Project P4: Code Generation

Due May 10th – no late submissions accepted!

Total 120 points.

- Generate assembly code for an input program.
- ***Program must compile and run on hellbender***

Grammar Semantics

- **Identifiers:** t2 tokens are identifiers
- **Numbers:** t3 tokens are integers
  - A leading uppercase letter indicates a positive integer
  - A leading lowercase letter indicates a negative integer
  - Zero can be represented as either case followed by one or more zeros
  - Note you will need to process these tokens to remove the leading character
- **Reserved:** t1 tokens
  - **!** negates the following argument
    - e.g. **! +980** means to multiply the value stored for +980 by -1
  - **"** means to allocate memory for the given identifier and initialize its value to zero
    - e.g. **" +33** means to allocate memory for identifier +33 and set its value to 0
  - **#** means to allocate memory for the identifier, read in a number, and assign the value to the identifier
    - e.g. **# +4** means to allocate memory for +4, request and read in a number, and store the value to +4
  - **$** means to print out the value of the identifier or immediate number to the screen
    - e.g. **$ Z45** would print the number 45 to the screen
  - **%** is the assignment operator
    - e.g. **+580 % a3** means to set +580 equal to -3
  - **&** is the same as standard plus sign, but it uses preorder
    - e.g. **& +24 R100** means to add the value for identifier +24 plus 100
  - **‘** means that if the first argument is strictly greater than the second argument, repeat the fourth argument *n* times, where *n* is the third argument
    - e.g. **‘ C24 b20 H4 $ +20** would cause the value set for +20 to be printed four times
    - If *n* is less than 1, do not perform the command
    - In other words, this is a conditional for loop
      - The condition is that the first argument must be greater than the second argument
        - If it is less than or equal to the second argument, nothing is done
      - The number of iterations for the for loop (if it is activated) is equal to the value of the third argument
  - **()** are delimiters
  - Note that arguments may be either an integer or an identifier for **! $ & ‘** and the second argument for **%**
    - If it is an identifier, use the value assigned to the identifier

Data

- All data is 2-byte signed integers
- Assume no overflow in operations

Target Language

- VM ACCumulator assembly language
- Executable is available on Canvas (virtMach) or you can use the online implementation: https://comp.umsl.edu/assembler/interpreter
- Description provided in VM_Architecture.pdf and VM_Language.pdf on Canvas

Requirements

- Invocation:
  > P4 [*file*]

- Call code generation function on the tree after calling static semantics function in main. Note that the container must be created in static semantics but is also accessed in code generation.

- The nodes of the parse tree are equivalent to the program in left to right traversal
  - Therefore, perform left to right traversal to generate the code
- When visiting code-generating nodes:
  - temporary variables may be needed - generate global pool variables (with distinguishable names)
  - if a value is produced, *always leave the result in the ACCumulator*
  - each node-kind generates the same code
    - regardless of parents
    - may be one of multiple cases
- At the end of the traversal, print STOP to target, followed by global variables *plus temporaries* in storage allocation

Suggestions

- In order to avoid naming temporary variables with an identifier that is given in an incoming program, use two or more alphabet characters for the base (e.g. temp1, temp2, etc.)
- While testing, your assembly programs can be run using either the executable, virtMach, or the online implementation at https://comp.umsl.edu/assembler/interpreter
  - Please upload 'virtMach' into the desired directory on hellbender and make it executable by typing 'chmod 700 virtMach'. You can then run programs by typing 'virtMach filename.asm'
  - If you use the online browser, you can upload your .asm files or copy and paste the text in the provided window. You may need to click on 'Reset Runtime' in the top menu bar between runs.

Test Files

- Given in powerpoint slides posted on Canvas. Be sure to generate nested statements for your tests.

Rubric

- 60 points for implementation and correctness, 60 points for comments.