FULL ASSEMBLER

Reuse the Basic Assembler code and modify it to create Full Assembler which translates assembly programs (asm) *With Symbols* into binary code (hack)

Input Format

Input is provided through STDIN in either of the following forms. *Symbols* support should be taken into account.

- Independent assembly statements
- Complete assembly code with multiple statements which perform a certain function.

Remember to provide support for Symbols in addition to handling whitespaces (line comments, in-line comments, newline spaces , indentation etc)

NOTE: Input ends with new-line character. Eg:

```
@num // line
D=M  // line
   // new-line
```

Constraints

- Firstly, there are 4 Sample test cases (visible) so that you can *RUN* and debug your program. These are worth ***0 Points*** in total.
- Then, there are 3 Graded test cases (hidden) on which your program will be evaluated after clicking *SUBMIT*. These are worth ***35 Points*** in total.
- Finally, there are 3 Bonus test cases (hidden) to check for syntax errors (if any). These will also be evaluated after clicking *SUBMIT*. These are worth ***6 Points*** in total. We are looking to detect only two kinds of syntax errors (if any) as follows. As soon as your program detects either of the syntax error first, it must immediately exit the program with the corresponding String type error message on STDOUT.

ERROR: end of line expected but semicolon is found

ERROR: expression expected

Output Format

1. Output must be 16-bit binary code with reference to HACK assembly reference sheet.
2. For Bonus test cases, output is one of the two error messages of String datatype.
3. Output must be written onto STDOUT

Sample Input 0

```
@R1
```

Sample Output 0

FULL ASSEMBLER

```
0000000000000001
```

Explanation 0

- Input is an A-instruction, especially, for in-built symbol R1 which has value 1 on symbol table.
- Output should be of format *0value* where value is a 15-bit binary constant.
- Refer to HACK reference Sheet and In-Built Symbol Table for conversion.

Sample Input 1

```
@ARG
```

Sample Output 1

```
0000000000000010
```

Explanation 1

- Input is an A-instruction, especially, for in-built symbol ARG which has value 2 on symbol table.
- Output should be of format *0value* where value is a 15-bit binary constant.
- Refer to HACK reference Sheet and In-Built Symbol Table for conversion.

Sample Input 2

```
// Handling Variables

@num  // happens to be first variable
```

Sample Output 2

```
0000000000010000
```

Explanation 2

- Input is an A-instruction, especially, for variable num which should have value 16 on symbol table.
- Output should be of format *0value* where value is a 15-bit binary constant.
- Refer to HACK reference Sheet for conversion.

Sample Input 3

```
// Handle label declaration and usage
@6
D=A
(END) // label declaration
   @END  // label usage
```

```
0;JMP
```

Sample Output 3

```
0000000000000110
1110110000010000
0000000000000010
1110101010000111
```

Explanation 3

- Input is set of A-instructions {*@6*, *@END*} , and C-instructions {*D=A*, *0;JMP*}.
- Note *(END)* is not a part of asm instruction, simply, a declaration
- Output should be of format *0value* where value is a 15-bit binary constant or *111accccccdddjjj* based on instruction type.
- Refer to HACK reference Sheet for conversion.