Project P4: Code Generation
Due December 8, 2019 at 11:59 pm

Total 100 points.

- Generate assembly code for an input program written in our course grammar.
- The program is to parse the input, generate a parse tree, perform static semantics, and then generate a target file.
- Any error should display detailed message, including line number if available (depending on scanner).
- The target name should be
    - kb.asm if keyboard input
    - file.asm if file input. The base name matches that of the input file, that is if input basename is t2 then output should be t2.asm
- **Upon success, only display the name of the target file generated and no other output.**
- **Upon error and the error message, no other display should be generated and no target should be generated.**

Invocation

compile [file]

Where file is as before for previous projects. Wrong invocations will not be graded.

Grammar Semantics

- Program executes sequentially from beginning to end, one statement at a time
- Delimiters:
    - . , # [ ]
- Less than and greater than are standard symbols
    - < >
- '&' means absolute value for number
    - Number may be immediate or stored at location indicated by identifier
- The following operators have standard definitions, except no precedence
    - + - * /
    - Operations are applied from left to right
- 'let identifier : M' assigns the value of M to the given identifier
- '=' and '==' are both treated like standard '=='
    - Assignment of values to variables can only be performed using '[let] identifier : M'
- 'if [M Z M] T'
    - Means to do T if and only if (M Z M) is true (always true if Z is ':')
- 'repeat [M Z M] T'
    - Means to repeat T until (M Z M) is false
- When scanning an identifier, allocate memory using the identifier as the variable name and set value to zero
- When scanning a number, allocate memory using a temporary variable name and set value to the number

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)
- Description provided in VM_Architecture.pdf and VM_Language.pdf on Canvas

Suggestions

- Call code generation function on the tree after calling static semantics function in main. Note that the ST container must be created in static semantics but accessed in code generation
- The parse tree is equivalent to the program in left to right traversal (skipping syntactic tokens if not stored in the tree)
    - Therefore, perform left to right traversal to generate the code
- Some nodes are
    - code generating - most likely only those that have a token(s)
    - not code generating - most likely without tokens
        - if no children, return
        - if children, continue traversal then return
- When visiting code generating node
    - some actions can be preorder, some inorder, some postorder
    - temporary variables may be needed - generate global pool variables and allocate in storage allocation as global
    - if value is produced, always leave the result in the ACCumulator
    - each node-kind generates the same code
        - regardless of parents and children
        - may be one of multiple cases
        - the only difference may come from the token found in the node
- At the end of the traversal, print STOP to target (to be followed by global variables+temporaries in storage allocation)
    - In order to avoid naming temporary variables with an identifier that is given in an incoming program, use an upper-case letter for the first character of the temporary variable name (e.g. T1, T2, etc.)