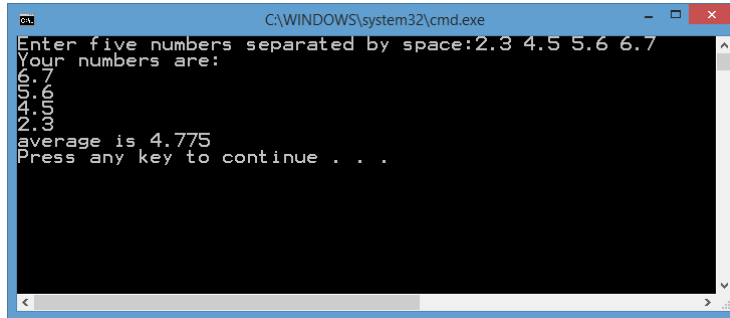


## Final Review Fall 2015

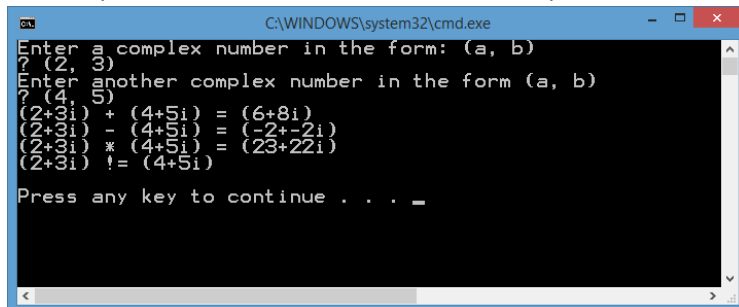
### Programs

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.

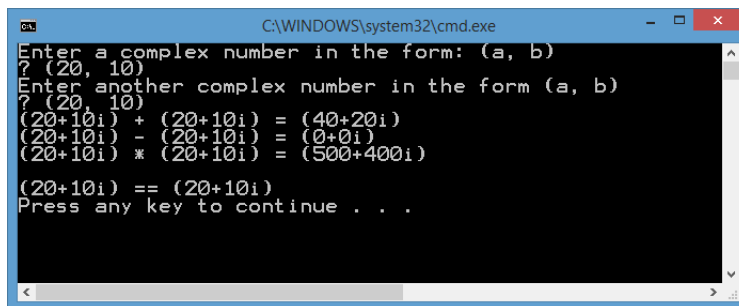


```
C:\WINDOWS\system32\cmd.exe
Enter five numbers separated by space:2.3 4.5 5.6 6.7
Your numbers are:
6.7
5.6
4.5
2.3
Average is 4.775
Press any key to continue . . .
```

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



```
C:\WINDOWS\system32\cmd.exe
Enter a complex number in the form: (a, b)
? (2, 3)
Enter another complex number in the form (a, b)
? (4, 5)
(2+3i) + (4+5i) = (6+8i)
(2+3i) - (4+5i) = (-2+-2i)
(2+3i) * (4+5i) = (23+22i)
(2+3i) != (4+5i)
Press any key to continue . . .
```



```
C:\WINDOWS\system32\cmd.exe
Enter a complex number in the form: (a, b)
? (20, 10)
Enter another complex number in the form (a, b)
? (20, 10)
(20+10i) + (20+10i) = (40+20i)
(20+10i) - (20+10i) = (0+0i)
(20+10i) * (20+10i) = (500+400i)
(20+10i) == (20+10i)
Press any key to continue . . .
```

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

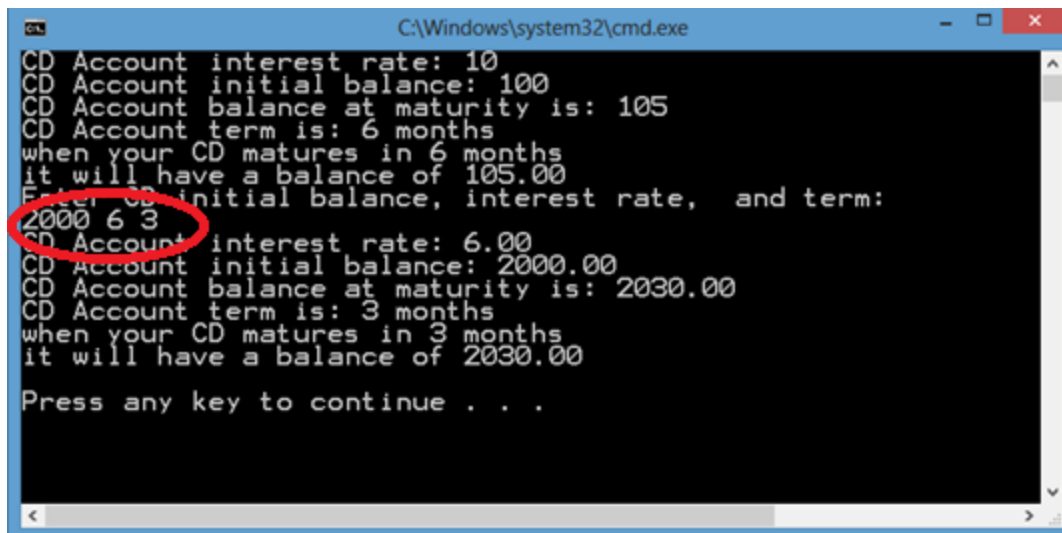
InitialBalance() which will return balance in our case here

