficulty to come up with the solution, try to fill in the following code

```
#include <iostream>
using namespace std;
class CDAccount
{
public:
	CDAccount();
	CDAccount(double bal, double intRate, int T);
	double InterestRate();
	double InitialBalance();
	double BalanceAtMaturity();
	int Term();
	void input(istream&);
```

```
        void output(ostream&);
private:
        double balance;
        double interestRate; // in PER CENT
        int term; // months to maturity;
};
int main()
{
        double balance; double intRate;
        int term;
        CDAccount account = CDAccount(100.0, 10.0, 6);
        cout << "CD Account interest rate: "
                << account.InterestRate() << endl;
        cout << "CD Account initial balance: "
                << account.InitialBalance() << endl;
        cout << "CD Account balance at maturity is: "
                << account.BalanceAtMaturity() << endl;
        cout << "CD Account term is: " << account.Term()
                << " months" << endl;
        account.output(cout);
        ////////////////////////////////////////////////////
        cout << "Enter CD initial balance, interest rate, "
                << " and term: " << endl;
        account.input(cin);
        cout << "CD Account interest rate: "
                << account.InterestRate() << endl;
        cout << "CD Account initial balance: "
                << account.InitialBalance() << endl;
        cout << "CD Account balance at maturity is: "
                << account.BalanceAtMaturity() << endl;
        cout << "CD Account term is: " << account.Term()
                << " months" << endl;
        account.output(cout);
        cout << endl;
}

//////////////////// implement the class member functions /////////////////////////
CDAccount::CDAccount() { /* do nothing */ }
CDAccount::CDAccount(double bal, double intRate, int T)
{
        balance = bal;
        interestRate = intRate;
        term = T;
}
double CDAccount::InterestRate()
{
        return interestRate;
}
double CDAccount::InitialBalance()
{
        return balance;
}
double CDAccount::BalanceAtMaturity()
{
        return balance * (1 + (interestRate / 100)*(term / 12.0));
}
int CDAccount::Term()
{
        return term;
```

```
}
void CDAccount::input(istream& inStream)
{
	inStream >> balance;
	inStream >> interestRate;
	inStream >> term;
}
void CDAccount::output(ostream& outStream)
{
	outStream.setf(ios::fixed);
	outStream.setf(ios::showpoint);
	outStream.precision(2);
	outStream << "when your CD matures in " << term
		<< " months" << endl
		<< "it will have a balance of "
		<< BalanceAtMaturity() << endl;
}
```

4. Write a class definition (prototype) for a class called it Smartbut that is a sub class of base class Smart

Below is the Smart.h the header file of Smart class

```
#ifndef SMART_H
#define SMART_H
class Smart
{
public:
	Smart();
	void print_answer() const;
protected:
	int a;
	int b;
};

#endif
```

The constructor Smart() will give initialize values of a, and b.

The derived class Smartbut shall have an additional field called crazy.

There is another member function called is_crazy() that takes no arguments and return a bool value. Smartbut has a default constructor takes no argument and also a three parameter constructor that takes a, b, and crazy as parameters in its constructor heading

The answer is

```
#ifndef SMARTBUT_H
#define SMARTBUT_H
#include "Smart.h"
class Smartbut : public Smart
{
public:
	Smartbut();
	Smartbut(int newA, int newB, bool newCrazy);
	bool is_crazy() const;
private:
```

```
        bool crazy;
};
#endif
```

5. Draw a class hierarchical chart of Shapes, define Square and cube class

Use the polymorphism skill we learned from class, write C++ program modules that can output the area of different geometry objects : Circle, Square, Sphere, and Cube by using virtual function for each of these classes. Assuming that there is class called Shape which is the base class for all other geometrical classes. TwoDimensionalShape and ThreeDimensionalShape are derived from Shape class.

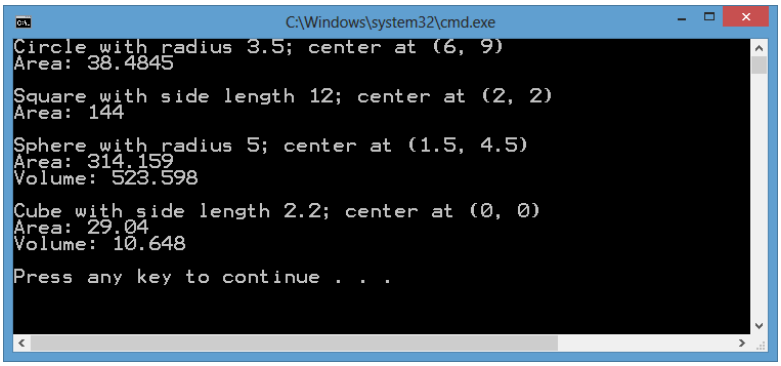
Draw a class hierarchical chart based on the driver program listed below.

