README File

Jesse Bevans, Kevin Watchuk,
Ryan Wenman, Thomas Richardson

Group names and contributions:

Jesse Bevans
Scanner:

-Symbol table
-Driver program(collaborated)
-Administration(collaborated)

Responsible for the symbol table, most importantly the logic behind inserting and comparing given tokens properly. Used a hash function found online, and implemented linear probing. On all other parts of program worked with the other members to make sure we didn't miss potential problems and squash bugs.

Parser:

Wrote out grammar to a proper LL(1) CFG form, generated first and follow sets. Wrote a basic skeleton for the parser using simple switch and if statements that ended up being the base for the parser.

Thomas Richardson
Scanner:

-Driver program
-Makefile
-Administration(collaborated)

During this phase of the project, I was assigned the makefile and driver program. From there I helped Ryan with Administration implementation. We worked together after this point for error checking and piecing together the rest of the files.

Parser:

-Makefile
-TestFiles
-Technical Document(Collaborated)
-Parser(Collaborated)

During this phase of the project I helped in initial design of the parser, and finding appropriate test files for testing our parser.

Ryan Wenman
Scanner:

-Administration
-Test Files
-README file
-Technical document

During this phase I made the Administration class. I also wrote up the README file and the technical document. As a team we worked heavily together on the later half of the project combining the classes and error checking together. I also went through the text book and made the unit testing cases.

Parser:

-README file
-Technical document

-Error checking

During the parser phase of the assignment I made the write ups and focused on helping put the code together.

Kevin Watchuk

Scanner:

-Scanner

-Symbol.h

-Token(NameToken,SymToken,Number Token)

During this phase of the assignment I wrote the Scanner, the base and derived classes of Token and the enumeration being used for symbols.In the later half of the assignment error checking and connecting disparate parts together was a group effort.

Parser:

Worked on adding error detection to the basic parser skeleton.

Together:

Scanner:

In the design, preparation, and implementation our group began meeting to outline the division of work for raw documents. This work is outlined above. Once those raw documents were completed we started collaborating together to piece together our respective documents. Fine tuning and proper communication between files took the bulk of the time due to proper testing and identification of shortcomings of design implementation. For the bulk of the project we worked heavily in a group setting together to allow for the greatest learning experience, and efficient implementation.

Parser:

During this phase we found ourselves very busy and on different time schedules for most of the duration of this phase. This lead to doing most of the work independently and taking what worked best and adding that to the first phase of the project. However similar to the first phase near the end we worked heavily together to finish up the code together as best as possible.

Names of files included:

Scanner:

makefile

.h files:

Administration.h
NameToken.h
NumberToken.h
Scanner.h
Symbol.h
SymbolTable.h
SymToken.h
Token.h

.cpp files:

main.cpp
Administration.cpp
NameToken.cpp
NumberToken.cpp
Scanner.cpp
SymbolTable.cpp
SymToken.cpp
Token.cpp

Text files:

tests/NoFailures.txt
tests/MaxFailures.txt
tests/EmptyTest.txt
tests/SymbolFailures.txt
Documentation/Technical Document.pdf
Documentation/README File.pdf

Parser:

Parser.h
Parser.cpp
LinearSearchTestFile.txt
ScopeTest.txt
SyntaxErrorsTest.txt
Test2.txt
Test4.txt
Test6.txt
Test7.txt
Test8.txt
Test9.txt
Test10.txt

How to compile and run code:

General Form -> ./plc <inputFile> <outputFile>
<inputFile> and <outputFile> should be replaced with the desired file names.

Please note: that files in other folders can be accessed, they just need to be properly identified in the name, for example tests/MaxFailures.txt instead of MaxFailures.txt.

Our program if downloaded from source needs to be compiled. Which can be done with the given makefile by calling <make> in the terminal. The makefile creates an executable file called <plc>, which for some computers to execute require the use of a period and a slash ./<executableFile>. Once plc has been compiled successfully, it requires two arguments. One for input and one for output, if one or the other is not provided, “incorrect arguments: x given, 2 expected” where x is any number of arguments.

Test Examples:

Scanner:

test1) ./plc tests/EmptyTest.txt output.txt
test2) ./plc tests/MaxFailures.txt output.txt
test3) ./plc tests/NoFailures.txt output.txt
test4) ./plc tests/SymbolFailures.txt output.txt

Parser:

Test 1) ./plc tests/LinearSearchTestFile.txt output.txt
Test 2) ./plc tests/ScopeTest.txt output.txt
Test 3) ./plc tests/SyntaxErrorsTest.txt output.txt
Test 4) ./plc tests/Test2.txt output.txt
Test 5) ./plc tests/Test4.txt output.txt
Test 6) ./plc tests/Test6.txt output.txt
Test 7) ./plc tests/Test7.txt output.txt
Test 8) ./plc tests/Test8.txt output.txt
Test 9) ./plc tests/Test9.txt output.txt
Test 10) ./plc tests/Test10.txt output.txt

Indicate if code is free of compile-time and runtime-errors:

Scanner:

According to all testing and error checking that we have done the code is error free during compile-time and run-time.

Parser:

According to all testing and error checking that we have done the code is error free during compile-time and run-time.

Approximate time (hours) spend on assignment part one:

Scanner:

Jesse Bevans:

Total (20 hours)

Thomas Richardson:

Total(15 hours)

Ryan Wenman:

Total(22 hours)

Kevin:

Total(29 hours)

Parser:

Jesse Bevans:

16 Hours

Thomas Richardson:

Total(16 hours)

Ryan Wenman:

Total(14 hours)

Kevin:

20 Hours