COEN 175

Phase 4 - Week 7

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.

Quick Notes Regarding Last Week's Work:

- When printing error messages about "sizeof", write "sizeof" as the operator
  - Not "Sizeof", not "SIZEOF", etc.
- Write logically independent checks as different functions!
  - ADD and SUB should be DIFFERENT FUNCTIONS
    - They have different pointer semantics!
  - Don't put all the unary checks in a single function
    - Dear god what were you thinking they do completely different things
  - etc.
    - Note that *certain* checker functions can be put together, (MUL and DIV for example), but if it isn't immediately obvious to you why, just don't bother–keep things separate.
- When the PDF mentions use of "void pointer" (e.g. in ADD, SUB, etc.), it specifically refers to a single-indirection void pointer scalar
  - e.g. the type created by: Type(VOID,1)

High-Level Overview: Phase 4 Week 2

Goal: Create a Type checker

Week 7 Objectives (more on these in the following slides)

1. Type Check Function Calls (in primaryExpression)
2. Use the “lvalue” Variable!
    a. Involves setting it to “true” or “false” as needed
    b. Also includes use in various checker functions
3. Modify “statement”, “statements”, and “assignment” in parser.cpp
    a. Will involve adding checker functions to checker.cpp

Submission

- Submit a tarball of your cpps and makefiles in a folder called *phase4*
- Worth 20% of your project grade
- Due **THIS** Sunday, February 26th by 11:59 PM

Objective 1: Type Check Function Calls in PrimaryExpression

Pro-Tip: Make sure the ID you matched is *actually* a function type!

- Else report one of the error messages and return an error Type()

Two Cases:

1. Function has only been declared
    a. This means that any number and *virtually* any type of arguments are valid (since there isn't an established definition of which types of/how many arguments the function accepts yet).
    b. *Still have to check that all the types can be promoted to a predicate type!*
2. Function has been defined
    a. This means that the function has a strict requirement for ***how many*** and ***which types*** of arguments make for a valid call.
    b. Make sure that all of the arguments are *compatibleWith* the function's parameters!
    c. Remember that *parameters* in your function type is a POINTER!
        i. Access value in a vector pointer: **`vectPtr->at(index)`**

In All Cases:

- If the function object OR any of the arguments are of type ERROR, return an error type.

Objective 2: Use the “lvalue” Variable

1. Set “lvalue” to true/false as needed across your expression parser, as dictated by the rules in the assignment PDF.
    a. Should be true when:
        i. Matching variables in “primaryExpression” (check the PDF for specifics on this)
        ii. Indexing
        iii. Dereferencing
    b. See last week’s slides for where to write the code that sets the lvalue to “true”/”false” for generic binary and unary operations (in parser.cpp).
        i. Immediately after the call to your checker function!
2. Use the lvalue in the “checkAddress” checker function too
    a. See the assignment PDF for the rules pertaining to such

Objective 3.1: Modify “statement”

- Parsing “return”:
  - Pass down return type (not function type) as parameter to statement/statements
    - Passed down initially from “globalOrFunction”
    - Example in “globalOrFunction”:

```
openScope();
defineFunction(name, Type(typespec, indirection, parameters()));
match(')');
match('{');
declarations();
statements(Type(typespec, indirection));
closeScope();
match('}');
```

  - In “statement”: Check if the defined return type and the returned expression type are compatible!
- Statement still returns void

Objective 3.2: Modify “statements”

- Still returns void
- See previous slide for what it should accept as an parameter

Objective 3.3: Modify “assignment”

- Assignment just takes in left-side lvalue and expression types
- checkAssignment(left,right,lvalue)
  - Check if left is an lvalue
  - Check if left and right are compatible types
  - Does not return a type

Tips

- Read carefully to translate the rules from english to C++ code
  - Individually they are not complicated logic
- Recompile your code frequently to make sure it still works
- Check for error types first thing in checker functions
  - This includes “checkAssignment”!
- Run your code on CHECKSUB.sh before submitting
- Check your code on each operator before moving on
  - Not every operator is checked by the examples Dr. Atkinson gave you this week, ***make sure to write extensive test cases on your own to try and tease out each and every single error message from your checker functions!***
- **READ THE SEMANTIC RULES CAREFULLY**
  - ***!!! Especially true for phase 4 !!!***