UCT 2005 CSC305 Compilers

Practical Assignment 4: IR Generation

The IR Code Generation module of a compiler generates IR Trees corresponding to the Abstract Syntax Trees. The aim of this assignment is to perform some of these code generation tasks for a subset of the possible instructions, based on the Minijava language and the IR language recommended by Appel.

For example, if the input file includes:
```
  void test () { int i; i=i+1; }
```
then the output will include:
```
  someClass.test
     MOVE [TEMP[I], BINOP[PLUS,TEMP[I],1]]
```

Your main program must output the full IR tree for a file that is read from standard input – you ONLY have to write the code to generate IR sub-trees for NOT, ASSIGN and WHILE – the rest will be provided. Your output must make obvious what the structure and contents are, as this will form the primary basis for determining correctness. For simplicity, you may draw your trees in ASCII, similar to the AST assignment – this code is also provided.

Test your code with and provide output for the 2 Minijava files attached. In each case, use output redirection to capture the output and save it to appropriately-named files.

You may use any code available on the textbook's website, but are not required to do so. Use the grammar file and your AST from the previous tutorial. The model solution AST code from the previous tutorial will be made available after the final submission date for that tutorial – do use it! In addition, a comprehensive template is provided, with all supporting classes and the code for all methods that support this tutorial. You are REQUIRED to document all this code as well as your own.

Thus the requirements for this tutorial can be summarised as follows:

- write the ASSIGN code generator
- write the NOT code generator
- write the WHILE code generator
- document all code, including that in the routines/class in the provided template and your own routines

  Assume that all the test/sample programs will be error-free.

The template and your own code should make the following assumptions and simplifications:

- All memory access is defined by TEMP nodes, assuming that the decision to use registers or memory will be taken later. Variables with a known name are stored in a temporary named with the same string of characters. Return values are stored in a temporary called "return".
- Classes will contain only methods and no variables. There will be no inheritance.
- Do not generate code for stack manipulation or parameter passing. Assume this will be handled by a different part of the compiler (that manages the Frame).

- Assume the existence of system routines for output (system.print), allocating memory (system.memalloc), allocating new objects (system.newObject) and determining the size of memory allocated (system.memlength). None of the code you write ought to need these system routines.
- The CALL IR node has three parameters: an expression (usually TEMP) that refers to the memory where the object is stored, an identifier for the method to call and a list of expressions.
- All the test/sample programs will be error-free – you may do type-checking but it is not required.
- Sample output is provided also for the BubbleSort.java and TreeVisitor.java programs to indicate how you should generate your sub-trees.

Your assignment will be marked according to the following criteria:
Correctness (40%) (test programs generate correct output at each layer)
Documentation (25%) (comments within hand-written source and short report, 3 pages max)
Efficiency (15%)
Stress (20%) (hidden test programs generate correct output at each layer)