Project 2

Evaluating Expressions by Parsing

Samuel Scalf

CMSC 330 · Alin Suciu

University of Maryland Global Campus

Table of Contents

Process and Lessons Learned

This was honestly one of the toughest assignments I've had to date. I have never used the C++ programming language, at least not that I can remember. A lot of the syntax and lexemes seemed familiar but were somehow different. It felt like going to a different country that kind of spoke English, but the meanings of key words and grammar were completely different. There were so many times going through this project that I would look up how to use a feature in C++ only to find out it does not exist. A great example of this belongs to enums. While it is possible to get the name and ordinal of an enumerator in Java, this is not possible in C++ as the "string" names are stripped away during compilation.

One silly challenge that got me caught up had to do with simple console I/O. The task was to read the input from a file instead of the console. I was unaware of what "cin" really meant, so I found myself perplexed as I sat in front of my computer staring at a blinking cursor waiting for the program to run its course. Meanwhile, the program was sitting idly waiting for me to provide it input. So, in the end nothing was happening. I could not for the life of me grasp what was going on. Even after creating output lines to know where the program was, that "cin" had me going in circles. For some reason I had gotten it into my head that you could use "cin" to parse the input. This was most likely cause by the first "cin >> paren" statement, followed by later "cin >> operation", "cin >> ws", etc. statements. This full well looked like a program that should at least read the single line equation using only '+' and parse it properly. Looking back, I understand it to be parsing one input at a time (i.e. "cin >> paren" expected a parentheses, then the later "cin" would prompt for the next part of the equation, and so forth.

Another big challenge I had was with include statements and header files. The first time I tried to create and compile this program, I did it with nano and g++, copying the Module 3 information as I went. This proved incorrect at least to some extent, because the program failed to compile stating that certain classes and/or functions had already been defined or had not yet been declared. I feel I essentially gave up on this one, because I ended up downloading and using Eclipse to help create and compile the program. I did learn a lot of useful ways to protect against the errors I was having with header files and the proper order of include statements. For example, I learned that if I include <string> in the cpp file before including the header file, then I don't need to include it in the header file. I also learned that by using a preprocessor if-statement to check if the header file had been defined, I could prevent it from being redefined.

My last great challenge was actual compilation. With Java, compiling the program was as simple as proving javac with the path of the source file containing the main() method. Javac would then compile other necessary files as needed. This is most definitely not the case with C++! Yikes. My program did not want to compile, let alone run. That's when I learned that C and C++ programs must be told in which order to compile and link the files. Eclipse was kind enough to create a few files, to include the primary "makefile," that I was able to export and study. This is where I learned most about C and C++ compilation. First, the source cpp files must be compiled into objects. Then those objects must be linked to each other and the libraries, if any.

There were so many times I said I hated C++, called it a horrible language, and so much more. After having overcome the challenges, however, I have grown to understand it and it has grown on me a bit. I still find Java to be much more, shall we say, beginner-friendly and more convenient, but I can see the allure of a C++ program. Recently, I started getting into open source software, which is usually distributed as source files. I've learned that this is actually really great. The compiler compiles the program for the architecture of your machine, whereas an already compiled program may not necessarily run out-of-the-box on your machine because the architecture may not match.

