Lab 8

COEN 175 Compilers

Overview For Lab 8

Goal

- Begin generating code for storage allocation

Submission

- Submit a tarball of your cpps and make files in folder called phase5
- Due Date: Sunday February 27th

Phase 5 Outline

1. Add offset to Symbol Class
2. Type::size()
3. Procedure::generate()
4. Overload Ostream
5. Statement::generate()
6. Assignment/Call::generate()
7. .comm directives

1. Add Offset to Symbol class

- To store the offset for symbols on the stack
- _offset public member to Symbol class
- Initialize to 0 in constructor

2. Write Type::size()

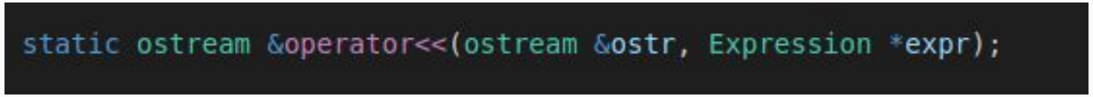
- Initially defined in Type.h
- Int is signed and needs 4 bytes
- Char is signed and needs 1 byte
- Pointers and callbacks need 4 bytes
- Array needs to account for length of array
    - Lower indices are at lower addresses
- No structures this week

3. Write Procedure::generate()

- Create generator.cpp
- These are function definitions
- Use symbols array from body scope in Procedure
  - Contains parameters THEN declarations
- Allocate (set offset) for parameters
  - First parameter 8 bytes up
  - The first parameter has the lowest address
  - Positive offset
- Allocate (set offset) for declarations
  - Negative offset
- _body->generate()
- Change the proc->write() in parser to be a proc->generate()

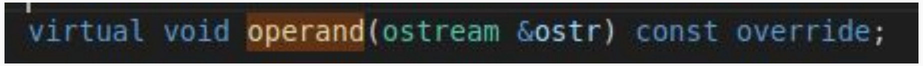
4. Overload Ostream Operator

- Add ostream declaration to top of generator.cpp



```
static ostream &operator<<(ostream &ostr, Expression *expr);
```

- Override operand function on Number and Identifier in Tree.h



```
virtual void operand(ostream &ostr) const override;
```

- Ostream should just call polymorphic operand method and pass ostream
- Number should cout $value (e.g. $4)
- Global identifiers should cout name (e.g. foo) and local identifiers should cout offset from base pointer (e.g. -4(%ebp))
    - Check if offset == 0 to differentiate

5. Override Statement::generate()

- Override in needed statement subclasses from Tree.h
- Simple::generate(), Block::generate() given in class
- Need to work on Assignment::generate() and Call::generate()
- Comments provided in Tree.h to help guide you
- Add virtual function overrides to Tree.h
- Implement in generator.cpp

6. Write Assignment/Call::generate()

- Assignment
    - Left will always be a scalar variable
    - Right will always be a literal
    - Use movl
- Call
    - Push _args onto stack (last _args pushed first)
    - Call function
    - Add to stack pointer to free arguments on stack

7. Output the .comm directives

- Generate the .comm directives for global variables
  - .comm name, size
- Write generateGlobals() in generator.cpp
  - Takes in scope (global)
  - Outputs .comm directives for all non-function symbols
- Use return from closeScope in main() in parser.cpp
- Add to generator.h

Tips

- Recompile your code frequently to make sure it still works
- Don’t forget to add your generator to your Makefile
- Since you are overriding functions, you don’t need to have the others completed to *compile*
- Only need one generate() call in parser, the rest will be called recursively
- Check your output with the gcc using the -S flag
    - This will generate more optimal code than yours most likely, worry about correctness

Checking your code

- $ scc < file.c > file.s 2> /dev/null
- $ gcc -m32 file.s [additional-source-files]
- $ ./a.out

- 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 before submission