Take Assessment: Exercise 4: Profiling Lab

Profiling Lab: Understanding Program Performance

In this exercise, you will modify an existing program to make it run faster. The program is not written for speed. You will find many things to optimize, but you should concentrate on optimizations that will make a significant impact on run time.

The program is all in one file called substitute.cpp. The purpose of the program is to perform string substitutions on a list of files. The program input is specified on the command line, for example:

substitute.exe replacements.txt file1.txt file2.txt ... fileN.txt

The first file, replacements.txt in this example, contains a list of substitutions to perform. Each line of this file specifies a substitution by giving a string to search for and a string to replace it by. The strings are in double quotes. The following is an example substitution file:

"the" "that"
"his" "her"

If you need to include a backslash (\) or quote (") in either string, you must escape the character by preceding it with a backslash. For example, to replace double quotes by backslashes, you would write the line:

"\"" "\\"

The remaining files on the command line are the files to be modified. The program reads each file, performs the substitutions one line at a time, and then writes the file. To perform a substitution, the program looks for an exact match to the first string within the file. The matching characters are replaced by the second string. Then the file is searched for another match to the first string. This match is performed on the new state of the file, so it may include characters from any previous substitution. In fact, if the replacement string contains the search string, the program will go into an infinite loop. When no more matches are found, the program moves on to the next line in the substitution file.

A good programming style would be to avoid reading in a whole file at a time because the files might be very large. For this exercise, however, you can assume that there is always enough memory to read the whole file.

Compiling substitute.cpp

Create a new project as a Console Application. Add substitute.cpp to the project. Before you compile it, you should go to the **Project Settings** dialog and, under **General, Microsoft Foundation Classes:**, select **Use MFC in a Shared DLL**. This is necessary because this program uses several MFC classes, including CString and CFile.

Running substitute.exe

Before running substitute.exe, you need some data to run it on. Unzip the data file called Data.zip, create a folder for it, and extract the files from the .zip archive. Since the program will modify some of these files, you will want to either save the .zip archive or save a copy of the files. (You will be running the program many times on this test data.)

To run substitute.exe, you can move it to a directory of your choice, create an MS-DOS command prompt window, and type in the program and command line. This gets tiring, so I recommend the following alternative: In the Project Settings dialog box, select the Debug tab, and set the Program argument fields to contain the command line arguments:

replace.txt call.cpp semantics.cpp math.cpp mach.cpp compiler.cpp

If you leave the Working directory blank, it will default to the program location, which will be something like C:\...*your_path*...\substitute\Debug. So before you run the program, copy the test files from wherever you put them to C:\...*your_path*...\substitute\Debug. Be sure to set up the **Program arguments** fields and put your test files in C:\...*your_path*...\substitute\Release when you test your optimized program using the **Release configuration**.

Note: I had trouble making the **Working directory** field work when debugging the Release configuration, so be careful if you deviate from these setup recommendations.

Profiling substitute.exe

Once everything is running (check the test files to see that the substitutions were applied), you are ready to start optimizing. The first step will be to use the profiler to find out where the program is spending its time and what it is doing with that time.

Depending on what compiler environment you are using, you may have different solutions.

Solution 1: on early version of Microsoft Visual Studio compilers:

An important way to make programs go faster is to turn on the optimizing compiler. Normally you do this by using **Set Active Configuration ...** under the Build menu to change from the Debug configuration to the Release configuration. Release is already set up to optimize your program. However, you will probably need to (once again) go to the **Project Settings** dialog and, under **General, Microsoft Foundation Classes:**, select **Use MFC in a Shared DLL** to apply this setting to the **Release configuration**. In addition, *to run the profiler on the **Release** configuration*, you need to:

- Open the **Project Settings** dialog box.
- Make sure your project and configuration (substitute, Win32 Release) are selected.
- Select the **Link** tab.
- Enable **Generate debug info** and **Enable profiling**.

If you are using NT 4.0 operating system, the **Profile...** option under the **Build** drop-down listbox may be grayed out. Microsoft provides a work-around in their support website. Search for Article ID *Q224382*.

Solution 2: on new version of Microsoft Visual Studio .NET compilers:

If you are using the new version of Microsoft Visual Studio .NET, you are suggested to activate a plugged profiler for profiling work. To do so, please unzip the ht_profiler_mod.zip file first, and read ReadMe.html under the zipped folder carefully, and follow the instructions in the file to finish your performance-measure work.

To enable the profiler, run the batch file enableprofiler.bat. This will register the profiler in the registry and setup an environment for using the profiler. Running any managed program under the environment will then be profiled.

Your Assignment

You should make a new version of substitute.exe and demonstrate, using profiling output, that it runs faster. You should be able to obtain at least a factor of 2 speedup (old run time divided by new run time). You do not have to use Microsoft Foundation Class objects, but given that these are well written and probably correct, you should only replace code that is doing unnecessary work as reflected in profiler measurements.

Submit two files:

1. Your modified substitute.cpp
2. A file that contains

1. a clear but concise description of what you observed before optimization. It should be substantiated by an empirical evidence of the profiler output.
2. the bottlenecks you noticed
3. the actions you took to address the bottlenecks, and the improvements you observed (again substantiated by empirical evidence)
4. reasons for why you did NOT attempt to optimize any more than you did.

To verify the correctness of your solution, compare the output files produced before making any source code changes with the output files produced after making source code changes. Since your optimization should not change the external behavior of the program, the corresponding output files should be identical. If the output files differ, your solution is incorrect. You can use the "comp" command in windows to check if the contents of two files differ.

Final Word

This project is for educational value only. However, you might be intrigued by the power of a string replacement engine like substitute.exe. Often, in software engineering and even web page maintenance, you will need to perform global replacement of variables, classes, and even misspellings. This is so common in large projects that there are many special tools to facilitate this job. Unix tools such as Find and SED and languages such as Awk and Perl make this kind of job simple. In fact, a SED script to replace strings in a file is not much more complicated than the "replace.txt" file read in by this project. With most tools, you can match patterns, allowing you do more powerful things such as substituting only when the search string is followed by any non-alphanumeric—that is, when the search string is not a prefix of some other identifier.

Few programmers are experts at all of these tools. Novices tend to ignore them and do things the hard way. Experienced programmers know they can learn to use another programming language, they teach themselves what they need to know on an as-needed basis, and they get the job done quickly.