Programming Assignment V  
Due Wednesday, December 4, 2002

1 Introduction

In this assignment you will implement a code generator for Cool. This assignment is the end of the line: when successfully completed, you will have a fully functional Cool compiler.

The code generator makes use of the AST constructed in PA3 and static analysis performed in PA4. Your code generator should produce MIPS assembly code that faithfully implements any correct Cool program. There is no error recovery in code generation—all erroneous Cool programs have been detected by the front-end phases of the compiler.

As with the static analysis assignment, this assignment has considerable room for design decisions. Your program is correct if it generates correct code; how you achieve that goal is up to you. We will suggest certain conventions that we believe will make your life easier, but you don’t have to take our advice. This assignment is comparable in size and difficulty to the previous programming assignment. Start early!

2 Files and Directories

To get started, create a directory where you want to do the assignment and execute one of the following commands in that directory. For the C++ version of the assignment, you should type

\texttt{gmake -f soffa/cs2210/html/cs2210/cool/assignments/PA5/Makefile}

This command will copy a number of files to your directory. Some of the files will be copied read-only (using symbolic links). You should not edit these files. In fact, if you make and modify private copies of these files, you may find it impossible to complete the assignment. See the instructions in the \texttt{README} file. The files that you will need to modify:

- \texttt{cgen.cc}
  This file will contain your code generator. We have provided an implementation of some aspects of code generation; studying this code will help you write the rest of the code generator. It includes a call to code that will build an inheritance graph from the provided AST. You can use the provided code or replace it with your own.

- \texttt{cgen.h}
  This file is the header for the code generator. You may add anything you like to this file. It provides classes for implementing the inheritance graph. You may replace or modify them as you wish.

- \texttt{emit.h}
  This file contains code generation macros. You may modify this file.

- \texttt{cool-tree.h} and \texttt{cool-tree-handcode.h}
  As usual, these files contain the declarations of classes for AST nodes. You can add field declarations to the classes in \texttt{cool-tree.h} or \texttt{cool-tree-handcode.h}. The definitions of the methods should be added to \texttt{cgen.cc}.
CS 2210  Compiler Design

- **cgen_supp.cc**
  This file contains general support code for the code generator. You will find a number of handy functions here. Modify the file as you see fit, but don’t change anything that’s already there.

- **example.cl**
  This file should contain a test program of your own design. Test as many features of the code generator as you can manage to fit into one file.

  As usual, there are other files used in the assignment that are symbolically linked to your directory or are included from ~/soffa/public/html/cs2210/cool/include/PA5. You should not modify these files. Almost all of these files have have been described in previous assignments.

### 3 Designing and Testing the Code Generator

You will need a working scanner, parser, and semantic analyzer to test your code generator analyzer. You may use either your own components or the components from coolc. By default, the coolc components are used. To change that, replace the lexer and/or parser and/or semant executable (which are symbolic links in your project directory) with your own scanner/parser. Even if you use your own components, it is wise to test your semantic analyzer with the coolc scanner, parser, and semantic analyzer at least once, because we will grade your semantic analyzer using coolc’s version.

You will run your code generator using mycoolc, a shell script that “glues” together the generator with the rest of compiler phases. Note that mycoolc takes a `-c` flag for debugging the code generator; using this flag merely causes cgendebug (a global variable) to be set. Adding the actual code to produce useful debugging information is up to you. See the project README for details.

There are many possible ways to write the code generator. One reasonable strategy is to perform code generation in two passes. The first pass decides the object layout for each class, particularly the offset at which each attribute is stored in an object. Using this information, the second pass recursively walks each feature and generates stack machine code for each expression.

There are a number of things you must keep in mind while designing your code generator:

- Your code generator must work correctly with the Cool runtime system, which is explained in the Cool Tour manual.

- You should have a clear picture of the runtime semantics of Cool programs. The semantics are described informally in the first part of the CoolAid, and a precise description of how Cool programs should behave is given in Section 13 of the manual.

- You should understand the MIPS instruction set. An overview of MIPS operations is given in the spim documentation, which is in the course reader and on the class Web page.

- You should decide what invariants your generated code will observe and expect; i.e., what registers will be saved, which might be overwritten, etc. You may also find it useful to refer to information on code generation in the lecture notes and portions of the text, primarily ASU Chapter 9.

### 4 Garbage Collection

To receive full credit for this assignment, you do not have to use a garbage collector (this is default). To receive extra credit, your generated code must work correctly with the generational garbage collector in
the Cool runtime system. The skeletons contain functions `code.select.gc` that generate code that sets GC options from command line flags. The command line flags that affect garbage collection are `-g`, `-t`, and `-T`. Garbage collection is disabled by default; the flag `-g` enables it. When enabled, the garbage collector not only reclaims memory, but also verifies that "-1" separates all objects in the heap, thus checking that the program (or the collector!) has not accidentally overwritten the end of an object. The `-t` and `-T` flags are used for additional testing. With `-t` the collector performs collections very frequently (on every allocation). The garbage collector does not directly use `-T`; in `cool` the `-T` option causes extra code to be generated that performs more runtime validity checks. You are free to use (or not use) `-T` for whatever you wish.

For your implementation, the simplest way to start is not to use the collector at all (this is the default). When you decide to use the collector, be sure to carefully review the garbage collection interface described in the Cool Tour. Ensuring that your code generator correctly works with the garbage collector in all circumstances is not trivial.

5 Spim and XSpim

You will find `spim` and `xspim` useful for debugging your generated code. `xspim` works like `spim` in that it lets you run MIPS assembly programs. However, it has many features that allow you to examine the virtual machine’s state, including the memory locations, registers, data segment, and code segment of the program. You can also set breakpoints and single step your program. Look at the documentation for `spim/xspim` in the course reader or in the course web page.

Warning: One thing that makes debugging with `spim` difficult is that `spim` is an interpreter for assembly code and not a true assembler. If your code or data definitions refer to undefined labels, the error shows up only if the executing code actually refers to such a label. Moreover, an error is reported only for undefined labels that appear in the code section of your program. If you have constant data definitions that refer to undefined labels, `spim` won’t tell you anything. It will just assume the value 0 for such undefined labels.

6 Handing in your assignment

You should follow the same procedure as done in the previous assignments.