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

The target machine is word addressable and has a memory of 200 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

- An immediate operand, which your program does not change.
- An absolute address, which your program does not change.
- A relative address, which your program relocates.
- An external address, which your program resolves.

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 lists: definitions of symbols that may be used in other modules, uses in this module of symbols defined in other modules, and the program text. The format of the lists is given below and a complete sample input is on the next page and in the class notes

The linker processes the input twice (that is why it is called two-pass). Pass one determines the base address for each module and produces a symbol table containing the absolute address for each defined symbol.

We assume the first module has base address zero; the base address of module M+1 is equal to the base address of module M plus the length of module M. 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 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.

The use list is a count NU followed by the NU external symbols used in the module. An E reference in the program text with address K represents the Kth symbol in the use list, using 0-based counting. For example, if the use list is ‘‘2 f g’’, then an instruction ‘‘E 7000’’ refers to f, and an instruction ‘‘E 5001’’ refers to g.

Required Error Checking

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 continue processing after encountering an error and must be able to detect multiple errors in the same run.

- If a symbol is defined but not used, print a warning message.
- 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 an address appearing in a definition exceeds the size of the module, print an error message and treat the address as 0 (relative).
- If an external address is too large to reference an entry in the use list, print an error message and treat the address as immediate.
- If a symbol appears in a use list but it not actually used in the module (i.e., not referred to in an E-type address), print a warning message.
- If an absolute address exceeds the size of the machine, print an error message and use the value zero.
- If a relative address exceeds the size of the module, print an error message and use the value zero (absolute).

There are several sample input sets on the web, together with the expected output for each. Let me know right away (via the class mailing list) if you find any errors in the output. The first input and output are below and the second input is simply a re-formatted version of the first (so it has the same output). 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 via *classes.nyu.edu* the SOURCE code for your lab, together with a README file (required) describing how to compile and run your program. Your program must either read from standard input, i.e., the keyboard, or accept a command line argument giving the name of the input file. Note that your program may **not** query the user for the name of the input file. 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.

```
4
1 xy 2
2 z xy
5 R 1004  I 5678  E 2000  R 8002  E 7001
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
```

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 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
```