COEN 175

Phase 2 - Week 2

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 - Syntax Analysis

Goal

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

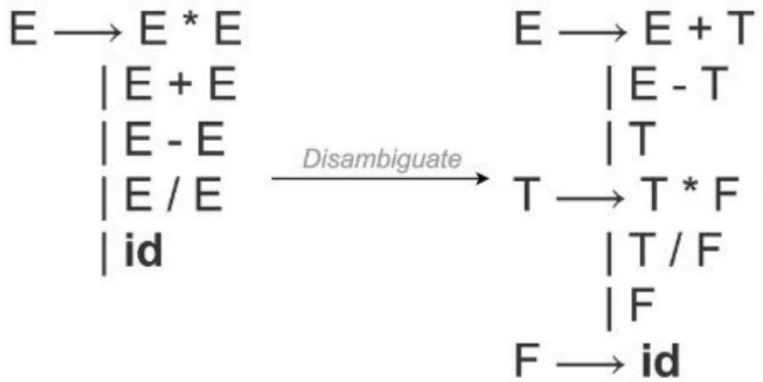
Objectives

1. Disambiguate expression Grammar
2. Modify lexer.l to return tokens from tokens.h
3. Test new lexer with parser.cpp
4. Continue parser.cpp for expressions (**on your own this week, after wednesday lecture**)

- Due 11:59PM on Sunday, January 29th
- Completing half this week
  - ***Seriously do this – you do not want to do it all in one week!***

1. Disambiguate Expression Grammar

Use the operator associativity/precedence table to disambiguate all of the expression grammar on the phase 2 assignment



2. Modify lexer.l

- Start from phase 1 solutions
  - Download solution.tar from camino (Project → 1)
- Edit **tokens.h** to include all tokens
  - All unique operators (e.g. +,-,/,%)
  - ID, num, string, error, done(make this constant 0)
  - All keywords (given)
- Modify **lexer.l**
  - Return appropriate token instead of printing them out
    - Ex. *return AUTO* instead of calling *printToken(“keyword”)*
  - Single char operators
    - return **yytext*

3. Test New Lexer

- Download parser.cpp from camino (labs → 2)
- Use the Makefile from lab 1 to compile parser.cpp
- Run phase 1 examples with the new lexer and parser to confirm that your new lexer is ready to go
  - Should not need to edit lexer again after this point
- Once you confirm that your lexer is working, save parser.cpp for next week

4. Writing the Parser (**After Wednesday Lecture**)

- Remember to import lexer.h and tokens.h
  - *tokens.h* provides a "*report*" function to signal errors
    - Check out its function signature in *tokens.h* to understand how to use "*report*"!
- Write your main() and match() functions (need to declare a global int lookahead)
  - Read lecture 4 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 (assignment doc has the required output for each operator)
- Goal is to have expression written by beginning of next lab section (can hold off on sizeof until next week if you'd like)

Tips

- Trust your disambiguation when you start writing code
- Don’t condense too much, it will work out better to be thorough than optimized
- Your functions can be written recursively, but it might be easier to write them iteratively except for prefix expressions (after wednesday)
- Postfix expressions can go forever but be careful what you keep parsing
- **READ THE SYNTAX RULES CAREFULLY**

Examples

```
[agigliot@linux10601 phase2]$ ./scc
a + b * d - c / d
mul
add
div
sub
[agigliot@linux10601 phase2]$
```

```
[agigliot@linux10601 phase2]$ ./scc
(a > b + c) * (1234 - a[b] || c && d)
add
gtn
index
sub
and
or
mul
[agigliot@linux10601 phase2]$
```

- **ctrl+d** to end input