CSC488S Code Generation Templates

The purpose of requiring code templates is to help ensure that you have a complete and correct mapping from the project language to the machine and to help you plan for the code generator implementation in Assignment 5.

The code generation templates should describe in great detail how you plan to map the project language onto the CSC488S pseudo machine. You should describe your overall design strategy and give specific details on each of the categories listed below. Organize your report by the categories listed below, and not by the code generation action numbers.

For each of the items listed below - describe your strategy for implementing the item - give one or more examples of the machine code you expect to generate.

1. Storage

Describe your strategy for allocating storage for and addressing all kinds of variables and constants

a. variables in the main program
b. variables in procedures and functions
c. variables in minor scopes (e.g. '{' declaration statement '}')
d. integer and boolean constants (e.g. **'true'** and **'false'**)
e. text constants (e.g. "Like This")

2. Expressions

Describe how all forms of expressions will be implemented.

a. describe how the values of constants (including text constants) will be accessed.
b. describe how the values of scalar variables will be accessed.
c. Describe how array elements will be accessed.
   Show details of array subscripting in the general case for one and two dimensional arrays.
d. describe how you will implement each of the arithmetic operators +, -, *, and /
e. describe how you will implement each of the comparison operators < , < =, =, **'not'** =, >=, >
f. describe how you will implement each of the boolean operators **'and'** , **'or'** , **'not'**
g. describe how you will implement conditional expressions

3. functions and procedures

Describe how you will implement

a. the activation record for functions and procedures.
b. procedure and function entrance code (activation record setup, allocation of local variables, etc)
c. procedure and function exit code (cleanup, return values, etc)
d. describe how you will implement parameter passing.
e. describe how you will implement function call and function value return.
f. describe how you will implement procedure call
g. Describe your display management strategy

4. Statements

Describe the general case of code you will generate for each of the statements in the language.

a. assignment statement
b. if statements
c. the while and repeat statements
d. all forms of exit statements
e. return statements
f. **'read'** and **'write'** statements
g. handling of minor scopes e.g. '{' declaration statement '}'

5. Everything Else

a. main program initialization and termination
b. any handling of scopes not described above
c. any other information that you think is relevant.

Notes

The design of a complete code generation scheme for mapping a language to a machine has a huge number of moving parts that all have to work well together to produce a correct result. **Consistency** and **Completeness** are hard to achieve in a design this size. There are a lot of degrees of freedom in designing this mapping, choices you make in one area will inevitably affect other areas. You need to clearly and completely describe how the individual parts of your design fit together to make a working whole.

A Suggested Strategy[1]

1. Each group member should independently review the entire document, ideally once or twice with a break in between reviews. This will help detect blatant contradictions / inconsistencies across the sections of the document. It's much better to discover these problems before submitting the document for marking.

2. A good Rule of Thumb would be *If everyone understands and agrees on all parts of the document then it is ready to submit*. If anyone doesn't understand any part of the document then either there is a communication problem, a mistake, or the person doesn't understand the machine model.

3. Be explicit about everything! Examples: order of operations/argument evaluation, What happens during function call (e.g. is it the caller or the callee of a function that is responsible for argument evaluation, activation record allocation/deallocation, etc.)

[1] Suggested by Peter Goodman, CSC488S TA, Winter 2013/204