The following is an example of running a testing to output the area of different geometry objects.

```
// Driver to test Shape hierarchy
#include <iostream>
#include <array>
#include "Shape.h"
#include "TwoDimensionalShape.h"
#include "ThreeDimensionalShape.h"
#include "Circle.h"
#include "Square.h"
#include "Sphere.h"
#include "Cube.h"
using namespace std;

int main()
{
   // create vector shapes
   array< Shape *, 4 > shapes;

   // initialize vector with Shapes
   shapes[ 0 ] = new Circle( 3.5, 6, 9 );
   shapes[ 1 ] = new Square( 12, 2, 2 );
   shapes[ 2 ] = new Sphere( 5, 1.5, 4.5 );
   shapes[ 3 ] = new Cube( 2.2 );

   // output Shape objects and display area and volume as appropriate
   for ( int i = 0; i < 4; ++i )
   {
      cout << *shapes[ i ] << endl;

      // downcast pointer
      TwoDimensionalShape *twoDimensionalShapePtr =
         dynamic_cast < TwoDimensionalShape * > ( shapes[ i ] );

      // if Shape is a TwoDimensionalShape, display its area
      if ( twoDimensionalShapePtr != 0 )
         cout << "Area: " << twoDimensionalShapePtr->getArea() << endl;

      // downcast pointer
      ThreeDimensionalShape *threeDimensionalShapePtr =
         dynamic_cast < ThreeDimensionalShape * > ( shapes[ i ] );

      // if Shape is a ThreeDimensionalShape, display its area and volume
      if ( threeDimensionalShapePtr != 0 )
         cout << "Area: " << threeDimensionalShapePtr->getArea()
            << "\nVolume: " << threeDimensionalShapePtr->getVolume()
            << endl;

      cout << endl;
```

```
   } // end for
} // end main
```

The screen shot of the running of the driver program



The difference between 2d objects (circle and square) and 3d objects (sphere and cube) is that 2d object doesn't have volume but 3d does.

Based on the output above and study of some of the classes definitions and implementations show below, fill in the missing part on this class-hierarchy:

Prototype and implement of

Square class and Cube class

The Shape class prototype and its class implementation.

The base

```
// Shape.h
// Definition of base-class Shape
#ifndef SHAPE_H
#define SHAPE_H

#include <iostream>
using namespace std;

class Shape {
   friend ostream & operator<<( ostream &, Shape & );
public:
   Shape( double = 0.0, double = 0.0 ); // default constructor
   double getCenterX() const; // return x from coordinate pair
   double getCenterY() const; // return y from coordinate pair
   virtual void print() const = 0; // output Shape object, a pure virtual function
protected:
   double xCenter; // x part of coordinate pair
   double yCenter; // y part of coordinate pair
}; // end class Shape

#endif
```

```
// Shape.cpp
// Member and friend definitions for class Shape

#include "Shape.h"

// default constructor
Shape::Shape( double x, double y )
{
   xCenter = x;
   yCenter = y;
} // end Shape constructor

// return x from coordinate pair
double Shape::getCenterX() const
{
   return xCenter;
} // end function getCenterX

// return y from coordinate pair
double Shape::getCenterY() const
{
   return yCenter;
} // end function getCenterY

// overloaded output operator
```

```
ostream & operator<<( ostream &out, Shape &s )
{
   s.print();
   return out;
} // end overloaded output operator function
```

```cpp
// TwoDimensionalShape.h
// Definition of class TwoDimensionalShape
#ifndef TWODIM_H
#define TWODIM_H

#include "Shape.h"

class TwoDimensionalShape : public Shape
{
public:
   // default constructor
   TwoDimensionalShape( double x, double y ) : Shape( x, y ) { }

   virtual double getArea() const = 0; // area of TwoDimensionalShape
}; // end class TwoDimensionalShape

#endif
```

