CompilerDesign
Project 1-C
Simple calculator language — Code generation

1 Code generation

Convert the *abstract syntax tree (AST)* constructed during parsing into a sequence of assembly code. Uh, yeah, that's it.

2 Getting started

Begin by grabbing some starting source code at:

`bit.ly/cosc-371-p1c`

Extract the archive into a new directory. Only a few files, listed below, are included; all of the other Java source code should be copied from your Project 1-B solution.

- `Compiler.java`: Minor additions to `main()` so that the AST, the root of which is contained in the `Program`, is then called upon to generate code via a call to `toAssembly()`. The result is emitted into a `.asm` file.

- `Program.java`: The `toAssembly()` method is provided, emitting *stub code* to surround the code generated by walking the AST to have each element generate its own code. The code generated in `Program.toAssembly()` does three basic things:

  1. **Prologue:** Set up the beginning of the assembly file, defining symbols and beginning the code that will make up a `main()` function, within which all other code will be embedded.

  2. **Per-expression:** For each top-level expression in the program, pop its result from the stack and pass it to `printf()` so that we see the result of the evaluation.

  3. **Epilogue:** Complete `main()` by having it return. Also, generate a *statics* section that contains each expression as a formatting string as part of what is passed to `printf()` after each expression is evaluated.

- `compile.sh`: A script that, given the name of a Simple Calculator source code file (`.src`), will:

    1. Run our compiler on the source code, producing an assembly file (`.asm`). It also captures any output from the compiler in a log file (`.log`).

    2. Run the assembler (`nasm`) on that assembly file, producing an *object file* (`.o`).

    3. Run the linker (`ld`, via `gcc`) on the object file, producing an *executable* (no file extention).

3 Your assignment

Complete the code generation for our simple-calculator language. Specifically, add `toAssembly()` methods that traverse the AST, creating and returning (as a `String`) the assembly code corresponding to the program represented by the AST.

A good implementation of the code generator should:

- Correctly generate code that carries out the program.

- Emit code that a human can match (via comments embedded in the assembly) to the source.

To test your compiler on a program, use the `compile.sh` script. For example, if I have a Simple Calculator source code file that looks like this…

```
# Hello

4
(+ 3 1)
(* 20 5)
(* 20 (- 0 1))
(% 92 60)
```

…then I should be able to compile and run it and see the following…

```
$ ./compile.sh test.src
$ ./test
4 = 4
(+ 3 1) = 4
(* 20 5) = 100
(* 20 (- 0 1)) = -20
(% 92 60) = 32
```

4 How to submit your work

Copy/upload **all of the Java source code files for this project** in the `project-1-C` folder in your shared *Google Drive* folder for this course.

This assignment is due on Thursday, Sep-26, 11:59 pm.