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.



```
C:\Windows\system32\cmd.exe
CD Account interest rate: 10
CD Account initial balance: 100
CD Account balance at maturity is: 105
CD Account term is: 6 months
when your CD matures in 6 months
it will have a balance of 105.00
Enter CD initial balance, interest rate, and term:
2000 6 3
CD Account interest rate: 6.00
CD Account initial balance: 2000.00
CD Account balance at maturity is: 2030.00
CD Account term is: 3 months
when your CD matures in 3 months
it will have a balance of 2030.00
Press any key to continue . . .
```

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
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 
}
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)
{
}
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

## 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;

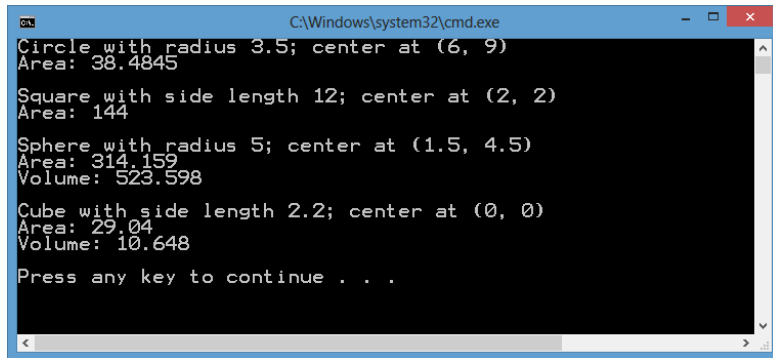
        // downcast pointer
        ThreeDimensionalShape *threeDimensionalShapePtr =
            dynamic_cast < ThreeDimensionalShape * > ( shapes[ i ] );

        // 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



```
C:\Windows\system32\cmd.exe  
Circle with radius 3.5; center at (6, 9)  
Area: 38.4845  
Square with side length 12; center at (2, 2)  
Area: 144  
Sphere with radius 5; center at (1.5, 4.5)  
Area: 314.159  
Volume: 523.598  
Cube with side length 2.2; center at (0, 0)  
Area: 29.04  
Volume: 10.648  
Press any key to continue . . .
```

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
```

```

// 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 constructor
    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
```

```

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

XXXXXXXXXXXXXXXXXXXXX

Write square class prototype and its implementation here

Write a driver and test your implementation

XXXXXXXXXXXXXXXXXXXXX

```
// 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 );

    Fill in missing part here; // return length of sides
    Fill in missing part here; // return area of Square
    Fill in missing part here; // output Square object
private:
    double sideLength; // length of sides
}; // end class Square

#endif
```

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

***Fill in the missing constructor and three function implementations here***

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
```

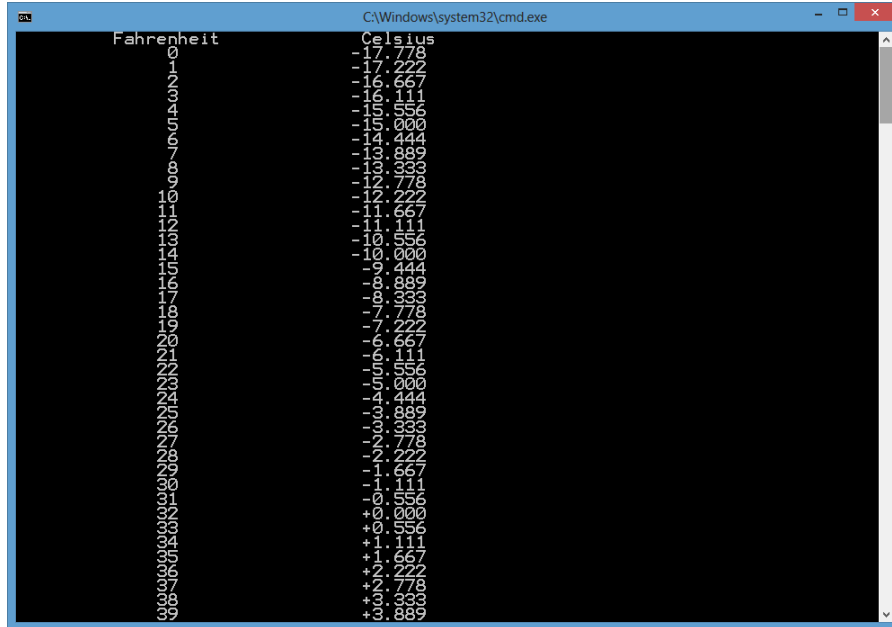
*The Cube class is similar. Write its prototype and its definition*



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

$$\text{celsius} = 5.0/9.0 * (\text{fahrenheit} - 32)$$

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



Fahrenheit	Celsius
0	-17.778
1	-17.222
2	-16.667
3	-16.111
4	-15.556
5	-15.000
6	-14.444
7	-13.889
8	-13.333
9	-12.778
10	-12.222
11	-11.667
12	-11.111
13	-10.556
14	-10.000
15	-9.444
16	-8.889
17	-8.333
18	-7.778
19	-7.222
20	-6.667
21	-6.111
22	-5.556
23	-5.000
24	-4.444
25	-3.889
26	-3.333
27	-2.778
28	-2.222
29	-1.667
30	-1.111
31	-0.556
32	0.000
33	+0.556
34	+1.111
35	+1.667
36	+2.222
37	+2.778
38	+3.333
39	+3.889

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"
```

