CSci 1113
Lab Exercise 7 (Week 8): Strings

Strings

Representing textual information using sequences of characters is common throughout computing. Names, sentences, text, prompts, etc. all need a proper representation. We've been using *string literals* since the first week of the course when we discovered how to write "Hello World" to the computer display.

In addition to representing non-numeric and qualitative information, string objects are frequently used in engineering and scientific applications to input and process large text files containing measurements, experimental test data, and so forth.

Comma Separated Value files (CSV)

One common format for representing large data files is the "comma separated value" (CSV) format. For example, if you have data in an Excel spreadsheet, it is a simple matter to output it to a file in CSV format. The CSV format is simplicity itself: each data element is stored as a *text* string separated from the succeeding value by a single comma (' , '). If the file data is tabular (rows and columns), the *row*s are separated using a single *newline* character ('`\n`'). This makes it possible to use a standard text-editor to view the contents of any CSV formatted file in order to determine its organization.

Getline()

Processing the data in a CSV formatted file requires that you identify and separate the individual *values* in each row. Individual rows are read using the `getline(`*stream*, *string*`)` function which returns the *entire* comma-separated row as a C++ string object. The row string must then be parsed by your program, separating values from the comma separators. This requires some fluency with manipulating strings which we will explore in this Lab Exercise.

Converting Strings to Values

The "values" that are obtained from each CSV string (row) are character strings. Before they can be used in a computation, they must be converted to numeric values (*floating-point* or *integer*). There are many clever ways to do this in C++, but the simplest method is to use the functions `atof` and/or `atoi`, found in the standard C++ library: <`cstdlib`>. These functions both take a *c_string* as an argument and return a `double` or `int` respectively. Recall that *c_strings* and C++ string objects are not the same thing! If the character string you wish to convert to a value is stored in a C++ string object, you first need to convert it to a *c_string* using the string class method `c_str`:

```
double foo = atof(somestring.c_str())
```

Mystery-Box Problem

Here is your next mystery-box problem. What is the output produced by the following code segment?

```
bool mystery(string fstr)
{  string rstr;
   for(int i=fstr.length()-1; i>=0 ;i--)
      rstr += fstr[i];
   return rstr == fstr;
}
```

Warm-up

1) **Returning Individual CSV Values**

Write a function named `nextString` that will return a single 'value' (i.e. substring) from a Comma Separated Value" string. Your function will take two arguments: a string variable containing a comma separated list of values, and an integer containing the starting index; your function should return a single string object with the value that starts at that index and ends right before the next comma ',' (do not include the comma in the returned string!) :

```
          string nextString(string str, int start_index);
```

If, however, the start index is after the last comma in the string, then the function should return the value starting at that index and continuing to the end of the string.

For example,

```
     cout << nextString("my,cat,ate,my,homework",3);
```

will print

```
     cat
```

and

```
     cout << nextString("my,cat,ate,my,homework",8);
```

will print

```
     te
```

and

```
     cout << nextString("my,cat,ate,my,homework",18);
```

will print

```
     work
```

When you have written your function, then write a short test program that will take in a simple comma separated string using `getline`:

```
     getline(cin,somestring)
```

and output values in the string using the `nextString` function.

Stretch

1) Split

Now, extend your test program by adding a second function named `split` that will identify *all* the individual values in a comma separated value string and return them in an array of string objects:

```
     int split(string str, string a[], int max_size);
```

Your function will take three arguments: a comma separated value string `str`, an array `a` of string objects, and the maximum size of the array. You must use the `nextString` function from Stretch Problem (1) to obtain each value in the string and store it in the array starting, with the first element. Return the total number of values stored in the array.

For example:

```
     string varray[VALUES];
```

```
int cnt = split("my,cat,ate,my,homework",varray,VALUES);
for( int i=0; i<cnt; i++)
   cout << varray[i] << endl;
```

Should produce:

```
my
cat
ate
my
homework
```

Workout

1) **Earthquake Data, Part 1**

Large repositories of recorded measurement data are available on the World Wide Web from a wide spectrum of applications such as stock market data, weather/climate data, etc.

Use your Web browser to view the following URL:

```
http://earthquake.usgs.gov/earthquakes/map/
```

This site is maintained by the US Geological Service and reports recent earthquake activity around the world. If you examine the upper left corner of this web page, you will see a “Click for more information” button that leads to 'download' link. Select this link and then choose 'CSV' as the file format. Now save the downloaded file in your home directory and then use a text editor to examine it.

The file contains a large quantity of information that is detailed in the first line of the file. We will only be interested in the magnitude of the earthquake and the place (the string indicating where the event occurred, not the latitude/longitude). As you examine the file, note that the `place` is actually *two* comma separated strings. The first begins with a double-quote and the second ends with another double quote. The "general" location we are interested in is the *second* of the two strings that make up the place description.

First, write a program that will read the file and output the categories from the first line, along with their relative location in the line:

Example:

```
0  time
1  latitude
2  longitude
3  depth
4  mag
5  magType
6  nst
7  gap
8  dmin
9  rms
10 net
11 id
12 updated
13 place
14 type
```

You should use the `getline()` function to read the first line of the file and the `split` function from Stretch 2 to create the value array.

2) **Earthquake Data, Part 2**

Now, modify your program to print out the magnitude and location of each earthquake in the file. Compare your program results to the actual file data to verify that it works correctly.

Challenge

Here is an extra challenge problem. You should try to complete the warm-up, stretch and workout problems in the lab. Try this challenge problem if you have extra time or would like additional practice outside of lab.

1). **Earthquake Report**

Modify the bubble-sort function from last week's lab exercise to sort the rows of a two dimensional array in descending order of the first element in each row. That is, as it sorts the first element in each row, it should move entire rows so that each row of data stays as a row throughout the sort.

Next, read the earthquake data and store only the magnitude and location in a two dimensional array with *n* rows and two columns.

Now sort the earthquake data in descending order of magnitude, and print out a list of all the earthquake magnitudes and their associated locations in order from the highest to the lowest. Note you will need to store the magnitude data as a string object in the array, but use its equivalent floating-point *value* for the comparison. Can you describe why?

[Hint: you can use the `atof() or stod()` function in your sort comparison.]