

# Homework 5: Program Translation / IR generation

Instructor : Kiayias

Out Date:

11/15/2007

Due Date:

12/04/2007

---

## Objectives

The purpose of this homework is to translate a given source of C- to the appropriate assembly looking intermediate representation. The translation will be built by traversing over the abstract syntax tree in the same way that we performed the semantic analysis. While the translation will assume that the given source is semantically correct the semantic analysis component will be important and it will be executed prior to translation. This is due to the fact that in translation we will take advantage of the annotations of the abstract syntax tree that we have achieved during the semantic analysis phase. Each basic node class will provide the implementation of two methods `IRbuildstatic` and `IRgenerate` that will facilitate the translation. The translation, as in the case of analysis, will occur in two stages, the static data building stage (that is rather short as it doesn't need to traverse the whole abstract syntax tree) and the IR generation phase. In the generation phase the code will be synthesized in a bottom up fashion in the form of basic code blocks.

## Handout

To obtain your assignment, go to the class website in the handouts section and download `hw5-package.zip`. Similar to previous homeworks, this archive is a template project that you can import into Eclipse. It contains support code and basic structure. It is up to you to "fill in" the blanks. It includes plenty of support code as well as the C-grammar. The project now includes two new packages called `ir` and `irgen` that contain a number of support classes. In particular `ir` contains a number of classes that extend an abstract class `Instruction`. All IR instructions are formed with members of the `ir` package.

The `irgen` package on the other hand contains a collection of support classes that will be used in IR code generation. In particular it contains the abstract class `Location` and is used for the operands of `ir` instructions.

The class `Location` is extended in a number of classes including `Temporary`, `Immediate`, `RelativeSt`, `RelativeCl`, `Indexed` and `Label`. When locations are used as operands their semantics are as follows:

- `Temporary` locations : think of them as CPU registers. You are allowed an unlimited number of them for this homework.
- `Immediate` locations : they are constants.
- `RelativeSt` locations : they provide a `setOffset()` method and they are offset locations from a memory location signified as `[fp + offset]` (this notation will denote the contents of that memory location). In practice `fp` is the frame pointer.
- `RelativeCl` locations : they provide a `setOffset()` method and they are offset locations from a memory location signified as `[self + offset]` (this notation will denote the contents of that memory location). In practice `self` is the memory address where the current object is stored.

- **Indexed** locations : they are composite locations of the general form `base (offset)` where `base` is a location that signifies the base address and `offset` an offset to be added. Note that you may have compositions of such locations as in `[self + 4](8)` which means the address that is found after you add 8 to the contents of the address `self +4`.
- **Label** locations : these are labels used for branching and beginning of methods.

The IRgeneration will be building an object `Block` that will contain a vector of instructions. This will be the main output of your translation engine. Note that the block of instructions will be attached to a class object called `Code` that may contain many code blocks. At present it may contain at most two, the translated program and the block required to spawn the main method of C-- program.

The convention regarding the main method is that it should take no arguments and should always return type `int`. No system calls are implemented whatsoever (such as printing for example) except a call to `malloc` that allocates memory for the objects as we need it.

One of the delicate aspects of this homework is to handle the labels when composing instruction vectors that use branching (such as `for`, `while` blocks etc). In all IR generation there is a label called `firstlabel` that is flowing down as an inherited attribute during your AST traversal. Whenever you can find a place to put it you will have to assign it to a certain instruction. When this is not possible you will simply pass it back upwards using the `exitpoint` attribute of the block you are returning.

Compared to hw4, the package `sem` has been upgraded from the previous version of hw4 to account for the additional functionality that is required at this stage.

## To Do

Inside the package `ast` you will find all abstract syntax tree class that are referenced in `parser.cup`. In most classes the instantiation of the method `IRbuildstatic` and `IRgenerate` is missing; you will have to provide these methods. I have provided some comments inside each file that explain how you can solve them.

In the zip file you will also find a collection of 16 test files named `test1.cmm` to `test16.cmm` written in C--. These have to be translated correctly (the translations are also provided).

**Remark 1.** Feel free to toss away the grammar or any code that I provide to you and substitute it for the one you have built from homeworks #3-#4. Only minor modifications to your code will be required to do this.

**Remark 2.** There are a number of additional things one can do here and there to optimize the generation of the IR code. Any documented improvements will receive **extra credit**.

## Hand-in

Your solution should be an archive of the whole project (that should compile correctly and perform the translation as good as you can for as many of the test files as possible) and should include the following:

1. All the classes of the packages that you are required to fill in appropriately completed.

You submit your homework by midnight on the due date as a zip archive exported from eclipse. The e-mail address to submit is `c244fa07@cse.uconn.edu`.

Have fun!