1. Problem Statement

For our third assignment we were required to handle the symbol table and generate the assembly code.

2. How to use your program

The program uses Python as its base and requires one line of command for the program to begin, which is the path for the file containing the code. As an example for directing the path to the file, if your file name is test1.txt then you will simply put that name in the command line and the result will be displayed in the console.

3. Design of your program

The way this program works is that it uses functions called get_address, gen_inst, and backPatch. We also added lines into our parser to allow for the correct instructions to be added to the instruction table as well as making sure there are no redundant symbols in the symbol table. The get_address(identifier) function retrieves the memory address value of an id from the symbol table, or sets it to a new address with its type, id, and address (as well as incrementing the memory counter starting at 5000) if not already in the symbol table. We use this when generating the instructions so that we can retrieve an address of an identifier; either generating a new one or retrieving a current one, such as in a 'POPM' instruction which requires a memory location. The gen_inst(operation,operand) function creates an instruction with an operation and the operand, puts it in the instruction table, and increments the address counter. We use this function anytime an instruction is needed, in which the operation is an instruction from our instruction set, and the operand is either an address, value, or 'NIL'. The backPatch(jumpAddr) function gets an instruction from the instruction table using the address from the top of stack in jumpstack and sets the operand to the jumpAddress. Backpatching is used to fulfill unspecified information in which a jump is required such as in an "If" or "While" statement. We use three helper global variables to store values for use in generating instructions, such as storage for the qualifier, which will specify the type of the identifier when it is pushed into the symbol table, storage for the relational operator, which is used in generating comparison instructions, and storage for a get list, which stores the memory addresses to pop values into after a 'STDIN' instruction. Instructions are generated in relation to the rule associated with it, for example, if the rule called from the parser is the put rule, it will generate an 'STDOUT' instruction.

4. Any Limitation

None

5. Any shortcomings

Could not implement/Did not properly implement the <Compound> rule. Functionality may have been implemented in conditional statement.