Project 3

Programming Assignment 3
Ebay Auction

Time due: 9:00 PM Saturday, February 10th

Before you ask a question about this specification, see if it has already been addressed by the Project 3 FAQ. And read the FAQ before you turn in this project, to be sure you didn't misinterpret anything. Be sure you also do the warmup exercises accompanying this project.

Introduction

According to Wikipedia, eBay Inc. is a multinational e-commerce corporation headquartered in San Jose, California that facilitates consumer-to-consumer and business-to-consumer sales through its website. eBay was founded by Pierre Omidyar in 1995, and became a notable success story of the dot-com bubble. Today, eBay is a multi-billion-dollar business with operations in about 30 countries. The company manages eBay.com, an online auction and shopping website in which people and businesses buy and sell a wide variety of goods and services worldwide. The website is free to use for buyers, but sellers are charged fees for listing items after a limited number of free listings, and again when those items are sold.

Based on this system, I'd like you read and process a string of auction bids as reported in the following format:

L# B# B+# W U

The list price is indicated by the letter L followed by a particular price. For simplicity sake, all prices will be in whole dollars (not quarters or pennies...). The list price shall immediately follow the letter L and has been abbreviated by # in the format string shown above. For example, the auction string

```
L100
```

lists an item for $100.

The first opening bid is indicated by the letter B followed by a particular price. Again, for simplicity sake, all prices will be in whole dollars. The opening can be any value greater than 0 and does not need to exceed the list price. For example, the auction string

```
L100B50
```

lists an item for $100 which then receives a bid for $50.

Following an opening bid, later bids are indicated by the letters B+ followed by a bid increment. Again, for simplicity sake, all prices will be in whole dollars. The bid increment can be any value greater than 0. For example, the auction string

```
L100B50B+10
```

lists an item for $100 which then receives a bid for $50 followed by another bid for $60 (which is 50 + 10).

Once an auction string is fully processed, if the total bid amount exceeds the list price, the item is sold for that highest total bid. For example, the auction string

```
L100B50B+10
```

lists an item for $100 which did not sell. In contrast, the auction string

```
L100B50B+10B+41
```

lists an item for $100 which did eventually sell for $101 (which is 50 + 10 + 41).

Finally, once the list price is established, a user can watch or unwatch the auction. The letter W establishes a watcher. The letter U unwatches an auction. For example, the auction string

```
L100B50B+10
```

concluded without selling the item and never had any watchers. In contrast, the auction string

```
L100B50WB+10WB+41W
```

lists an item for $100 which did eventually sell for $101 (which is 50 + 10 + 41) and concluded with a total of 3 watchers, one for each W in the auction string. Alternatively, the auction string

```
L100B50WB+10UWB+41W
```

lists an item for $100 which did eventually sell for $101 (which is 50 + 10 + 41) and concluded with a total of 2 watchers, one for each W in the auction string minus the U.

All of the following are examples of valid auction strings:

- L99B50WB+10UWB+40W (item sold for $100 with 2 watchers at the end)
- l99b50wb+10uwb+40w (lowercase letters are allowed)
- L100B99 (item did not sell)
- L100B50WWWWWWW
- L100B50WUWUWUWWWW
- L100B50WWUUB+10WWUU
- L100W (watching/unwatching allowed after the listing price)
- L100WU
- L100B50WU

All of the following are examples of invalid auction strings:

- L100 B50 W B+10 U W B+40 W (spaces not allowed)
- L100.50B50 (floating point values not allowed)
- L100B50UW (watching must occur before unwatching an auction)
- L100BW50 (watching can only occur after bid is complete)
- L100B50WBU+50 (unwatching can only occur after bid is complete)
- L100B50UUUUUUUUU (unwatching can only occur after watching)
- L100B50WWUUUUUUUUU (unwatching can only occur as many times as watching)
- WUL100 (listing price must always come first)
- L100L50 (listing price cannot be changed)
- L100B+50 (opening bid must come via B command, not B+)
- L100B50B+50B20 (opening bid only allowed once)

Your task

For this project, you will implement the following four functions, using the exact function names, parameter types, and return types shown in this specification. (The parameter *names* may be different if you wish).

`bool isValidEbayListingString(string auctionstring)`
This function returns `true` if its parameter is a well-formed auction string as described above, and `false` otherwise.

`bool listingSold(string auctionstring)`
If the parameter is a well-formed auction string, this function should process all the bids and determine if the total bid amount exceeds the list price. If the total final bid exceeds the list price, return `true` or `false` otherwise.

`int howMuch(string auctionstring)`
If the parameter is a well-formed auction string and the listing sold, this function should process all the bids and return the total final bid. If the auction string is not valid, return `-1`. If the auction string was valid but the item did not sell, return `0`.

`int watchers(string auctionstring)`
If the parameter is a well-formed auction string, this function should count up all the watchers that existed at the end of the auction. If the auction string is not valid, return `-1`.

