CSCI 1730 - Programming Assignment 5 - 50 pts.
Due Date: Thursday, May 7, 2015

What you need to turn in:

- **Code listing:** A printed copy of your C++ code solution for each problem. (From CodeBlocks, save your code as a PDF and then print the PDF.) Remember to include your name on every page you turn in. Be sure to follow the "Code Style Guidelines" specified in the "Assignment Information and Guidelines" handout that was distributed on the first day of class (these guidelines are also available at the class web page).
- **Code files:** E-mail me a copy of your C++ code. Copy/paste all code into one e-mail message – please, do not send attachments. **Enter your name in the subject line of your email.**
- **Working in Pairs:** If you want to work with one other student in our class on this assignment, this is acceptable provided that both members of the pair make a contribution to the solution. If you decide to work in a pair, turn in only one copy of the solution – clearly identify the name of each pair member on everything that you turn in.
- **Late Assignments:** Assignments are due **by the end of class** on the specified due date (both the paper copies and the e-mail copies). Assignments turned in late will be assessed a 20% penalty per day late.

1. (10 pts.) Write a **C++ program** that will repeatedly perform the following input/output and analysis tasks for a text file containing a text passage written in English.

- It will prompt the user to enter a text file name containing a passage written in English and then open the file for reading.
- It will count the number of words in the file and display this count along with the average word length (average number of characters per word), the total number of word characters, the total number of punctuation characters, the length of the shortest word, and the length of the longest word.
- It will list all of the shortest words found in the file.
- It will list all of the longest words found in the file.
- It will search for a word specified by the user and report how many times the word occurs in the file.

The program will display all of the results on the screen as well as writing the results to an output report text file. The `main` function of your program should first ask the user for the name of the output report file and open the file for writing. Then it should repeatedly offer the user a menu of the above tasks. Each of the above tasks should be implemented in your program as separate functions.

Text processing notes:

- A word is defined to be a string of alphanumeric characters that is preceded and followed by whitespace or the beginning of the line or the end of the line. If such a string begins and ends with non-punctuation characters, then the string is a word. If such a string begins or ends with punctuation characters, the word embedded in the string will be the string with preceding and ending punctuation characters removed. In either case, punctuation characters in the middle of the string should not be removed and should be considered part of the word. So, the punctuation count should include those punctuation characters before or after a word, but not include punctuation characters embedded in a word.
- The length of a word followed by a punctuation character does not include the punctuation character.
- A punctuation character is a printing character other than whitespace, a digit, or a letter.

- The `cctype` function `ispunct` (see p.390 of text) might be useful for this problem
- Here are two string class functions that might be useful for this problem:
  - `length` (see p.401 of text)
  - `erase` (see description of remove on p.404 of text – it should be `erase` instead of `remove`)

File handling notes:

- An input file must be open before the user selects any of the above options other than the first one. So, the functions that handle the other tasks need to check to see if an input file is open before doing their task.
- To determine if the input file is currently open, you can use this function call which returns true if the file associated with fin is open:

```
fin.is_open()
```

- If the user selects the first option and an input file is already open, the function handling that task should first close the input file before attempting to open a new one.
- Each function that processes the input file will need to have the input file positioned to read at its beginning. Include this code to accomplish this task (assume that `fin` is the `ifstream` object that refers to your input file in the program):

```
fin.clear();
fin.seekg(0);
```

- `ifstream` and `ofstream` function parameters must be passed by reference.

You can download two text files (story.txt and story2.txt) for testing your program.

To see how this program will work, download the executable version of my solution (a5-1.exe) and then double-click on the filename to run the program.

2. (20 pts.) **Vectors** are math quantities that have both magnitude and direction. This problem involves the use of two-dimensional vectors – here are the math details:

- A **two-dimensional vector** is a directed line segment from the origin of the *xy*-plane to a point, (*a*,*b*). This vector is denoted <*a*,*b*>.
- **Vector addition/subtraction:** Two vectors can be added or subtracted resulting in a vector: <*a*,*b*> + <*c*,*d*> = <*a*+*c*,*b*+*d*> or <*a*,*b*> – <*c*,*d*> = <*a*–*c*,*b*–*d*>
- **Vector dot product:** Two vectors can be multiplied resulting in a number – this operation is called the dot product of two vectors: <*a*,*b*> * <*c*,*d*> = *ac*+*bd*
- **Vector/scalar multiplication:** A vector can be multiplied by a scalar (a number) resulting in a vector**;** for any scalar *r*, <*a*,*b*>*r* = <*ra*, *rb*>
- **Equality of vectors:** <*a*,*b*> = <*c*,*d*> means *a* = *c* and *b* = *d*.

