CSEN 175

Week 1 - Phase 1

TA Office Hours

Jordan Randleman

Email: jrandleman@scu.edu

Office Hour: Thursday 9:10-10:10 AM (Heafey Atrium)

Nolan Anderson

Email: nranderson@scu.edu

Office Hour: Thursday 1:00-2:00 PM (Heafey Atrium)

Welcome to Compilers

Motivation:

- Understand compiler technologies
- Be able to deal with a very complex project across several files
- Gain experience with C++ programming

To Keep in Mind:

- Time intensive
- Detail intensive
- Dr. Atkinson will absolutely hold you back from graduation
    - Some people have had to take compilers 3+ times
- The lab is 40% of your lecture grade
- 100% doable if you fully exercise your cognition and resources :)
- You get a clean, correct phase from Dr. Atkinson each time

DO NOT CHEAT

(discussion on this)

How to Ask for Asynchronous Help

Do:

- Email Dr. Atkinson, Jordan, and Nolan all in the same email
- Include a *clean* (make clean) copy of your phase*N*.tar
- Include a detailed description of what’s wrong
- Include a detailed description of what you’ve done to try to solve the problem

Don’t:

- Email Dr. Atkinson, Jordan, or Nolan individually
- Send screenshots or individual files
- Just say “I have no idea what’s going on” and dump a ton of code

The “don’t”s are a sure-fire way to shoot your email to the bottom of the priority list!

Review of what the TA ***is*** and ***is not*** here for!

There are 3 kinds of questions you can ask:

1. I don’t understand how to build a compiler
   a. **That’s ok!** So long as you’ve checked the lab slides, PDF, and lecture slides–and you’re still confused as to how to structure your compiler–that’s what we’re here for! Ask away :)
2. I don’t understand C++
   a. **These are problems you have to figure out on your own.** You can Google each and every single one of these questions, C++ is a massive language with an even more massive amount of documentation on the web. You will get fired if you ask your employer how to use a string, where to find a function, or which method you should call on an object.
3. I don’t understand how to program
   a. **This is a call for introspection if you truly should even be taking compilers.** Everybody makes silly mistakes now and again, but consistently neglecting to run the code in your head, or ***not visualizing how the compiler is navigating the data*** as you write your program, is a recipe for a disastrous failure in this course.

Introduction

- Submissions
  - Tar file uploaded to camino
  - Typically due on Sundays at 11:59PM (-1pt for every minute late!)
- Advice
  - Keep up with lecture material
    - ***Or you will fail miserably and have to spend another $20K for an additional quarter at SCU***
  - Read the entire assignment (PDF & slides!) very carefully (most questions can be answered there)
  - Start early
    - ***A 2 week lab does not mean screw off for a week and a half then struggle at the last minute***
  - Write your own test cases
    - Dr. Atkinson's provided ones do ***NOT*** cover all possibilities – you must fully exercise your own code!
- Must run and compile on the **ROCKY 9** linux servers (Heafey 203/215)
  - Make sure that you use the ORIGINAL makefile when doing so! – all of your own changes must be reverted prior submission!

How our Compiler Works

- Read in from standard input
- Write to standard output

- Running
  - ./scc < *example.c* > output.out OR ./scc use cltr+d for eof
- Testing
  - diff output.out *example.out*
    - Program is correct if there is no output

Recommended Directory Structure

```
├── phase1
      ├── lexer.l
      ├── Makefile
├── examples
      ├── example1.c
      ├── example1.out
      ├── example2.c
      └── example2.out
├── CHECKSUB.sh
```

Submitting

1. Create tar file (phase1.tar) containing your source code
2. Run CHECKSUB to make sure tar file works
    a. ./CHECKSUB.sh phase1.tar examples.tar
        i. **If you get a permission error at this point run: chmod 770 CHECKSUB.sh**
3. Submit tar file to camino

- Run CHECKSUB before you leave lab today to make sure that you do not have issues running it later
- If CHECKSUB does not compile your submission, you will receive a 0.

Phase 1 - Lexical Analysis

- Write a lexical analyzer using flex
  - The Makefile will convert your “.l” file into a “.cpp” file for you, with the C++ code that flex created from your “.l” regex/C++ code pair instructions
- Print out all lexical constructs (tokens) recognized from standard in
- All whitespace, comments, illegal characters to be ignored
  - All rules on Assignment document
  - You will only be given lexically correct Simple C code in your tests for this phase

- Example
  - Standard in: **123**
  - Standard out: **integer 123**
- Due Sunday January 14th, 11:59PM

Hints 1

Comments

- Don’t bother trying to write a regular expression for a comment
- Write a function to scan a comment by hand
- Use yyinput() to read in a character

Strings

- Need to escape quote to properly match them
  - \” to match quotes
- Check lecture slides for more explanation
  - ***<u>This will be a recurring theme throughout all of this quarter!</u>***

