You are to implement a two-pass linker in C, C++, or Java and submit the **source** code, which we will compile and run.

The target machine is word addressable and has a memory of 600 words, each consisting of 4 decimal digits. The first (leftmost) digit is the opcode, which is unchanged by the linker. The remaining three digits (called the address field) are either (1) an immediate operand, which is unchanged; (2) an absolute address, which is unchanged; (3) a relative address, which is relocated; or (4) an external address, which is resolved. Relocating relative addresses and resolving external references were discussed in class and are in the notes.

Input consists of a series of object modules, each of which contains three parts: definition list, use list, and program text.

The linker processes the input twice (that is why it is called two-pass). Pass one determines the base address for each module and the absolute address for each external symbol, storing the later in the symbol table it produces. The first module has base address zero; the base address for module I+1 is equal to the base address of module I plus the length of module I. The absolute address for symbol S defined in module M is the base address of M plus the relative address of S within M. Pass two uses the base addresses and the symbol table computed in pass one to generate the actual output by relocating relative addresses and resolving external references.

The definition list is a count ND followed by ND pairs (S, R) where S is the symbol being defined and R is the relative address to which the symbol refers. Pass one relocates R forming the absolute address A and stores the pair (S, A) in the symbol table.

The use list is a count NU followed by NU pairs (S, R), where S is the symbol being used and R is a relative address where S is used. The (dummy) address initially in R is a pointer to the next use of S. The linked list of uses ends with a pointer of 777.

The program text consists of a count NT followed by NT pairs (type, word), where word is a 4-digit instruction described above and type is a single character indicating if the address in the word is **I**mmediate, **A**bsolute, **R**elative, or **E**xternal. NT is thus the length of the module.

Other requirements: error detection, arbitrary limits, and space used.

To received full credit, you must check the input for various errors. All error messages produced must be informative, e.g., ‘‘Error: The symbol ‘diagonal’ was used but not defined. It has been given the value 0’’. You should continue processing after encountering an error and should be able to detect multiple errors in the same run.

- If a symbol is multiply defined, print an error message and use the value given in the first definition.
- If a symbol is used but not defined, print an error message and use the value zero.
- If a symbol is defined but not used, print a warning message and continue.
- If an address appearing in a definition exceeds the size of the module, print an error message and treat the address given as 0 (relative).
- If an address appearing in a use list exceeds the size of the module, print an error message and treat the address as the sentinel ending the list.
- If an address on a use list is not type **E**, print an error message and treat the address as type **E**.
- If a type **E** address is not on a use list, print an error message and treat the address as type **I**.

You may need to set “arbitrary limits”, for example you may wish to limit the number of characters in a symbol to (say) 8. Any such limits should be clearly documented in the program and if the input fails to meet your limits, your program must print an error message and continue if possible. Naturally, the limits must be large enough for all the inputs on the web. Under no circumstances should your program reference an array out of bounds, etc.

During each pass, your linker must processes one module at a time. It must use space proportional to the size of one module not to the total size of the input program. The one exception is the symbol table, which in theory could as big as the program, but in practice is much smaller. In particular, you **CANNOT** read all the input for all the modules at once. Instead, you read one module, then process it, then read the next module, etc. This requirement does NOT make the program harder, just better.

There are several sample input sets on the web. The first is the one below and the second is an re-formatted version of the first. Some of the input sets contain errors that you are to detect as described above. We will run your lab on these (and other) input sets. Please submit the SOURCE code for your lab, together with a README file (required) describing how to compile and run it. Your program must either read an input set from standard input, or accept a command line argument giving the name of the input file; README specifies which option you chose. You may develop your lab on any machine you wish, but must insure that it compiles and runs on the NYU system assigned to the course. The expected output is also on the web. Let me know right away if you find any errors in the output.

```
1 xy 2
1 z 4
5 R 1004  I 5678  E 2777  R 8002  E 7002
0
1 z 3
6 R 8001  E 1777  E 1001  E 3002  R 1002  A 1010
0
1 z 1
2 R 5001  E 4777
1 z 2
1 xy 2
3 A 8000  E 1777  E 2001
```

The following is output annotated for clarity and class discussion. Your output is not expected to be this fancy.

```
Symbol Table
    xy=2
    z=15

            Memory Map
 +0
 0:       R 1004      1004+0 = 1004
 1:       I 5678               5678
 2: xy:   E 2777 ->z           2015
 3:       R 8002      8002+0 = 8002
 4: ->z   E 7002               7015
 +5
 0        R 8001      8001+5 = 8006
 1        E 1777 ->z           1015
 2        E 1001 ->z           1015
 3 ->z    E 3002               3015
 4        R 1002      1002+5 = 1007
 5        A 1010               1010
 +11
 0        R 5001      5001+11= 5012
 1 ->z    E 4777               4015
 +13
 0        A 8000               8000
 1        E 1777 ->xy          1002
 2 z:->xy E 2001               2002
```