Note the `TwoDimensionalShape` class is an abstract class and has a pure virtual function getArea. It doesn’t have class implementation. This class is used as a base for all 2d objects

Let’s define and implement the Circle class

```
// Circle.h
// Definition of class Circle
#ifndef CIRCLE_H
#define CIRCLE_H

#include "TwoDimensionalShape.h"

class Circle : public TwoDimensionalShape
{
public:
   // default consturctor
   Circle( double = 0.0, double = 0.0, double = 0.0 );

   virtual double getRadius() const; // return radius
   virtual double getArea() const; // return area
   void print() const; // output Circle object
private:
   double radius; // Circle's radius
}; // end class Circle

#endif
```

```cpp
/**************************************************************************
**************************************************************************/
// Circle.cpp
// Member-function definitions for class Circle
#include <iostream>
#include <stdexcept>
#include "Circle.h"
using namespace std;

// default constructor
Circle::Circle( double r, double x, double y )
   : TwoDimensionalShape( x, y )
{
   if ( r >= 0.0 )
      radius = r;
   else
      throw invalid_argument( "Radius must be >= 0.0" );
} // end Circle constructor

// return radius of circle object
double Circle::getRadius() const
{
   return radius;
} // end function getRadius

// return area of circle object
double Circle::getArea() const
{
   return 3.14159 * radius * radius;
} // end function getArea

// output circle object
void Circle::print() const
{
   cout << "Circle with radius " << radius << "; center at ("
      << xCenter << ", " << yCenter << ")";
} // end function print
```

Square class is not the same as Circle but it is similar. So give square prototype and its implementation here

Xxxxxxxxxxxxxxxxxxx

Write square class prototype and its implementation here

Write a driver and test your implementation

Xxxxxxxxxxxxxxxxxxx

```
// Square.cpp
// Member-function definitions for class Square
#include <iostream>
#include <stdexcept>
#include "Square.h"
using namespace std;

// default constructor
Square::Square( double s, double x, double y )
   : TwoDimensionalShape( x, y )
{
   if ( s >= 0.0 )
      sideLength = s;
   else
      throw invalid_argument( "Side must be >= 0.0" );
} // end Square constructor

// return side of Square
double Square::getSideLength() const
{
   return sideLength;
} // end function getSideLength

// return area of Square
double Square::getArea() const
{
   return sideLength * sideLength;
} // end function getArea

// output Square object
void Square::print() const
{
   cout << "Square with side length " << sideLength << "; center at ("
      << xCenter << ", " << yCenter << ")";
} // end function print


// Square.h
// Definition of class Square
#ifndef SQUARE_H
#define SQUARE_H

#include "TwoDimensionalShape.h"

class Square : public TwoDimensionalShape
{
public:
   // default constructor
   Square( double = 0.0, double = 0.0, double = 0.0 );
```

```
   virtual double getSideLength() const; // return length of sides
   virtual double getArea() const; // return area of Square
   void print() const; // output Square object
private:
   double sideLength; // length of sides
}; // end class Square

#endif
```

Let’s look at the **3D shapes**.

Similarly let’s define an abstract class ThreeDimensionalShape.

Note it is an abstract class and no implementation.

Note also the difference between 2d and 3d. The ThreeDimensionalShape provide a function getVolume which is not in TwoDimensionalShape

---

```
##### 3D shapes classes
// ThreeDimensionalShape.h
// Definition of class ThreeDimensionalShape
#ifndef THREEDIM_H
#define THREEDIM_H

#include "Shape.h"

class ThreeDimensionalShape : public Shape
{
public:
   // default constructor
   ThreeDimensionalShape( double x, double y ) : Shape( x, y ) { }

   virtual double getArea() const = 0; // area of 3-dimensional shape
   virtual double getVolume() const = 0; // volume of 3-dimensional shape
}; // end class ThreeDimensionalShape

#endif
```

Let’s study Sphere class (a 3d class) its definition and implementation.

