Final Assigment:

1. Write a program that takes an operation followed by two operands and outputs the result. For example:

   + 100 3.14

   * 4 5

   Read the operation into a string called operation and use an if-statement to figure out which operation the user wants, for example, if (operation=="+"). Read the operands into variables of type double. Implement this for operations called +, -,*,/, plus, minus, mult, and div with their obvious meanings.

2. Make a vector holding the ten string values "zero", "one", ... "nine". Use that in a program that converts a digit to its corresponding spelled-out value; e.g., the input 7 gives the output seven. Have the same program, using the same input loop, convert spelled numbers into their digit form; e.g., the input seven gives the output 7.
3. Write a function that finds the smallest and the largest element of a vector argument and also computes the mean and the median. Do not use global variables. Either return a struct containing the results or pass them back through reference arguments. Which of the two ways of returning several result values do you prefer and why?
4. Design and implement a Money class for calculations involving “reais” and “centavos” (cents) where arithmetic has to be accurate to the last cent using the 4/5 rounding rule (.5 of a centavo rounds up; anything less than .5 rounds down). Represent a monetary amount as a number of cents in a long, but input and output as “reais” and cents, e.g., R$123.45. Do not worry about amounts that don't fit into a long.
5. Refine the Money class by adding a currency (given as a constructor argument). Accept a floating-point initializer as long as it can be exactly represented as a long. Don't accept illegal operations. For example, Money*Money doesn't make sense, and R$1.23+USD5.0 makes sense only if you provide a conversion table defining the conversion factor between Reais (R$) and U.S. dollars (USD).
6. Write a program that produces the sum of all the whitespace separated integers in a text file. For example, "bears: 17 elephants 9 end" should output 26.
7. Template drill:
   a. define `template<class T>struct S{T val;};` Make `val` private
   b. Add a constructor, so that you can initialize with a `T`
   c. Define variables of types `S<int>, S<char>, S<double>, S<string>,` and `S< vector<int> >;` initialize them with values of your choice.
   d. Read those values and print them.
   e. Add a function template `get()` that returns a reference to `val`.
   f. Put the definition of `get()` outside the class.
   g. Add a `set()` function template so that you can change `val`.
   h. Add an `operator[]` with the same functionality of `get()` and `set()`.
   i. Provide const and non-const versions of `operator[]`.

j. Define a function `template<class T> read_val(T& v)` that reads from cin into `v`.

k. Use `read_val()` to read into each of the variables from c) except the `S< vector<int> >` variable.

8. Study and write an example of the Factory Pattern from Gamma et al. Your factory is should build Shapes based on a string identification of concrete shapes. Your main program is shown bellow. Write the code that allows this main to run.

```
void main()
{
   // Give me a circle
   Shape* obj1 = Shape::Create("circle");

   // Give me a square
   Shape* obj2 = Shape::Create("square");

   obj1->Draw();
   obj2->Draw();
}
```