This problem will also involve the use of **complex numbers**. Here is a review of complex numbers:

- A **complex number** is a number of the form $a + bi$ where *a* and *b* are real numbers and i is the imaginary unit, $i = \sqrt{-1}$.
- **Complex number addition/subtraction:** $(a+bi) \pm (c+di) = (a \pm c) + (b \pm d)i$
- **Complex number multiplication:** $(a+bi)*(c+di) = (ac-bd) + (ad+bc)i$
- **Complex number division:** $\frac{a+bi}{c+di} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$
- **Equality of complex numbers:** $a+bi = c+di$ means *a* = *c* and *b* = *d*.

Notice that both two-dimensional vectors and complex numbers essentially are defined by two numbers. Some of the arithmetic performed on each are the same (addition and subtraction), as is the meaning of equality; the other arithmetic operations are different.

Write a **C++ program** that will model both two-dimensional vectors and complex numbers using **derived classes**. Start with a base class, `pairs`, which should model a pair of floating point numbers and include overloaded addition, subtraction, and is-equal-to operators in addition to appropriate constructors. Then, include two derived classes, `complex` and `vect`. Both classes will inherit their data from the base `pairs` class. The `complex` class should include overloaded multiplication and division operators in addition to appropriate constructors. It should also include overloaded friend insertion and extraction operators. The `vect` class should include overloaded multiplication operators in addition to appropriate constructors. Note that it will need two overloaded multiplication operators – one to handle the dot product and a second to handle scalar multiplication. It should also include overloaded friend insertion and extraction operators.

Note: When you try to assign the result of an addition or subtraction of two of your derived class objects to another object of that class, you will get a compiler error message about not being able to do the assignment. To avoid this problem, include copy constructors in the `pairs` class and in each of the derived classes as follows (assumes the `pairs` data name are `a` and `b`):

```
pairs(const pairs& p): a(p.a),b(p.b) {}
complex(const pairs& p):pairs(p) {}
vect(const pairs& p):pairs(p) {}
```

To exercise your new collection of classes, use this main program:

```
const int SZ=6;
int main()
{
   int sel;
   complex cx[SZ];
   vect vx[SZ];
   do {
      cout << "Select an option - (1) Enter Complex Number Land\n";
      cout << "                   (2) Enter Vector Land\n";
      cout << "                   (3) Exit\n";
      cin >> sel;
      if(sel == 1)
         ComplexLand(cx);
      else if(sel == 2)
         VectorLand(vx);
      else if (sel == 3)
         cout << "Bye...\n";
      else
         cout << "Invalid input - try again\n";
   } while (sel != 3);
   char ch; cin >> ch;
   return 0;}
```

You will need to write the stand-alone functions `ComplexLand` and `VectorLand` – here is what each does:

- `ComplexLand` – this function receives an array of 6 complex numbers from `main` and will repeatedly prompt the user to select one of four options:

1. Enter a complex number
2. Display all complex numbers
3. Perform arithmetic or equality comparisons of complex numbers
4. Exit the function

When the user selects option 1, the function asks the user to enter a complex number, reads it in, and then asks where to store the number. When the user selects option 2, all of the complex numbers in the array are displayed. When the user selects option 3, the function will ask for the operation to be used. If the operation selected is an arithmetic operator, the function will next ask for the array locations of the numbers to be combined using the operator, as well as the array location of where the result should be stored. In all cases, the chosen operation is then performed. If the operation selected is the is-equal-to operator, the function will only ask for the array locations of the numbers to be compared and will display the result of the comparison.

- `VectorLand` – this function receives an array of 6 vectors from `main` and will repeatedly prompt the user to select one of four options:
  1. Enter a vector
  2. Display all vectors
  3. Perform arithmetic or equality comparisons of vectors
  4. Exit the function

  When the user selects option 1, the function asks the user to enter a vector, reads it in, and then asks where to store the vector. When the user selects option 2, all of the vectors in the array are displayed. When the user selects option 3, the function will ask for the operation to be used. If the operation selected is addition or subtraction, the function will next ask for the array locations of the vectors to be combined using the operator, as well as the array location of where the result should be stored. If the operation selected is vector-scalar multiplication, the function will ask for the array location of the vector to be multiplied, the scalar value, and the location of the result vector. In all cases, the chosen operation is then performed. If the operation selected is the is-equal-to operator, the function will only ask for the array locations of the vectors to be compared and will display the result of the comparison. If the operation selected is dot product, the function will next ask for the array locations of the vectors to be combined using the operator, and will display the dot product result.

To see how this program will work, download the executable version of my solution (a5-2.exe) and then double-click on the filename to run the program.