Sphere class prototype

```
// Sphere.h
// Definition of class Sphere
#ifndef SPHERE_H
#define SPHERE_H

#include "ThreeDimensionalShape.h"

class Sphere : public ThreeDimensionalShape
{
public:
   // default constructor
   Sphere( double = 0.0, double = 0.0, double = 0.0 );

   virtual double getArea() const; // return area of Sphere
   virtual double getVolume() const; // return volume of Sphere
   double getRadius() const; // return radius of Sphere
   void print() const; // output Sphere object
private:
   double radius; // radius of Sphere
}; // end class Sphere

#endif

// Sphere.cpp
// Member-function definitions for class Sphere
#include <iostream>
#include <stdexcept>
#include "Sphere.h"
using namespace std;

// default constructor
Sphere::Sphere( double r, double x, double y )
   : ThreeDimensionalShape( x, y )
{
   if ( r >= 0.0 )
      radius = r;
   else
      throw invalid_argument( "Radius must be >= 0.0" );
} // end Sphere constructor

// return area of Sphere
double Sphere::getArea() const
{
   return 4.0 * 3.14159 * radius * radius;
} // end function getArea

// return volume of Sphere
double Sphere::getVolume() const
{
   return ( 4.0 / 3.0 ) * 3.14159 * radius * radius * radius;
} // end function getVolume

// return radius of Sphere
double Sphere::getRadius() const
{
```

```
   return radius;
} // end function getRadius

// output Sphere object
void Sphere::print() const
{
   cout << "Sphere with radius " << radius << "; center at ("
      << xCenter << ", " << yCenter << ")";
} // end function print
```

The Cube class is similar. Write its prototype and its definition

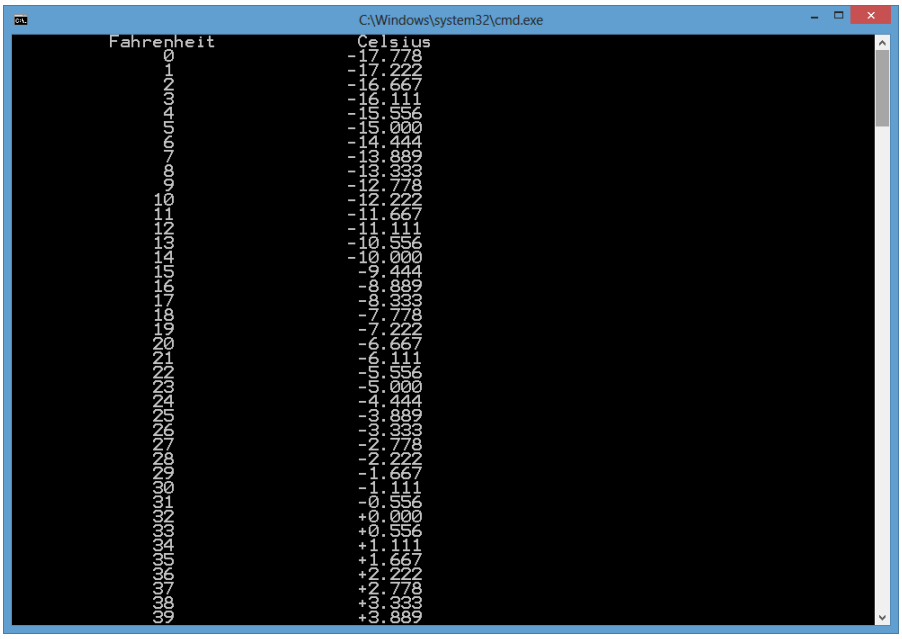
```
// Cube.h
// Definition of class Cube
#ifndef CUBE_H
#define CUBE_H

#include "ThreeDimensionalShape.h"

class Cube : public ThreeDimensionalShape
{
public:
   // default constructor
   Cube( double = 0.0, double = 0.0, double = 0.0 );

   virtual double getArea() const; // return area of Cube object
   virtual double getVolume() const; // return volume of Cube object
   double getSideLength() const; // return length of sides
   void print() const; // output Cube object
private:
   double sideLength; // length of sides of Cube
}; // end class Cube

#endif


// Cube.cpp
// Member-function definitions for class Cube
#include <iostream>
#include <stdexcept>
#include "Cube.h"
using namespace std;

// default constructor
Cube::Cube( double s, double x, double y )
   : ThreeDimensionalShape( x, y )
{
   if ( s >= 0.0 )
      sideLength = s;
   else
      throw invalid_argument( "Side must be >= 0.0" );
} // end Cube constructor

// return area of Cube
double Cube::getArea() const
{
   return 6 * sideLength * sideLength;
} // end function getArea

// return volume of Cube
double Cube::getVolume() const
{
   return sideLength * sideLength * sideLength;
} // end function getVolume

// return length of sides
double Cube::getSideLength() const
{
   return sideLength;
} // end function getSideLength
```

```
// output Cube object
void Cube::print() const
{
   cout << "Cube with side length " << sideLength << "; center at ("
      << xCenter << ", " << yCenter << ")";
} // end function print
```

