COEN 175

Phase 2 - Week 3

TAs

Stephen Tambussi

Email: stambussi@scu.edu

Office Hours: Tuesday 1:00 - 2:00PM / Wednesday 4:00 - 5:00PM (Heafey Atrium)

Jordan Randleman

Email: jrandleman@scu.edu

Office Hours: Tuesday 9:10-10:10AM / Thursday 9:10-10:10AM (Heafey Atrium)

Extra Help / Tutoring

Tau Beta Pi Tutoring

- Thursday 2-3pm Heafey Atrium

Phase 2 Overview - Syntax Analysis

Goal

- Write recursive-descent parser for Simple C that will print out the operation order of a given input program.

Last Week [THIS SHOULD BE DONE]

1. Disambiguate expression Grammar
2. Modify lexer.l to return tokens from tokens.h
3. Test new lexer with parser.cpp
4. Create parser.cpp and do expressions
    a. *Refer to Friday's lecture*
    b. *Test your expression parser against the "step0" files in "stepExamples.tar" under "labs" on Camino*
        i. *-:- ALSO -:- write 2 custom test cases of your own prior to asking for help with other grammar sections*
            1. *!!! THE TAs WILL ENFORCE THIS AS A RULE !!!*

This Week

1. Left factor translation unit
2. Implement the rest of the grammar in C++

Submission

- Submit a tarball of your cpps and makefiles in a folder called *phase2*
    - ***Must be the exact same "phase2.tar" file you checked with "CHECKSUB.sh"!***
- **Due 11:59PM on Sunday, January 29th**

Left Factoring Translation Unit

- Combine
  - Function-definition
  - Global-declaration
  - Global-declarator-list
- This should be your top level function that is called in your main
  - Check lecture slides

Step -1: Testing Your Lexer [**This should be done**]

- If you haven’t finished this you’re screwed
- Make sure you’re returning appropriate tokens from all the right places
  - Tokens shouldn’t be getting printed ANYWHERE in “lexer.l”
  - “return *yytext” should only be present only ONCE in your file!
  - If you aren’t returning ID, STRING, NUM, or CHARACTER anywhere, you’re doing it wrong.

Step 0: Writing the Expression Parser [**This should be done**]

- Remember to import lexer.h and tokens.h
  - *lexer.h* provides a "*report*" function to signal errors
    - Check out its function signature in *lexer.h* to understand how to use "*report*"!
- Write your main() and match() functions (need to declare a global int lookahead)
  - Read lecture slides for examples
- Write the code for expressions:
  - Start with algebraic binary (+, -, *, /, %)
  - Then prefix (!, &, ...)
  - The rest of expressions
- Remember to print out the output for each operator once the whole operation has been matched and completed
  - ***Phase 2 PDF has the required output for each operator! (Table 1)***

Step 1: Finishing up the Parser - Write “statement()”

- ***DIFFERENT FROM “statements()”***
  - Write “statements” & “declarations” as an empty function at first
    - These are tricky, leaving them out at first allows you to just test simple statements
      - Example: “if(a<b) x = y * z;”
- When testing only “statement()”, replace your “expression()” call in “main” to call “statement()”
  - This is crucial for obvious reasons

Step 2: Finishing up the Parser - Write “declaration()”

- ***DIFFERENT FROM “declarations()”***
- Write “specifier()”
- Write “pointers()”
  - Infinite pointers in theory
    - Write a single function to match them all
- Note that type declarations don’t have outputs
  - Example: “int* x;” doesn’t output “mul”
- Then write “declarations()”

Step 3: Finishing up the Parser - Write “statements()”

- Any lists of statements always ends with a “}”
  - Normal for “stepExamples/step3.c” to trigger “line 14: syntax error at end of file” at the end (due to how searching for a terminating “}” works with our lookahead)

Step 4: Finishing up the Parser - Write “functionOrGlobal()”

- Matches global declarations as well as function definitions
- Should be the first function called by your “main” function in a loop

Testing your code

Step-by-step examples provided on camino (**projects → 2 → stepExamples.tar**)

- Should replace contents with your own tests afterwards too (I've only provided a start)!

Phase 2 examples provided on camino (**projects → 2 → examples.tar**)

Run the following commands:

$ make clean all

$ ./scc < [***example***].c | diff - [***example***].out

=> [***example***] determined by what part you're testing in *examples*

Tips

- Check the lecture slides, they are VERY helpful
- Thoroughly test your code with your own examples
  - The provided examples will NOT test everything
  - Don’t run “scc” as a terminal prompt anymore, instead write entire custom simple C test files!
- Test your parser after *each and every* step in the slides
  - Make sure you’re calling the right function from “main” each time!
- **MIND YOUR SPELLING**
  - “statement” vs. “statements”, “declaration” vs. “declarations”, etc.!
- **READ THE SYNTAX RULES CAREFULLY**