ECS PROJECT #13 Guidelines
CSCI-410

As you have undoubtedly noticed, the Jack language has a number of deficiencies compared to most programming languages. Most of these are intentional as the authors strove to present a bare-bones language for which a recursive decent parser and compiler could be readily implemented in the time frame of just a couple weeks. Incorporating support to eliminate some of these deficiencies is straightforward, while others require significant effort. The purpose of this assignment, which is optional, is to address at least some of these deficiencies.

Task #1: Compiler support for ASCII character constants.

Many programming languages use the syntax `'X'` as an expression that evaluates to the character code for the character **X**. In most cases, this is the ASCII code for that character. Jack supports a limited set of characters that cannot be typed directly into the source code such as the newline character as well as arrow keys and other special keys. To represent these in the source code, we "escape" them by reserving one character, the backslash, as an escape character which tells the compiler that the next character should be interpreted differently.

`\n newline`	`\> right arrow`	`\[ page up`	`\e escape`
`\b backspace`	`\v down arrow`	`\] page down`	`\fnn Fnn key`
`\< left arrow`	`\h home`	`\+ insert`	`\" double quote`
`\^ up arrow`	`\: end`	`\- delete`	`\\ backslash`

Notice that all of the escape sequences, except for the function keys, consist of single character after the escape character. For the function keys, there are exactly three characters, starting with the character f and followed by the two digit number (padded with a leading zero if necessary) of the function key in question.

Extend the Jack compiler to recognize character constants any place where an integer constant could be used.

Task #2: Compiler support for escape sequences within string constants.

Extend the compiler's ability to construct string constants to include those strings that contain any of the escape sequences listed in Task #1.

ECS PROJECT #13 Guidelines
CSCI-410

Task #3: Compiler support for proper passing of string constants as function parameters.

While the present compiler will compile code in which a string constant is used as an argument to a function, it results in a memory leak because the pointer to the constant is lost. Your goal here is to correct that situation by either inserting code to properly dispose of the string when finished with it or to construct string constants within the heap space that are passed instead. This task is non-trivial, which is why the present specification ignores the issue.

Task #4: Compiler support for standard order of operations

Update the compiler so that it enforces the normal order of operations for arithmetic, namely that multiplication and division have precedence over addition and subtraction and that operations of the same precedence are evaluated left to right.

Task #5: Compiler support for all six relational operators

Update the compiler so that all six relational operators, namely {`=,<,>,!=,<=,>=`}, are supported.

Task #6: Compiler support for the distinction between bitwise and logical operations.

Update the compiler so that {`&,|,~`} perform bitwise operations on two integers while {`&&,||,!`} perform logic operations (i.e., produce a **`true`** or **`false`** result).

Task #7: Compiler support for bitwise and logical XOR.

Update the compiler to support the operators {`^,^^`} as the bitwise and logical XOR operators, respectively.

Task #8: Compiler support for variable initialization

Update the compiler so that non-object variables can be initialized when they are declared.

Task #9: OS support for scrolling text screen.

When the bottom of the screen is reached while outputting text, the present behavior of the OS library routines is unspecified and, in most implementations, either simply wraps to the top of the screen and starts overwriting text or remains at the bottom of the screen overwriting text on the bottom line. Your task here is to have the cursor remain at the bottom of the screen, but to scroll the entire contents of the screen up one line.

ECS PROJECT #13 Guidelines
CSCI-410

Task #10: OS support for base-2 logarithms.

Add two functions to the Math library as follows:

```
int log2(int x)
```

Computes the base-two logarithm of x, returning an int value y that is a fixed point value such that the actual value represented is equal to y/2^10.

```
int pow2(int x)
```

Takes a fixed point value x that represents an actual value of v=x/2^10 and returns the value 2^v.

Grading

Each task above is worth up to 10pts of extra credit.

Submission Procedures

For each task that you are attempting, place the code files for that task in a directory named Tnn where nn is the two-digit task number. All of these directories, in turn, should be in a directory named "13" (just like any other ECS project). Zip up the "13" directory like normal and submit like normal.