6. Write a program that convert integer Fahrenheit temperature form 0 to 212 degrees to double Celsius temperatures with 3 digits of precision. Use formula

celsius = 5.0/9.0 * (fahrenheit -32)

Output two columns justified and show both negative sign and positive sign for Celsius values like the following (part of screen shot)

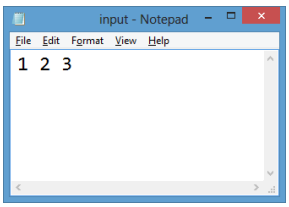


One of the possible solution

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

int main()
{
	double celsius; // holds celsius temperature

	// create column headings with fields of length 20
	cout << setw(20) << "Fahrenheit " << setw(20) << "Celsius\n"
		<< fixed << showpoint;
//	 showing the sign for celsius temperatures
	for (int fahrenheit = 0; fahrenheit <= 212; ++fahrenheit)
	{
		celsius = 5.0 / 9.0 * (fahrenheit - 32);
		cout << setw(15) << noshowpos << fahrenheit << setw(23)
			<< setprecision(3) << showpos << celsius << '\n';
	} // end for
} // end main
```

7. Create a new project. Create a source.cpp file such as the following.

```
//Reads three numbers from the file input.dat, sums the numbers,
//and writes the sum to the file outfile.dat.
#include <fstream>
#include <iostream>
#include <cstdlib>

int main()
{
	using namespace std;
	ifstream in_stream;
	ofstream out_stream;

	in_stream.open("input.dat");
	if (in_stream.fail())
	{
		cout << "Input file opening failed.\n";
		exit(1);
	}
	cout << "Now try open outfile.dat" << endl;

	out_stream.open("outfile.dat");
	if (out_stream.fail())
	{
		cout << "Output file opening failed.\n";
		exit(1);
	}

	int first, second, third;
	in_stream >> first >> second >> third;
	out_stream << "The sum of the first 3\n"
		<< "numbers in infile.dat\n"
		<< "is " << (first + second + third)
		<< endl;

	in_stream.close();
	out_stream.close();

	return 0;
}
```



Create a text file called input.dat which contains three numbers.

Put this text file in the Documents\Visual Studio 2013\Projects\mock-final-exam-file-1\mock-final-exam-file-1\ folder if you are using VS-2013. Other version shall be similar.

If you are using Cygwin, then put the file in the same directory where your program files reside.

(i.e., put this file where your source.cpp resides. Adjust the location based on your installation of visual studio)

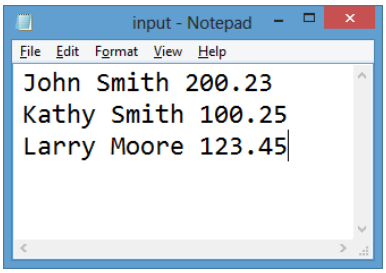
Run the program and see the results.

Open up the outfile.dat and see the result

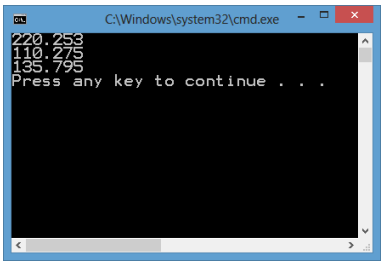
Now, create a program that read in data from input.dat and then write balance *1.1 to an outfile.dat.

After that open outfile.dat and print the results on the screen

Say this is the input



This would be the output. NOTE all numbers multiply 1.1. so 200.23 became 220.253, etc.

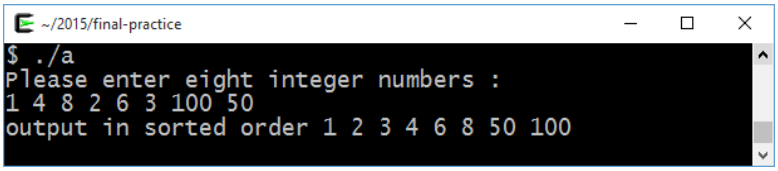


Sol

