HACK Assembler: Overall Construction

Week 12

Recap: Lookup Tables

In charge of translating symbols and instructions into their respective binary formats

Lookup Table:

symbol → address

instruction → binary

Symbol Tables

symbol table will provide the equivalent address in ROM or data memory

```
symbolTable = HashTable(string, integer)
```

Instruction Tables

Three tables were defined for instructions: one for comp, dest, and jump

```
compTable = HashTable(string, integer)
destTable = HashTable(string, integer)
jumpTable = HashTable(string, integer)
```

Assembler Construction

Input: file name

Steps:

1. Create and populate the symbol table
2. Create and populate the instruction tables
3. Parse each line of the assembly, tokenize, and translate to binary

break line up into parts

@value

dest = comp; jump

Assembler Construction

<u>Steps for Parsing Lines:</u>

For each line:

- Skip if it is a comment or empty line
- If it is an A instruction, translate to binary or look up in symbol table
- If it is a C instruction, break it up into its parts and look up each one in the correct instruction table.

Main Implementation

```
asmFileName = argument[1]
symbolTable = createSymbolTable(asmFileName)
compTable = createCompTable()
destTable = createDestTable()
jumpTable = createJumpTable()
file = openFile(asmFileName)
if (failedToOpen(file)):
    printErrorMessage("failed to open file")
    exit
fileNameWithoutExtension = asmFileName.Strip(".asm")
binaryFileOut = openFile(fileNameWithoutExtension + ".hack")
```

Main Implementation

```
 for line in file:
     line = removeWhitespace(line)
     line = stripComments(line)
     if ( isEmptyLine(line) ):
         continue

     if ( isAInstruction(line) ):
         bin = aInstruction(line)
         binaryFileOut.Write(bin)
         continue

     if ( isCInstruction(line) ):
         bin = cInstruction(line)
         binaryFileOut.Write(bin)
         continue

binaryFileOut.close()
file.close()
```

A Instruction Functions

```
function isAInstruction(line):

    if (line[0] != '@') return false

   dropAt = line[1 : len(line) -1 ]

    if ( positiveInteger(dropAt) ):
         return true

    if ( isDigit(dropAt[0]) ):
         return false

    if ( validSymbolChars(dropAt) ):
         return true

    return false
```

```
function aInstruction(line):

    dropAt = line[1 : len(line) -1 ]

    if ( positiveInteger(dropAt) ):
         bin = toBinary15Bit(dropAt)
         return ( '0' + bin )

    if ( onlyAlphabetChars(dropAt) ):
         bin = symbolTable[dropAt]
         return ( '0' + bin)

    error()
```

because of the checks done before calling aInstruction, this line should not be reached

C Instruction Functions: Checking for C Instruction

```
function isCInstruction(line):

    if NOT ( ( countChars(line, '=') == 1 OR countChars(line, ';') == 1) ) :
        return false

   tokens = splitStringAtChars(line, ['=', ';'])

   if (tokens.size() != 2 AND tokens.size() != 3 ):
        return false

    // next step: check if instruction is in proper format
    if (tokens.size() == 2 ): // checking the two tokens
        if ( countChars(line, '=') == 1 ):
            // check for dest and comp
        else:
            // check for comp and jump
    else: // checking the three tokens
        // check for dest, comp, and jump
    return true
```

C Instruction Functions: Checking for C Instruction Cont

```
// checking the two tokens
if (tokens.size() == 2 ):
    if ( countChars(line, '=') == 1 ):
        // dest and comp
        if ( NOT destTable.contains(tokens[0]) ):
            return false
        if ( NOT compTable.contains(tokens[1]) ):
            return false
    else:
        // comp and jump
        if ( NOT compTable.contains(tokens[0]) ):
            return false
        if ( NOT jumpTable.contains(tokens[1]) ):
            return false
```

C Instruction Functions: Checking for C Instruction Cont

```
// checking the three tokens
else:
    // contains dest, comp, and jump
    if ( NOT destTable.contains(tokens[0]) ):
        return false

    if ( NOT compTable.contains(tokens[1]) ):
        return false

    if ( NOT jumpTable.contains(tokens[2]) ):
        return false

return true
```

C Instruction Functions: Translate C Instruction

```
function cInstruction(line):

    tokens = splitStringAtChars(line, ['=', ';'])
    prefix = "111"
    // next steps: translate tokens
    if (tokens.size() == 2): // translate two tokens
        if ( countChars(line, '=') == 1 ):
            // get dest and comp binary
            // error check translation
        else:
            // get comp and jump binary
            // error check translation
    else: // translate all three tokens
        // get binary for dest, comp, and jump
        // error check translation

    return prefix + comp + dest + jump
```

C Instruction Functions: Translate C Instruction Cont

```
// translate two tokens
if (tokens.size() == 2):
    if ( countChars(line, '=') == 1 ):
        // dest and comp
        dest = destTable[ tokens[0] ]
        comp = compTable[ tokens[1] ]
        jump = "000"
        if (dest == null || comp == null):
            error() // not value C instruction
    else:
        // comp and jump
        dest = "000"
        comp = compTable[ tokens[0] ]
        jump = jumpTable[ tokens[1] ]
        if (comp == null || jump == null ):
            error() // not valid C instruction
```

C Instruction Functions: Translate C Instruction Cont

```
// translate all three tokens
else:
    dest = destTable[ tokens[0] ]
    comp = compTable[ tokens[1] ]
    jump = jumpTable[ tokens[2] ]

    if (dest == null || comp == null || jump == null):
        error() // not valid C instruction

return prefix + comp + dest + jump
```

Combine everything together and the assembler is complete!