Compilation

I was able to successfully compile the entire program using the “make” command. Note that I am using g++ 9.3.0.

```
sam@sambuntu:~/Documents/cmsc330/projects/scalf-project2/module3$ make all
Building file: src/fileparser.cpp
Invoking: GCC C++ Compiler
g++ -O3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/fileparser.d" -MT"src/fileparser.o" -o "src/fileparser.o" "src/fil
eparser.cpp"
Finished building: src/fileparser.cpp

Building file: src/module3.cpp
Invoking: GCC C++ Compiler
g++ -O3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/module3.d" -MT"src/module3.o" -o "src/module3.o" "src/module3.cpp"
Finished building: src/module3.cpp

Building file: src/operand.cpp
Invoking: GCC C++ Compiler
g++ -O3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/operand.d" -MT"src/operand.o" -o "src/operand.o" "src/operand.cpp"
Finished building: src/operand.cpp

Building file: src/parse.cpp
Invoking: GCC C++ Compiler
g++ -O3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/parse.d" -MT"src/parse.o" -o "src/parse.o" "src/parse.cpp"
Finished building: src/parse.cpp

Building file: src/subexpression.cpp
Invoking: GCC C++ Compiler
g++ -O3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/subexpression.d" -MT"src/subexpression.o" -o "src/subexpression.o"
 "src/subexpression.cpp"
Finished building: src/subexpression.cpp

Building file: src/symboltable.cpp
Invoking: GCC C++ Compiler
g++ -O3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/symboltable.d" -MT"src/symboltable.o" -o "src/symboltable.o" "src/
symboltable.cpp"
Finished building: src/symboltable.cpp

Building file: src/variable.cpp
Invoking: GCC C++ Compiler
g++ -O3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/variable.d" -MT"src/variable.o" -o "src/variable.o" "src/variable.
cpp"
Finished building: src/variable.cpp

Building target: scalf-module3
Invoking: GCC C++ Linker
g++  -o "scalf-module3"  ./src/fileparser.o ./src/module3.o ./src/operand.o ./src/parse.o ./src/subexpression.o ./src/s
ymboltable.o ./src/variable.o
Finished building target: scalf-module3
```

Test Cases

Since this program parses equations from a text file with one equation per line, each line in my “default.txt” file will be considered a test case. The program will accept the path to a file as a command line argument should a different file be desired. The program outputs the equation followed by the result, or value, on the same line.

Test Equations

- (x + (y * 3)), x = 2, y = 5;
- (x + (y + (z * 3))), x = 3, y = 6, z = 7;
- (x / y), x = 7, y = 2;
- ((x / y) / z), x = 22, y = 3, z = 2;
- (x!), x = 0;
- ((x!)!), x = 1;
- ((x = y)!), x = 1, y = 0;
- ((x!) & (y | z)), x = 1, y = 0, z = 0;
- (x < y), x = 7, y = 4;
- ((x > y) & (z = y)), x = 2, y = 1, z = 1;
- (((x > y) & (z = y))!), x = 2, y = 1, z = 1;
- (x & y), x = 1, y = 2;
- (x & (y!)), x = 1, y = 2;
- (x : y ? z), x = 1, y = 0, z = 3;
- (x : y ? z), x = 1, y = 0, z = 0;
- ((x : y ? z)!), x = 1, y = 0, z = 0;

Output

I believe the output of the program to be correct. Some of the answers are mathematically incorrect due to integral division, or rather lossy conversion of a double to an integer. I did my best to safeguard the trimming of data by adding 0.5 to the answer. This causes a double with a decimal of 5 or greater to round up. Without this 7/2 = 3.5 would have been trimmed down to 3 as C++ simply truncates the decimal.

```
sam@sambuntu:~/Documents/cmsc330/projects/scalf-project2/module3$ ./scalf-module3 default.txt
(x + (y * 3)), x = 2, y = 5; Value = 17
(x + (y + ( z * 3))), x = 3, y = 6, z = 7; Value = 30
(x / y), x = 7, y = 2; Value = 4
((x / y) / z), x = 22, y = 3, z = 2; Value = 4
(x!), x = 0; Value = 1
((x!)!), x = 1; Value = 1
((x = y)!), x = 1, y = 0; Value = 1
((x!) & (y | z)), x = 1, y = 0, z = 0; Value = 0
(x < y), x = 7, y = 4; Value = 0
((x > y) & (z = y)), x = 2, y = 1, z = 1; Value = 1
(((x > y) & (z = y))!), x = 2, y = 1, z = 1; Value = 0
(x & y), x = 1, y = 2; Value = 1
(x & (y!)), x = 1, y = 2; Value = 0
(x : y ? z), x = 1, y = 0, z = 3; Value = 1
(x : y ? z), x = 1, y = 0, z = 0; Value = 0
((x : y ? z)!), x = 1, y = 0, z = 0; Value = 1
```