COEN 175

Phase 6 - Week 10

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 10 Objectives (more on these in the following slides)

1. Write Label class
2. Expression::test()
3. While::generate()
4. Address::generate() and Dereference::generate()
5. Finish Assignment::generate()
6. Strings
7. Return::generate()
8. Finish the rest of the compiler (rest of statement generation + more)

Submission

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

Objective 1: Write Label Class

- Write Label.cpp and Label.h (add to makefile)
- Check lecture notes for code

```
class Label {
    static unsigned _counter;
    unsigned _number;

public:
    Label();
    unsigned number() const;
};

ostream &operator <<(ostream &ostr, const Label &label);
```

Objective 2: Write Expression::test()

- New virtual function!
- Check lecture notes for this

Objective 3: While::generate()

- Check the lecture slides for this

Objective 4: Address::generate() & Dereference::generate()

- Address::generate()
  - Remember &*E == E
  - Use the helper function isDereference() for this case
- Dereference::generate()
  - Depending on the size, use the correct mov instruction
  - Remember that “suffix()” exists lol

```
void Address::generate()
{
    Expression *pointer;

    if (_expr->isDereference(pointer)) {
        pointer->generate();

        if(pointer->_register == nullptr)
            load(pointer, getreg());

        assign(this, pointer->_register);

    } else {
        assign(this, getreg());
        // leaq _expr, this
    }
}
```

```
void Dereference::generate()
{
    _expr->generate();
    if(_expr->_register == nullptr)
        load(_expr, getreg());

    // movl (_expr), expr
    // OR
    // movq (_expr), expr

    assign(this, _expr->_register);
}
```

Objective 5: Modify Assignment::generate()

- Modify Assignment::generate() to handle when LHS is a dereference

```
void Assignment::generate()
{
    Expression *pointer;

    _right->generate();

    if(_left->isDereference(pointer))
    {
        pointer->generate();

        // load pointer
        // load right
        // ...
    }
    else
    {
        // Case for Assignment that you should have already written

        // load right
        // ...
    }
}
```

Objective 6: Strings

- All strings belong in .data section of assembly file
  - Suggested approach: store all strings in a global map data structure so we can easily refer to the strings by Label later on. Denote static storage for strings at the end of the assembly file
- Modify String::operand and generateGlobals() to accommodate strings
- Create global map from string to label (Google “std::map”): `static map<string, Label> strings;`
- String::operand()
  - Add this prototype to the String class in Tree.h
  - Implement this function in generator.cpp
  - Check if string exists in map already
    - If not, create a label and add to map
  - Write the label to the “ostr” arg in the operand function
- generateGlobals()
  - Print out assembly for all strings in the map
  - Use escapeString() to properly print out strings
    - “escapeString()” is included in string.h

```
# Assembly file
.
.
.
    .data
.L10:    .asciz    “hello”
.L15:    .asciz    “coen175”
```

Objective 7: Return Statement

- Generate return value and load into “rax”
- Jump to return label
  - Hint: the name of the current function is a global variable declared at top of generator.cpp
    - Hence jump to the “<function-name>.exit” label
    - <function-name> is stored in generator.cpp’s global “funcname” variable
- Unassign register bound to _expr

Objective 8: Finish the rest of the Compiler

- Suggested order from here on out:
  - Cast::generate()
    - More help for this on the next slide
  - LogicalAnd::generate(), LogicalOr::generate()
  - For::generate()
    - similar to “while”
  - If::generate()
    - similar to “while”
  - More test() functions (these are ***OPTIONAL*** optimizations – don’t do until you can pass CHECKSUB!)
    - Logical Expressions
    - Comparative Expressions
    - Not: _expr->test(label, !ifTrue);

Objective 8.2: Cast::generate() Help

```
void Cast::generate()
{
    Register *reg;
    unsigned source, target;


    source = _expr->type().size();
    target = _type.size();
    _expr->generate();

    if (source >= target) {
      // think of what to do here! (need to get "_expr"
      // into a register and assign "this" node to it)
    } else {

      if(source == 1 && target == 4) {
        // movsbl
      } else if(source == 1 && target == 8) {
        // movsbq
      } else { // if(source == 4 && target == 8)
        // movslq
      }
      // bind "this" to the register with the casted value
    }
}
```

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

- Remember to emit comments with “#” to make your assembly code readable
- Recompile your code frequently to make sure it still works
  - DO ***make clean all*** AFTER EVERY TIME YOU ADD A VIRTUAL FUNCTION
- Since you are overriding functions, you don’t need to have the others 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