# COEN 175

Phase 6 - Week 9

# 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 or its alcove prior the conference room
  - Location depends on table availability

# How to Ask for Asynchronous Help

Do:

- Email Dr. Atkinson, Stephen, and Jordan 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, Stephen, or Jordan 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!***

# **Reminder:** the New Lab Policy

- From Dr. Atkinson: You can only attend the lab you registered!
  - No more staying past 5 to the next lab
  - No more going to other labs to ask questions till you understand.
    - Dr. A is really emphasizing the need for y'all to learn how to write/debug programs on your own.
- If a TA figures out you already went to another lab, you will be kicked out.
  - Anything else risks the TAs directly going against a directive straight from Dr. Atkinson, and could literally get us fired (believe it or not, this does happen 🙃).
- If you want to go another lab ***instead*** of (not as a supplement to) your registered class, ***make sure you get permission from the TA before lab***.

# Review of what the TAs ***are*** and ***are 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.

# High-Level Overview: Phase 6

**Goal**: Finish the compiler (generate code for expressions and statements)

**Week 9 Objectives (more on these in the following slides)**

1. Registers
2. Assignment
3. Add/Sub/Mul
4. Divide/Remainder
5. Relational and Equality Operators
6. Unary Operators

**Submission**

- Submit a tarball of your cpps and makefiles in a folder called *phase6*
- Worth 20% of your project grade
- Due ***Saturday***, March 18th by 11:59 PM

# Objective 1: Registers

- Register.h/Register.cpp given!
- Write Helper Functions in generator.cpp:
  - static void assign(Expression *expr, Register *reg)
    - From lecture
    - Delete the existing “assign” macro
    - *Most* functions end with `assign(this, resultRegister);`
  - static void load(Expression *expr, Register *reg)
    - From lecture
    - Delete the existing “load” function
  - static Register *getreg()
    - From lecture

# Objective 2: Modify Assignment to use registers

- Generate right
- Load right (into a register)
- Move right into left (don’t forget to use suffix())
  - “suffix()” is provided to you in the code
- Unassign all registers afterwards
  - This frees up the registers to be allocated for other expression compilations
- **Compile and test!**

# Objective 3: Add/Sub/Mul

- Write Add/Sub/Mul::generate(). These all have the same pattern.
- General structure
  - Generate left and right
  - Load left *(if not in register)*
  - Run the operation
  - Unassign right register
  - Assign left register to this expression
- **Compile and test!**
- No special case for pointers since amounts already scaled in checker.cpp
- **NOTE: divide/remainder are different and covered on the next slide**

# Objective 4: Divide/Remainder

- Write Divide/Remainder::generate()
- General structure
  - Generate left and right
  - Load left into rax
  - Unload rdx `load(nullptr, rdx`
  - Load right into rcx
  - Sign extend rax into rdx
    - Use “cltd” if result is size 4, else use “cqto”
  - idiv (with proper suffix) right; don’t forget to call Dr. Atkinson’s provided “suffix()”!
  - Unassign left and right register
  - Assign this to correct register
- Division result in rax, remainder result in rdx
- **Compile and test!**

# Objective 5: Relational and Equality Operators

- Write LessThan/…/Equal::generate()
- General structure
  - Generate left and right
  - Load left
  - Compare left and right
  - Unassign left and right
  - Assign this to a register
  - Store result of condition code in byte register
  - Zero-extend byte to long (using “movzbl”)
- Condition opcodes in lecture
- **Compile and test!**

# Objective 6: Unary Operators - Not/Negate

- Not
  - cmp* $0, _expr
    - * is the result of suffix()
  - sete _register->byte()
  - movzbl _register->byte(), _register
- Negate
  - neg*
    - * is the result of suffix()
- For both of the above
  - Start with generate and load expression
  - End with assign result expression to register

# Checking your code

- **$ scc < file.c > file.s**
- **$ gcc file.s file-lib.c**
- **$ ./a.out**

- Make -lib files if you want to use operations you haven’t implemented yet
- You don’t need to change any report() since those go to stderr
- Make sure you are sending your generated code to stdout
- Run with CHECKSUB.sh before submission
    - See the lab PDF for more details on how to check the assembly that GCC generates to compare it against yours (and the caveats inherent to doing so!)

# Tips

- Add comments via “#” for your assembly!
- For instructions that require a size suffix, use Dr. Atkinson’s provided suffix helper functions in generator.cpp
  - string suffix(unsigned long size)
  - string suffix(Expression *expr)
- Recompile your code frequently to make sure it still works
  - *Anytime you make changes to Tree.h/cpp do a **“make clean all”***
- Since you are overriding functions, you don’t need to have the other completed to compile
- _type will give the resulting type of the expression
- Don’t forget to assign resulting expression “this” at end
- ***NO changes in parser!***
- ***Check the lectures!***
- Check your output with the gcc using the -S flag
  - This will generate more optimal code than yours most likely, worry about correctness

# Tips Continued

- Call::generate() is done
  - *Note that* Dr. Atkinson's "Block" asserts that all registers should be empty at the end → **if you get this error that means you aren't deallocating registers properly!**
- Assignment will only work for ints and longs this week
  - Dereferences and chars will be handled next week
- Comparing 2 longs will yield an int
- **If you want more to work on before next week, try implementing these:**
  - Labels
  - Logical &&, Logical ||
  - If statements
  - Loops