These are the only four functions you are required to write. Your solution may use functions in addition to these four if you wish. While we won't test those additional functions separately, using them may help you structure your program more readably. Of course, to test them, you'll want to write a main routine that calls your functions. During the course of developing your solution, you might change that main routine many times. As long as your main routine compiles correctly when you turn in your solution, it doesn't matter what it does, since we will rename it to something harmless and never call it (because we will supply our own main routine to thoroughly test your functions).

Programming Guidelines

The functions you write must not use any global variables whose values may be changed during execution (so global *constants* are allowed).

When you turn in your solution, neither of the two required functions, nor any functions they call, may read any input from `cin` or write any output to `cout`. (Of course, during development, you may have them write whatever you like to help you debug.) If you want to print things out for debugging purposes, write to cerr instead of cout. cerr is the standard error destination; items written to it by default go to the screen. When we test your program, we will cause everything written to cerr to be discarded instead — we will never see that output, so you may leave those debugging output statements in your program if you wish.

The correctness of your program must not depend on undefined program behavior. For example, you can assume nothing about `c`'s value at the point indicated, nor even whether or not the program crashes:

```
int main()
{
    string s = "Hello";
    char c = s[5];   // c's value is undefined
    …
```

Be sure that your program builds successfully, and try to ensure that your functions do something reasonable for at least a few test cases. That way, you can get some partial credit for a solution that does not meet the entire specification.

There are a number of ways you might write your main routine to test your functions. One way is to interactively accept test strings:

```
int main()
{
    string s;

    cout.setf( ios::boolalpha ); // prints bool values as "true" or "false"

    for(;;)
    {
        cout << "Enter auction string: ";
        getline(cin, s);
        if (s == "quit") break;
        cout << "isValidEbayListingString returns ";
        cout << isValidEbayListingString(s) << endl;
        cout << "listingSold(s) returns ";
        cout << listingSold(s) << endl;
        cout << "howMuch(s) returns ";
        cout << howMuch(s) << endl;
        cout << "watchers(s) returns ";
        cout << watchers(s) << endl;

    }
}
```

While this is flexible, you run the risk of not being able to reproduce all your test cases if you make a change to your code and want to test that you didn't break anything that used to work.

Another way is to hard-code various tests and report which ones the program passes:

```
int main()
{
    if (!isValidEbayListingString("ua1+1"))
        cout << "Passed test 1: !isValidEbayListingString(\"ua1+1\")" << endl;
    if (!isValidEbayListingString("   "))
        cout << "Passed test 2: !isValidEbayListingString(\"   \")" << endl;
    …
```

This can get rather tedious. Fortunately, the library has a facility to make this easier: assert . If you `#include` the header `<cassert>`, you can call assert in the following manner:

```
assert(some boolean expression);
```

During execution, if the expression is true, nothing happens and execution continues normally; if it is false, a diagnostic message is written to `cerr` telling you the text and location of the failed assertion, and the program is terminated. As an example, here's a very incomplete set of tests:

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

```
…

int main()
{
    assert( ! isValidEbayListingString("UA1+1"));
    assert( ! isValidEbayListingString("    "));
    …
    cerr << "All tests succeeded" << endl;
}
```

The reason for writing one line of output at the end is to ensure that you can distinguish the situation of all tests succeeding from the case where one function you're testing silently crashes the program.

What to turn in

What you will turn in for this assignment is a zip file containing these two files and nothing more:

1. A text file named **ebay.cpp** that contains the source code for your C++ program. Your source code should have helpful comments that tell the purpose of the major program segments and explain any tricky code. The file must be a complete C++ program that can be built and run, so it must contain appropriate #include lines, a main routine, and any additional functions you may have chosen to write.
2. A file named **report.doc** or **report.docx** (in Microsoft Word format) or **report.txt** (an ordinary text file) that contains in addition **your name** and **your UCLA Id Number**:
   a. A brief description of notable obstacles you overcame.
   b. A description of the design of your program. You should use pseudocode in this description where it clarifies the presentation.
   c. A list of the test data that could be used to thoroughly test your program, along with the reason for each test. You don't have to include the results of the tests, but you must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.)

By February 8th, there will be a link on the class webpage that will enable you to turn in your zip file electronically. Turn in the file by the due time above. Give yourself enough time to be sure you can turn something in, because we will not accept excuses like "My network connection at home was down, and I didn't have a way to copy my files and bring them to a SEASnet machine." There's a lot to be said for turning in a preliminary version of your program and report early (You can always overwrite it with a later submission). That way you have something submitted in case there's a problem later. Notice that most of the test data portion of your report can be written from the requirements in this specification, before you even start designing your program.

Submission status

Submission status	No attempt
Grading status	Not graded
Due date	Saturday, 10 February 2018, 9:00 PM PST
Time remaining	12 days 8 hours
Last modified	-
Submission comments	▶ Comments (0)

Add submission