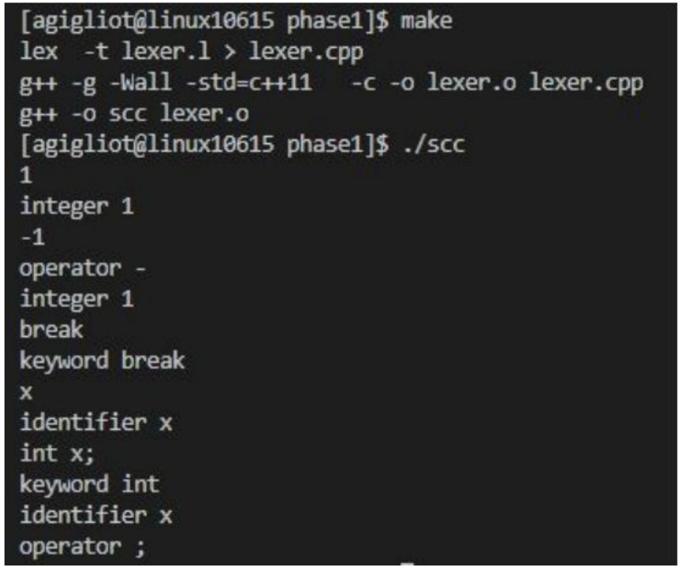
General

- Incrementally develop your solution: write one rule at a time and then test

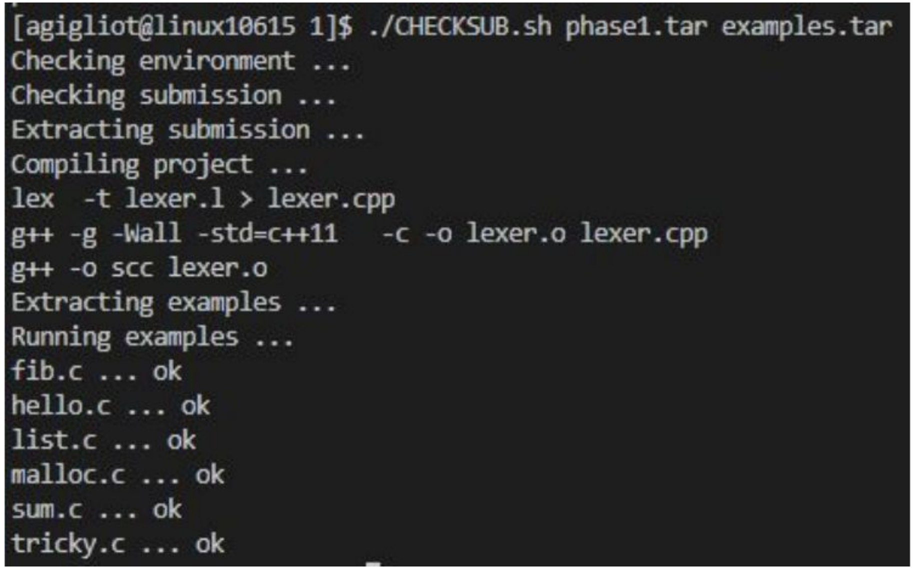
Hints 2

- Make sure to spell keywords correctly!
    - This mistake happens every quarter, and it's a ridiculous way to lose points
- Test incrementally! Every time you implement a new section, test it at the command-line, as done by the example in the following slide.
    - This is not the kind of class where you can just dump all the lab's code into a file then try debugging the entire lab at once: especially as the phases progress, incremental testing will be the only way to reliably (and sanely) fix bugs.
    - You are given a Makefile that will compile your "lexer.cpp" file into a "scc" binary
- Before submitting, ***MAKE SURE YOU PASS CHECKSHUB.sh***
    - Only do this after you have ***entirely completed*** the lab! CHECKSUB will ***not*** work on partially-completed labs.

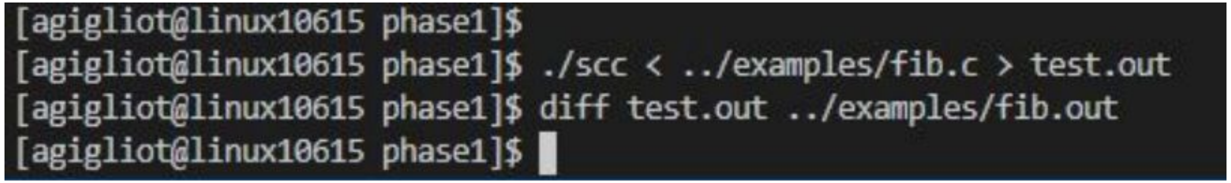
Examples



```
[agigliot@linux10615 phase1]$ make
lex  -t lexer.l > lexer.cpp
g++ -g -Wall -std=c++11   -c -o lexer.o lexer.cpp
g++ -o scc lexer.o
[agigliot@linux10615 phase1]$ ./scc
1
integer 1
-1
operator -
integer 1
break
keyword break
x
identifier x
int x;
keyword int
identifier x
operator ;
```



```
[agigliot@linux10615 1]$ ./CHECKSUB.sh phase1.tar examples.tar
Checking environment ...
Checking submission ...
Extracting submission ...
Compiling project ...
lex  -t lexer.l > lexer.cpp
g++ -g -Wall -std=c++11   -c -o lexer.o lexer.cpp
g++ -o scc lexer.o
Extracting examples ...
Running examples ...
fib.c ... ok
hello.c ... ok
list.c ... ok
malloc.c ... ok
sum.c ... ok
tricky.c ... ok
```



```
[agigliot@linux10615 phase1]$
[agigliot@linux10615 phase1]$ ./scc < ../examples/fib.c > test.out
[agigliot@linux10615 phase1]$ diff test.out ../examples/fib.out
[agigliot@linux10615 phase1]$
```