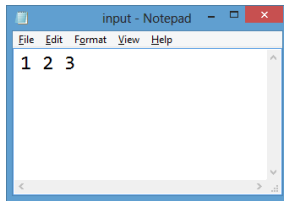
```
        << "numbers in infile.dat\n"
```

```
        << "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)*

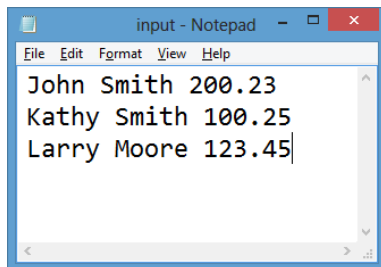
Run the program and see the results.

Open up the outfile.dat and see the result

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

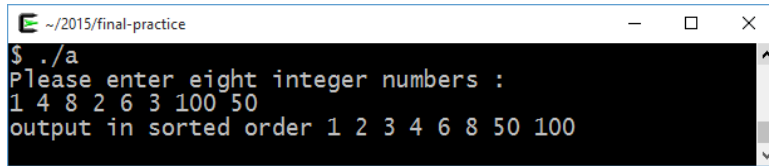
Say this is the input



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

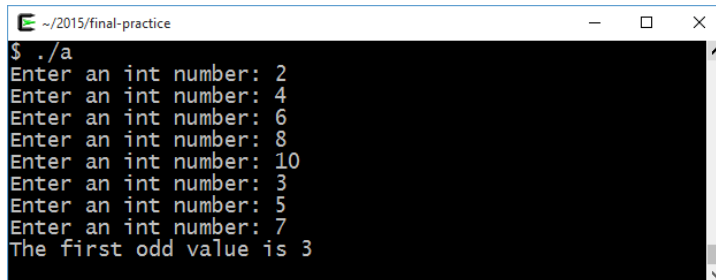
```
C:\Windows\system32\cmd.exe
220.253
110.275
135.795
Press any key to continue . . .
```

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.



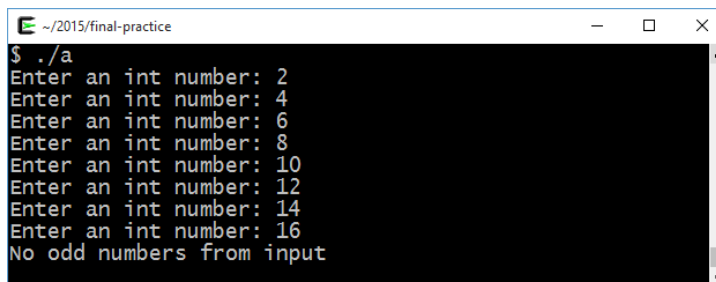
```
~/2015/final-practice
$ ./a
Please enter eight integer numbers :
1 4 8 2 6 3 100 50
output in sorted order 1 2 3 4 6 8 50 100
```

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



```
~/2015/final-practice
$ ./a
Enter an int number: 2
Enter an int number: 4
Enter an int number: 6
Enter an int number: 8
Enter an int number: 10
Enter an int number: 3
Enter an int number: 5
Enter an int number: 7
The first odd value is 3
```

If none are odd



```
~/2015/final-practice
$ ./a
Enter an int number: 2
Enter an int number: 4
Enter an int number: 6
Enter an int number: 8
Enter an int number: 10
Enter an int number: 12
Enter an int number: 14
Enter an int number: 16
No odd numbers from input
```