Implementing a two-pass linker. In general, a linker takes individually compiled code/object modules and creates a single executable by resolving external symbol references (e.g. variables and functions) and module relative addressing by assigning global addresses after placing the modules' object code at global addresses. Rather than dealing with complex x86 tool chains, we assume a target machine with the following properties: (a) word addressable, (b) addressable memory of 512 words, and (c) each valid word is represented by an integer (< 10, see errorcodes below. The operand is modified/retained based on the instruction type in the program text as follows: (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). (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. (I) an immediate operand is unchanged. (A) operand is an absolute address which will never be changed in pass2; however it can't be ">=" the machine size (512); The linker must process the input twice (that is why it is called two-pass) (to preempt the favored question: "Can I do it in one pass?" → NO, because storing tokens makes your program more complex). Pass One parses the input and verifies the correct syntax and determines the base address for each module and the absolute address for each defined symbol, storing the latter in a symbol table. The first module has base address zero; the base address for module X+1 is equal to the base address of module X plus the length of module X (defined as the number of instructions in a module). The absolute address for symbol S defined in module M is the base address of M plus the relative address of S within M. After pass one print the symbol table (including errors related to it (see rule2 later)). Do not store parsed tokens, the only data you should and need to store between passes is the symboltable. Pass Two again parses the input and uses the base addresses and the symbol table entries created in pass one to generate the actual output by relocating relative addresses and resolving external references. You should reuse pass-1 parser code just with different actions. You must clearly mark your two passes in the code through comments and/or proper function naming. Other requirements: error detection, limits, and space used.