COEN 175

Phase 3 - Week 4

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

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***.

Main Objectives for ALL of Phase 3

- You’re given a working solution for phase 2
  - Download these solutions to your machine, then immediately rename the directory to be “phase3” to avoid issues with moving files around
- Make the Symbol, Scope, & Type classes
- Modify your parser
- Write a checker

High-Level Overview: Phase 3 Week 1

Goal: Create a symbol table

Week 4 Objectives (more on these in the following slides)

1. Put cout statements in parser.cpp to denote where scopes open and close
2. Write openScope() & closeScope() skeleton functions in checker.cpp/h (create these files)
    a. Put the aforementioned scope cout statements in them
    b. Replace parser.cpp's print statements with calls to these functions instead
    c. These rest of these skeleton function bodies will be filled in during Week 5
3. Modify parser.cpp to pass around function/variable type information as needed
    a. Many of these changes were mentioned in the lecture slides, e.g. to declarator, etc.
4. Write the Type class
5. Create skeleton defineFunction(), declareFunction(), declareVariable(), and checkIdentifier() in checker.cpp/.h
    a. Add calls to the above skeleton functions in parser.cpp
    b. These skeleton functions will be filled in during Week 5

Submission

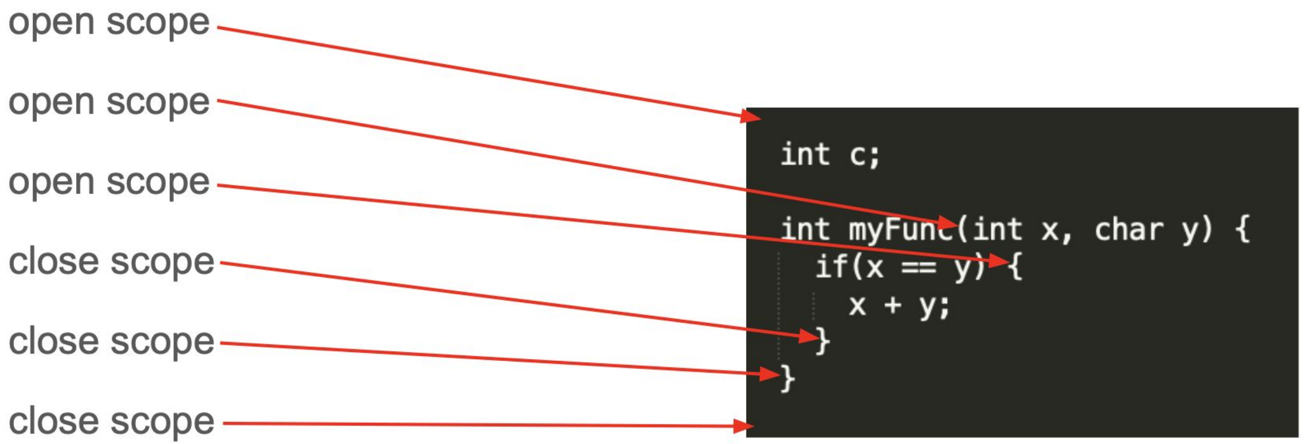
- Submit a tarball of your cpps and makefiles in a folder called *phase3*
- Worth 20% of your project grade
- Due Sunday, February 12th by 11:59 PM

Objective 1: Print where scopes open/close in parser.cpp

Add the following to parser.cpp where scopes start/end respectively:

- cout << “open scope” << endl;
- cout << “close scope” << endl;

Example:



```
int c;

int myFunc(int x, char y) {
  if(x == y) {
    x + y;
  }
}
```

Objective 2: Write skeleton openScope() and closeScope()

- Create Checker.cpp/h
- Add in “void openScope()” and “void closeScope()” to them
- Add in the print statements from the last slide to these function bodies
- Replace the scope print statements in parser.cpp with calls to these functions (that are now doing the printing themselves instead)

Objective 3: Modifying the Parser

- Return value from specifier(), pointers()
- Print out type information in parameter(), parameters()
    - *You'll return values from them once you write the Type class!*
- Pass in value into declarator()
    - *Account for changing your global declarator parsing logic to capture needed info too!*
- **Some of these were given in class**
- **Before** matching ID:
    - Capture the name of the id being matched that's stored in lexbuf
        - lexbuf is a std::string that associates to the textual input that created the current lookahead token (**HENCE "match" WILL CHANGE THE CONTENTS OF lexbuf**)
- **Before** matching the number length of an array:
    - Capture the length of the array by converting lexbuf to an unsigned long
        - Google how to do this without throwing an exception

Objective 4: Writing the Type Class

Create Type.cpp/h

Type Class:

- Should store the following
    - What the specifier type is (array, error, function, or scalar)
    - How much indirection there is
    - If an array, note the length
    - If a function, parameter information
- Overload the == and != operators for equality checking
- Overload the ostream operator <<
- **Type class was written in lecture**

Parameters can be thought of as a vector of Types:

- Hint: typedef is a thing that exists

Add return values to parameter() and parameters() now!

- After creating Type objects within them

Objective 5: Tracking variables/functions in Checker.cpp

Add the following skeleton functions to checker.cpp/h:

- void defineFunction(const std::string &name, const Type &type)
- void declareFunction(const std::string &name, const Type &type)
- void declareVariable(const std::string &name, const Type &type)
- void checkIdentifier(const std::string &name)

Call them in parser.cpp as appropriate

- Make sure to pass in the names and type info they require as arguments!

In the declare/define checker functions:

- Write the following to test your program's type parsing/identification:

  cout << name << ": " << type << endl;

  ^ prints the type via its overloaded "operator<<"

Testing Phase 3 Week 1 Outputs

- Phase 3 week 2 will focus on outputting errors E1-E5 to stderr
  - All stdout output will be ignored by CHECKSUB.sh
- BUT we can still use stdout to debug *this week's* program!
  - Can print out the openScope/closeScope results, as well as variable type information
  - Example stdout outputs for sample programs can be found on camino in Files > labs > 4 > outExamples.tar
    1. Note that your outputs do not need to exactly match the output in these files, rather they are a general guide as to what your output should look like
       - For example, we don't care about "int* p" vs "int *p" vs "int * p"
         - Hence don't do a diff, rather compare the files by looking at their contents
       - The number and order of "open/close scope" messages should be the same
       - The number and order of general variable/function type information should match

TIPS

- **A lot of code is/will be provided in class**
- ***Include new files in the Makefile after you create them!***
  - **Please dear God do this because holy crap I can already tell this is going to be an issue**
  - **Affects Checker, Type, etc.**
- You should include boolean member functions to check for each type
- Parameter information can be a vector of Types
- **READ THE SEMANTIC RULES CAREFULLY**