COEN 175

Phase 4 - Week 6

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.

Simple C Typing Review

Type Promotions

- CHAR can be promoted to INT
- ARRAY OF “T” can be promoted to POINTER TO “T”

Basic Type Groups

- Numeric
  - Type T is numeric if it is “int” or “long” after promotion
- Predicate
  - Type T is a predicate if it is numeric or “pointer to T” after promotion
- Pointer
  - Type T is a “pointer to T” if T can be promoted to such

Type Compatibility

Types are compatible if:

- They are both numeric
- They are both pointers of the same type T
- One is a pointer to T and the other is a pointer to VOID

NOTE:

- Remember to promote both types prior checking compatibility!

Lvalues vs. Rvalues

Lvalue:

- value that can appear on the (L)eft hand side of an assignment
  - E.g. they can STORE MEMORY – think “variable”, etc.

Rvalue:

- value that can ***only*** appear on the (R)ight hand side of an assignment
  - E.g. they ***CANNOT*** STORE MEMORY – think of the “1” integer literal, etc.

High-Level Overview: Phase 4 Week 1

Goal: Create a Type checker

Note: You're given a working solution for phase 3

- Download these solutions to your machine, then immediately rename the directory to be "phase4" to avoid issues with moving files around

Week 6 Objectives (more on these in the following slides)

1. Modify expression functions in parser.cpp (don't bother performing lvalue checks this week!)
2. Modify primary expression function in parser.cpp
3. Add and implement checker functions for parsed expression types
    a. This includes creating abstraction functions in Type class

Submission

- Submit a tarball of your cpps and makefiles in a folder called *phase4*
- Worth 20% of your project grade
- Due Sunday, February 26th by 11:59 PM
    - ***PEOPLE ARE STILL TRYING TO DO 2 WEEK LABS IN 1 WEEK. STOP IT. YOU'RE BEING CRAZY AND LOOK LIKE A CLOWN.***

Objective 1: Modify Expression Functions in parser.cpp

- Have them return the Type of resulted expression
- Pass boolean lvalue in by reference
- Example for expression() provided in the assignment doc
- Don’t bother performing lvalue checks this week
  - You’ll do this next week
  - Make sure to declare and pass in “bool lvalue” from statement when you call “expression()” though (just to make your code compile)

Objective 1: Modify Expression Functions in parser.cpp

=> Note that the “cout” statements are pre-provided in the phase3 solutions!

Binary Operator Example

```
static Type expression(bool &lvalue)
{
    Type left = logicalAndExpression(lvalue);

    while (lookahead == OR) {
        match(OR);
        Type right = logicalAndExpression(lvalue);
        cout << "check ||" << endl;
        lvalue = false;
    }
    return left;
}
```

Unary Operator Example

```
static Type prefixExpression(bool &lvalue)
{
    if (lookahead == '!') {
        match('!');
        Type left = prefixExpression(lvalue);
        cout << "check !" << endl;
        lvalue = false;
        return left;

    } else if (lookahead == '-') {
```

Objective 2: Modify Primary Expression Functions in parser.cpp

- Return Type object based on which case it hits
  - Identifier gets its type from its Symbol table entry
  - *On your own after lab prior week 2*:
    - Finish matching numbers, strings, and parenthesized expressions
      - You'll do function calls next week
      - Make sure to check the NUM value bounds with these macros to determine whether to return a LONG or INT type for number literals
        - Catch the literal value in lexbuf, turn that into a long variable, compare that long variable against the macro ranges in the above link to determine whether to return LONG or INT

Objective 3 Overview: Check Parsed Expression Types

Goal: Implement checker functions for parsed expressions

- Some of these will be/have been provided in class
- Remember to add their prototypes to checker.h
- Remember to handle error types passed as arguments
- Checker functions will verify given types are correct & report() error msgs otherwise (see lab PDF)

Add calls to the checker functions in parser.cpp as appropriate

- Should replace the “cout” statements in expression functions
- Checker function should take in operands as parameters (see pictures below)

You should implement the checker functions for operators the order of the following slides.

- *Be extremely careful to **follow the rules** from the **assignment PDF** document!*

Binary Operator Example

```
Type right = logicalAndExpression(lvalue);
left = checkLogicalOr(left, right);
lvalue = false;
```

Unary Operator Example

```
Type left = prefixExpression(lvalue);
left = checkNot(left);
lvalue = false;
return left;
```

Objective 3.1: if, while, for, ||, &&, !

The checker functions for these operators check for predicate types, hence implement the following Type class methods too:

- bool isPredicate() const;
  - Depends on:
    - bool isNumeric() const;
    - bool isPointer() const;
- Type promote() const;
  - *Remember to promote types prior to checking them in checker functions!*

Refer to the assignment PDF for the rules associated with these Type methods!

Objective 3.2: * (mul), /, %, - (neg)

The checker functions for these operators check for numeric types

- Make sure that the result is LONG if ***either*** argument is LONG

Objective 3.3: <, >, <=, >=

The checker functions for these operators check that their arguments both have the same type ***and*** that the type is either numeric or pointer

- *Which Type class method could do this for you?*

Objective 3.4: ==, !=

The checker functions for these operators check for type compatibility, hence add the following method to the Type class too:

- bool isCompatibleWith(const Type &that) const;

Refer to the assignment PDF for the rules associated with this Type method!

Objective 3.5: +, - (sub)

The checker functions for these operators must account for special cases with pointers

- and exceptions with void pointers

Objective 3.6: * (deref), [], &

Two of the checker functions for these operators have to handle exceptions with void pointers.

All of the checker functions work with the “lvalue” boolean.

- Don’t worry about this too much this week though, you’ll implement it in week 7

Objective 3.7: sizeof

- The checker function for sizeof must check that the type argument given actually has a size
  - *Think this through: which type object can’t have its “size” measured?*

Objective 3.8: ()

- No checker function needed: just passes up the type of the expression parsed between parentheses

TIPS

- Read carefully to translate the rules from english to C++ code
  - Individually they don’t have complicated logic
- Recompile your code frequently to make sure it still works
- Checker functions should account for promoted types
- ***!!! READ THE SEMANTIC RULES FROM LAB PDF CAREFULLY !!!***
  - *This is **ESPECIALLY** true this week!*