CS 7B - Fall '19 - Project 0 - Using the Debugger

Using the VS 2019 debugger is a great development tool, so we'll take some time at the start to learn how to use it. It helps you see what your program is doing while you run it. If you practice using it you'll find it very useful in finding errors in your code.

1. Open the nameHash.cpp program in a VS 2019 project and scroll down so you can see the entire function on your screen.

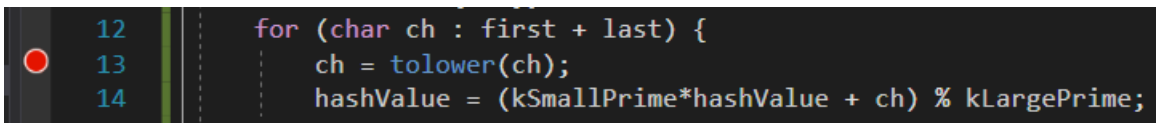
```
1  #include <string>
   using std::string;
3  #include <iostream>
   using std::cout;
5  using std::cin;

7  int namehash(string first, string last) {
     static const int kLargePrime = 16908799;
9    static const int kSmallPrime = 127;

11   int hashValue{ 0 };
     for (char ch : first + last) {
13     ch = tolower(ch);
       hashValue = (kSmallPrime*hashValue + ch) % kLargePrime;
15   }
     return hashValue;
17 }

19 int main() {
     string first, last;
21   cout << "Enter first and last name:\n";
     cin >> first >> last;
23   do {
       cout << "hash value: " << namehash(first, last) << '\n';
25     cout << "Enter first and last name:\n";
     } while (cin >> first >> last);
27 }
```

2. Move your cursor so it's just to the left of the number on line 13 and give a mouse click there. You should see a big red dot appear in the margin there. The is a "break point" for the debugger.



   When you run the program in debug mode, it allows you to pause and open up the debugger to see what's going on with the various variables and function calls involved in your program.

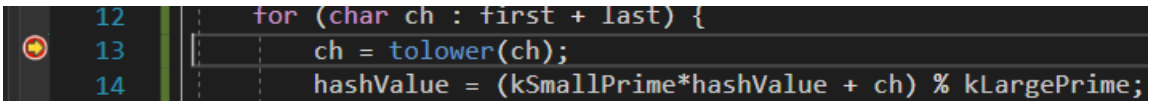
3. With the breakpoint set at line 13, run the program in debug mode. You can do this either from the "DEBUG" menu at the top of the window, or by clicking on the green "play" button at the top, or using the shortcut key "F5".

   This should open up a console window with the prompt:

```
Enter first and last name:
```

If the window is not visible, you should be able to find it in the tabs along the bottom of the screen.

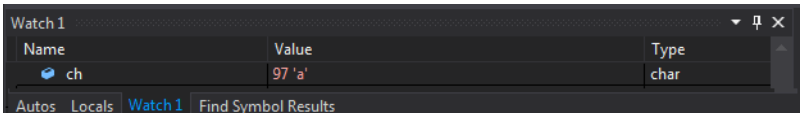
Enter "Ada Lovelace" as your first and last names. You should see the red dot on line 13 has been filled with an arrow, indicating that execution has paused at this point.



Now click on the "DEBUG" menu and observe the various options you have available for debugging. There are many options! Let's not execute any of them now and instead hover your mouse over the command `ch = tolower(ch);`, on line 13 until the green "Run to click" button appears, and then press the "Run to click" button. Repeat this for line 14. You should see the $\leq$ `1ms` diagnostic report for each line, indicating that these commands take very little time (less than one millisecond). Meanwhile, the yellow arrow in the margin will have advanced to line 14.

Common keyboard commands used to step through code include F10 (step over) and F11 (step into).

4. Notice in the lower left of the VS 2019 window is a panel that looks like this:



This is the "watch" window where you can monitor the value of variables in the active scope.

Another way to monitor variable values while the execution is paused is to hover the mouse over the variable until the watch dialog comes up and then click on the little pin in that dialog window to pin that watch to the screen at that location, which you can then drag to a more convenient place.

5. Next, click on the "call stack" tab in the panel on the lower right. Now, if your yellow arrow is indicating the execution is paused on line 14, then you'll see something like this as the first two entries on top of the call stack:

```
>   hash7B.exe!namehash(std::basic_string<char,std::char_traits<char>,
std::allocator<char> > first, std::basic_string<char,std::char_traits<char>,
std::allocator<char> > last) Line 14

>  hash7B.exe!main() Line 24
```

Double click on the "main()" on second entry in the call stack and you'll be taken to that section of the code (line 24). The yellow arrow stays where it was, but there's curvy white arrow in the margin for line 24, which, if you hover over it, will tell you "This is the next statement to execute when this thread returns from the current function." Notice that line 24 is the line that calls the `namehash()` function, which is how execution arrived at the breakpoint.

On the third line of the call stack you should see something like this:

```
>   hash7B.exe!invoke_main() Line 78
```

When you double click on that, you'll be taken to someplace like this:

```
static int __cdecl invoke_main()
{
    return main(__argc, __argv, _get_initial_narrow_environment());
}
```

Do you get the feeling that we're not in the Coachella Valley anymore? What happened? Whenever you compile a program, it will link in more code than what you wrote. If you dig too deep into the call stack, that's what you'll find. When debugging, you should just stick to the first entries in the call stack that involve code that you wrote. Double click on the top line of the call stack to return your position to the yellow arrow spot.

6. Go back to the panel on the lower left and click the "Locals" tab. You should see something like this:

```
<begin>$L0 0x0014f588 "AdaLovelace" char *
<end>$L0 0x0014f593 "" char *
<range>$L0 "AdaLovelace" std::basic_string<char,std::char_traits<char>,std::allocator<char>> &
ch 65 'A'char
first "Ada" std::basic_string<char,std::char_traits<char>,std::allocator<char> >
hashValue 0 int
kLargePrime 16908799 const int
kSmallPrime 127 const int
last "Lovelace" std::basic_string<char,std::char_traits<char>,std::allocator<char> >
```

Ignore the weird looking ones and you'll notice some nice, familiar names. For example, `kLargePrime` and `kSmallPrime` have the values we defined them with on lines 8 and 9. At this stage, you can see that `hashValue` is still 0.

As you step through the execution of the program, you'll see these values change, one step at a time.

7. Let's now examine the for-loop on lines 12-15. This is a range-based for-loop. You read this by saying "for every character `ch` in the concatenation (sum of strings) of `first + last`, starting with the first and progressing through to the last, execute the statements in the body of the loop."

Let's walk through the for loop step by step with F10. As you press F10 repeatedly (not so quickly that you can't observe the various changes as you go), watch the yellow arrow move and the values of the variables in the Locals panel change. Notice that entries in the "Values" column of the Locals panel turn different colors when they're changing.

In particular, observe how the values of `ch` and `hashValue` change. The expression assigned to `hashValue` is pretty complex-looking! You multiply the current `hashValue` (initially, zero) by `kSmallPrime`, add the ASCII current value of `ch` (from your `first + last` string) and then take the remainder after division by `kLargePrime`. This is what's iterated in the for-loop.

Also, observe how the ASCII values of `ch` change and where the `<begin>$L0` starts the substring of `AdaLovelace`.

8. The last thing to do is to email me at ghagopian@collegeofthedesert.edu with the subject line "debug" and three lines of text:
   (1) next-to-last hashvalue (the whole thing) for the input "`Ada Lovelace`"
   (2) the last hashvalue for the input "`Ada Lovelace`"
   (3) the hashvalue for your first/last name (this will be your alias on the "Grades" page, when that gets going.