

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

The target machine is word addressable and has a memory of 300 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.
- (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 the NU “pairs”. The first entry in the pair is an external symbol used in the module. The second entry is a list of relative addresses in the module in which the symbol is used. The list is terminated by a sentinel of -1. For example, if the use list is “2 f 3 1 4 -1 xyg 0 -1”, then the symbol f is used in instructions 1, 3, and 4, and the symbol xyg is used in instruction 0.

The program text consists of a count NT followed by NT 5-digit numbers. NT is the length of the module. The left four digits of each number form the instruction as described above. The last (rightmost) digit specifies the address type: 1 signifies “immediate”, 2 “absolute”, 3 “relative”, and 4 “external”.

Other requirements: error detection and arbitrary limits.

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 111”. 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 second definition.
- If a symbol is used but not defined, print an error message and use the value 111.
- If a symbol is defined but not used, print a warning message and continue.
- If multiple symbols are listed as used in the same instruction, print an error message and ignore all but the last usage given.
- If an address appearing in a definition exceeds the size of the module, print an error message and treat the address given as the last word in the module.
- If an address appearing in a use list exceeds the size of the module, print an error message and ignore this particular use.
- If an absolute address exceeds the size of the machine, print an error message and use the largest legal value.
- If a relative address exceeds the size of the module, print an error message and use the last address of the module

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.

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, i.e., the keyboard. 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
2 z 2 -1 xy 4 -1
5 10043 56781 20004 80023 70014
0
1 z 1 2 3 -1
6 80013 10004 10004 30004 10023 10102
0
1 z 1 -1
2 50013 40004
1 z 2
2 xy 2 -1 z 1 -1
3 80002 10014 20004

```

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:      10042      1004+0 = 1004
1:      56781      5678
2: xy:   20004 ->z      2015
3:      80023      8002+0 = 8002
4:      70014 ->xy      7002
+5
0      80013      8001+5 = 8006
1      10004 ->z      1015
2      10004 ->z      1015
3      30004 ->z      3015
4      10023      1002+5 = 1007
5      10102      1010
+11
0      50013      5001+11= 5012
1      40004 ->z      4015
+13
0      80002      8000
1      10014 ->z      1015
2 z:   20004 ->xy      2002

```