3. (10 pts.) Write a **C++ program** that uses virtual functions to input, calculate, and display some quantities associated with a variety of 2- and 3-dimensional objects. Design an abstract base class `Shape` from which you will derive classes `Rectangle`, `Circle`, `Triangle`, `Box`, `Can`, `Cone`, and `Ball`. The class `Shape` should contain pure virtual functions `Display`, `GetDimensions`, `Area`, `Perimeter`, and `Volume`. For each of the derived classes, write member functions `Display`, `GetDimensions`, `Area`, `Perimeter`, and `Volume` that will do the following:

- `Display` - output the type and dimensions for an object of the class.
- `GetDimensions` - get the dimensions for an object of the class.
- `Perimeter` - Calculate and display the perimeter of an object of a 2-dimensional shape class; do nothing for an object of a 3-dimensional shape class.
- `Area` - Calculate and display the area of an object of a 2-dimensional shape class; calculate and display the surface area for an object of a 3-dimensional shape class.

- `Volume` - Calculate and display the volume of an object of a 3-dimensional shape class; do nothing for an object of a 2-dimensional shape class.

Include appropriate constructors and any other member functions that you think are necessary.

Write a program driver that will allocate an array of 20 pointers to class `Shape` and then repeatedly prompt the user, up to a maximum 20 times, to enter a shape type, dynamically allocate memory for an appropriate shape object, and then prompt for and read the appropriate dimensions for the shape. After the input of shapes is complete, the program should then loop through the pointer array contents and display the shape type, dimensions, area, perimeter and/or volume for each of the input shape objects.

Formulas for the cone calculations:

Dimensions: Height and base radius (*h* and *r*); Surface area = $\pi r^2 + \pi r\sqrt{h^2 + r^2}$; Volume = $\frac{1}{3}\pi r^2 h$.

If you need any other formulas for the other shapes, let me know.
To see how this program will work, download the executable version of my solution (a5-3.exe) and then double-click on the filename to run the program.

4. (10 pts.) Write a **class template** `list` for modeling a list of five elements which will have values from one of four different data types (`int`, `float`, `char`, `Distance`). This class should support these list manipulation tasks:

- Initialize a list to "zero" values.
- Initialize a list to the values stored in a list of the same type.
- Allow for user input of a list.
- Display a list.
- Sort a list into ascending order.

Then, write a **template function** `demo`, which will accept a flag of the type data you would like and then create a list of that type to demonstrate the list class – it will do these tasks:

- Create a list of the selected type and display it, showing the initialized "zero" values it contains.
- Have the user enter values into the list and then display the values.
- Create a second list of the selected type, initializing it to the values stored in the first list, and then display the second list.
- Sort the values in the first list.
- Display the sorted list.

Program Requirements:

- To perform the sorting, use either a selection or bubble sort algorithm. See the document posted with the assignments, "sort and search.pdf", for details on these algorithms.
- The `Distance` class (used in numerous examples shown in class) is given below:

```
class Distance                          //English Distance class
   {
   private:
      int feet;
      float inches;
   public:
      Distance() : feet(0), inches(0.0)  //constructor (no args)
         {  }
      Distance(int ft, float in) : feet(ft), inches(in)
         {  }     //constructor (two args)
```

```
      Distance( float fltfeet )   //constructor (one arg)
         {                        //convert float to Distance
         feet = int(fltfeet);           //feet is integer part
         inches = 12*(fltfeet-feet);    //inches is what's left
         }
      bool operator < (Distance) const; //compare distances
      friend istream& operator >> (istream& s, Distance& d);
      friend ostream& operator << (ostream& s, Distance& d);
   };
bool Distance::operator < (Distance d2) const  //return the sum
   {
   float bf1 = feet + inches/12;
   float bf2 = d2.feet + d2.inches/12;
   return (bf1 < bf2) ? true : false;
   }
//-----------------------------------------------------------
istream& operator >> (istream& s, Distance& d)  //get Distance
   {                                            //from user
   cout << "\nEnter feet: ";  s >> d.feet;      //using
   cout << "Enter inches: ";  s >> d.inches;    //overloaded
   return s;                                    //>> operator
   }
//-----------------------------------------------------------
ostream& operator << (ostream& s, Distance& d)  //display
   {                                            //Distance
   s << d.feet << "\'-" << d.inches << '\"';    //using
   return s;                                    //overloaded
   }                                            //<< operator
```

Note: A `char` value can be initialized to zero – when displayed, no output appears.

To test your templates, try using the following `main` driver:

```
int main()
{
      int sel;
      bool end=false;
      int iflag=0;
      float fflag=0;
      char dflag=0;
      Distance Dflag;
      cout << "TEMPLATE DEMO PROGRAM\n";
      do{
            cout << "Enter list type (1=int 2=float 3=char 4=Distance 5=exit): ";
            cin >> sel;
            switch (sel)
            {
            case 1:
                  demo(iflag);
                  break;
            case 2:
                  demo(fflag);
                  break;
            case 3:
                  demo(dflag);
                  break;
            case 4:
                  demo(Dflag);
                  break;
            default:
                  end=true;
                  cout << "Bye...\n";
                  break;
            }
      }while(!end);
      return 0;
}
```