RULES

- You should stop processing if a syntax error is detected in the input, print a syntax error message with the line number and the character offset in the input file where observed. A syntax error is defined as a missing token (e.g. 4 used symbols are defined but only 3 are given) or an unexpected token. Stop processing and exit.
- If a symbol is defined multiple times, print an error message and use the first definition. The error message is to appear as part of printing the symbol table (following symbol=value printout on the same line).
- If a symbol is used in an E-instruction but not defined anywhere, print an error message and use the value absolute zero.
- If a symbol is defined but not used, print a warning message and continue.
- If an address appearing in a first definition of a symbol exceeds the size of the module, print a warning message in pass1 after processing the module and treat the address given as 0 (relative to the module). If the symbol is a redefinition then print a warning message (see message below).
- If an external operand is too large to reference an entry in the use list, print an error message and treat the operand as relative=0.
- If a symbol appears in a use list but is not actually used in the module (i.e., not referred to in any E-type address), print a warning message and continue. If the same unused symbol appears multiple times in uselist, multiple warnings are to be printed.
- If an absolute address exceeds the size of the machine, print an error message and use the absolute value zero.
- If a relative address exceeds the size of the module, print an error message and use the module relative value zero (that means you still need to remap "0" that to the correct absolute address).
- If an illegal immediate operand (I) is encountered (i.e. >= 900), print an error and convert the operand value to 999.
- If an illegal opcode is encountered (i.e. op >= 10), print an error and convert the <opcode,operand> to 9999.
- If a module operand is invalid, print an error message and assume module 0.
- Accepted symbols should be upto 16 characters long (not including terminations e.g. '\0'), any longer symbol names are erroneous.
- a uselist or deflist should support 16 definitions, but not more and an error should be raised.
- number of instructions are unlimited (hence the two pass system), but in reality they are limited to the machine size.
- Symbol table should support at least 256 symbols (reference program supports exactly 256 symbols).
- Module table should support at least 128 entries (reference program supports exactly 128 entries).

(M) operand is the number of a valid module and is replaced with the base address of that module.

(A) operand is an absolute address which will never be changed in pass2; however it can't be ">=".

(R) operand is a relative address in the module which is relocated by replacing the relative address with the absolute address of that relative address after the module's global address has been determined (absolute_addr = module_base+relative_addr).

(I) an immediate operand is unchanged, but must be less than 900.

(E) operand is an external address which is represented as an index into the uselist. For example, a reference in the program text with operand K represents the Kth symbol in the use list, using 0-based counting, e.g., if the use list is "2 f g", then an instruction "E 7000" refers to f, and an instruction "E 5001" refers to g. You must identify to which global address the symbol is assigned and then replace the operand with that global address.

INPUT

```
1
   xy 2
2 z          xy
5  R 1004  I   5432
     E 7001  R
8002  E 2000  0      1 z
6 R 8001  E 1000  E
1000  E 3000  R 1002  A 1010
0
1
z
2
R
5001
E 4000
1 z 2
2
xy z
3 A 8000
E 1001  E 2000
```

Tokenizer

TOKENIZED INPUT

Token: 1:1 : 1 Token: 2:5 : xy Token: 2:8 : 2	Def List
Token: 3:1 : 2 Token: 3:3 : z Token: 3:16 : xy	Use List
Token: 4:1 : 5 Token: 4:4 : R Token: 4:6 : 1004 Token: 4:12 : I Token: 4:16 : 5432 Token: 5:7 : E Token: 5:9 : 7001 Token: 5:15 : R Token: 6:1 : 8002 Token: 6:7 : E Token: 6:9 : 2000	Program Text
Token: 6:10 : 0	Def List
Token: 6:22 : 1 Token: 6:24 : z	Use List
Token: 7:1 : 6 Token: 7:3 : R Token: 7:5 : 8001 Token: 7:11 : E Token: 7:13 : 1000 Token: 7:19 : E Token: 8:1 : 1000 Token: 8:7 : E Token: 8:9 : 3000 Token: 8:15 : R Token: 8:17 : 1002 Token: 8:23 : A Token: 8:25 : 1010	Program Text
Token: 9:1 : 0	Def List
Token: 10:1 : 1 Token: 12:1 : z	Use List
Token: 13:1 : 2 Token: 14:1 : R Token: 15:1 : 5001 Token: 16:1 : E Token: 16:3 : 4000	Program Text
Token: 17:1 : 1 Token: 17:3 : z Token: 17:5 : 2	Def List
Token: 18:1 : 2 Token: 19:1 : xy Token: 19:4 : z	Use List
Token: 20:1 : 3 Token: 20:3 : A Token: 20:5 : 8000 Token: 21:1 : E Token: 21:3 : 1001 Token: 21:9 : E Token: 21:11 : 2000	Program Text
Final Spot in File : line=21 offset=15	

Module 1

Module 4

Parser 1

The first pass does syntax and early error checking and creates the symbol table.

OUTPUT - 1

```
Symbol Table
xy = 2
z = 0
Memory Map
000: 1004
001: 5432
002: 7001
003: 8002
004: 2000
005: 8001
006: 1000
007: 1000
008: 3000
009: 1002
010: 1010
011: 5001
012: 4000
013: 8000
014: 1001
015: 2000
```

Parser 2

The second pass performs additional error checking and instruction transformation.

OUTPUT - 2

```
Symbol Table
xy=2
z=15
Memory Map
+0
0:      R 1004         1004+0 =  1004
1:      I 5678                   5678
2: xy:  E 2000 ->z               2015
3:      R 8002         8002+0 =  8002
4:      E 7001 ->xy              7002
+5
0:      R 8001         8001+5 =  8006
1:      E 1000 ->z               1015
2:      E 1000 ->z               1015
3:      E 3000 ->z               3015
4:      R 1002         1002+5 =  1007
5:      A 1010                   1010
+11
0:      R 5001         5001+11=  5012
1:      E 4000 ->z               4015
+13
0:      A 8000                   8000
1:      E 1001 ->z               1015
2 z:    E 2000 ->xy              2002
```

Organized Input

Module 1	Def List	1 xy 2
	Use List	2 z xy
	Program Text	5 R 1004 I 5432 E 7001 R 8002 E 2000
Module 2	Def List	0
	Use List	1 z
	Program Text	6 R 8001 E 1000 E 1000 E 3000 R 1002 A 1010
Module 3	Def List	0
	Use List	1 z
	Program Text	2 R 5001 E 4000
Module 4	Def List	1 z 2
	Use List	2 xy z
	Program Text	3 A 8000 E 1001 E 2000