```
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <string>

int main()
{
	using namespace std;

	ifstream in_stream;
	ofstream out_stream;

	in_stream.open("infile.dat");
	if (in_stream.fail())
	{
		cout << "Input file opening failed.\n";
		exit(1);
	}
```

```
	out_stream.open("outfile.dat");
	if (out_stream.fail())
	{
		cout << "Output file opening failed.\n";
		exit(1);
	}

	string first, last;
	double balance;
	while (!in_stream.eof())
	{
		in_stream >> first >> last >> balance;
		out_stream << balance * 1.1<<endl;
	}

	//out_stream << "The sum of the first 3\n"
	//     << "numbers in infile.dat\n"
	//     << "is " << (first + second + third)
	//     << endl;

	in_stream.close();
	out_stream.close();


	// now open the outfile.dat as input and then output on the screen
	in_stream.open("outfile.dat");
	if (in_stream.fail())
	{
		cout << "Input file opening failed.\n";
		exit(1);
	}

	while (!in_stream.eof())
	{
		in_stream >> balance;
		if (in_stream.eof()) break;
		cout << balance << endl;
	}
	in_stream.close();

	return 0;
}
```

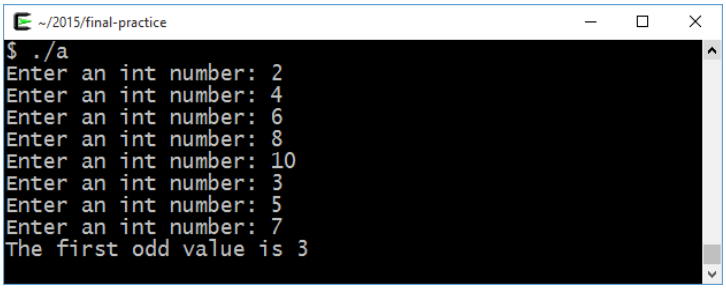
8. Use C++ algorithm sort function write a C++ program that will sort the numbers entered by the user. Assume your program asks user to enter eight integer numbers.



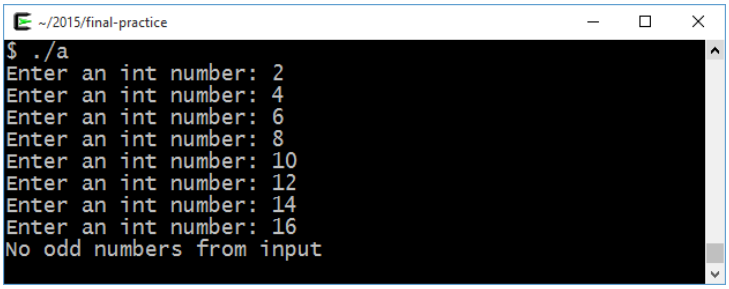
Sol:

```
// C++ sort algorithm
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector

using namespace std;

int main () {
  int myints[8];
  cout << "Please enter eight integer numbers : "<< endl;
  for (int i=0; i < 8; i++)
            cin >> myints[i];
  vector<int> v1  (myints, myints+8);

  // using default comparison (operator <):
  sort (v1.begin(), v1.begin()+8);

  // print out content:
  cout << "output in sorted order";
  for (vector<int>::iterator it=v1.begin(); it!=v1.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';

  return 0;
}
```

9. Declare a vector of typr integer and use find_if function from C++ standard template library to find the first odd integer.



If none are odd



SOL:

```
// find_if example
// from http://www.cplusplus.com/reference/algorithm/find_if/
// and do a little modification
#include <iostream>     // std::cout
#include <algorithm>    // std::find_if
#include <vector>       // std::vector
using namespace std;
bool IsOdd (int i) {
  return ((i%2)==1);
}

int main () {
  vector<int> myvector;
  int x;
  for (int i =0 ; i < 8; i++)
  {
    cout << "Enter an int number: ";
    cin >> x;
    myvector.push_back(x);
  }

  vector<int>::iterator it;
  it = find_if (myvector.begin(), myvector.end(), IsOdd);
```

```
  if (it != myvector.end())
  cout << "The first odd value is " << *it << '\n';
  else cout << "No odd numbers from input" << endl;

  return 0;
}
```