1024 A Linking Loader

An *object module* is produced by a compiler as a result of processing a source program. A *linking loader* (or just a *linker*) is used to combine the multiple object modules used when a program contains several separately compiled modules. Two of its primary tasks are to relocate the code and data in each object module (since the compiler does not know where in memory a module will be placed), and to resolve symbolic references from one module to another. For example, a main program may reference a square root function called `sqrt`, and that function may be defined in a separate source module. The linker will then minimally have to assign addresses to the code and data in each module, and put the address of the `sqrt` function in the appropriate location(s) in the main module's code.

An object module contains (in order) zero or more *external symbol definitions*, zero or more *external symbol references*, zero or more bytes of code and data (that may include references to the values of external symbols), and an end of module marker. In this problem, an object module is represented as a sequence of text lines, each beginning with a single uppercase character that characterizes the remainder of the line. The format of each of these lines is as follows. Whitespace (one or more blanks and/or tab characters) will appear between the fields in these lines. Additional whitespace may follow the last field in each line.

- A line of the form '`D` *symbol offset*' is an external symbol definition. It defines *symbol* as having the address *offset* bytes greater than the address where the first byte of code and data for the current object module is located by the linker. A *symbol* is a string of no more than eight upper case alphabetic characters. The *offset* is a hexadecimal number with no more than four digits (using only upper case alphabetic characters for the digits A through F). For example, in a module that is loaded starting at the address 100_{16}, the line '`D START 5C`' indicates that the symbol `START` is defined as being associated with the address $15C_{16}$. The number of "D" lines in a test case is at most 100.

- A line of the form '`E` *symbol*' is an external symbol reference, and indicates that the value of *symbol* (presumably defined in another object module) may be referenced as part of the code and data for the current module. For example, the line '`E START`' indicates that the value of the symbol `START` (that is, the address defined for it) may be used as part of the code and data for the module. Each of the "E" lines for each module is numbered sequentially, starting with 0, so they can be referenced in the "C" lines.

- A line of the form '`C` n $byte_1$ $byte_2$ $\ldots byte_n$' specifies the first or next n bytes of code and data for the current module. The value n is specified as a one or two digit hexadecimal number, and will be no larger than 10 hexadecimal. Each *byte* is either a one or two digit hexadecimal number, or a dollar sign. The first byte following a dollar sign (always on the same line) gives the 0-origin index of an external symbol reference for this module, and identifies the symbol which is to have its 16-bit value inserted at the current point in the linked program (that is, in the location indicated by the dollar sign and the following byte). The high-order byte is placed in the location indicated by the dollar sign. The values specified for the other bytes (those not following a dollar sign) are loaded into sequential memory locations, starting with the first (lowest) unused memory location. For example, the line '`C 4 25 $ 0 37`' would cause the values 25_{16} 01_{16} $5C_{16}$ and 37_{16} to be placed in the next four unused memory locations, assuming the first "E" line for the current module specified a symbol defined as having the address $15C_{16}$. If the 0-origin index of the external symbol reference is an undefined symbol, the 16-bit value inserted at the current point in the linked program is 0000_{16}.

- A line of the form 'Z' marks the end of an object module.

You may assume that no address requires more than four hexadecimal digits. Lines are always given in the order shown above. There are no syntax errors in the input.

Input

This problem has multiple input cases. The input for each case is one or more object modules, in sequence, that are to be linked, followed by a line beginning with a dollar sign. The first address at which code is to be loaded in each case is 100_{16}.

The last case will be followed by a line containing only a dollar sign.

Output

For each case, print the case number (starting with 1), the 16-bit checksum of the loaded bytes (as described below), and the load map showing the address of each externally defined or referenced symbol, in ascending order of symbol name. For undefined symbols, print the value as four question marks, but use zero as the symbol's value when it is referenced in "C" lines. If a symbol is defined more than once, print 'M' following the address shown in the load map, and use the value from the first definition encountered in any object module to satisfy external references. Format the output exactly as shown in the samples.

The 16-bit checksum is computed by first setting it to zero. Then, for each byte assigned to a memory location by the loader, in increasing address order, circularly left shift the checksum by one bit, and add the byte from the memory location, discarding any carry out of the low-order 16 bits.

Print a blank line between datasets.

Sample Input

```
D MAIN 0
D END 5
C 03 01 02 03
C 03 04 05 06
Z
$
D ENTRY 4
E SUBX
E SUBY
C 10 1 2 3 4 5 $ 0 6 7 8 9 A B C D E
C 8 10 20 30 40 50 60 70 80
C 8 90 A0 B0 C0 D0 E0 $ 1
C 5 $ 0 FF EE DD
Z
D SUBX 01
C 06 A B C D E F
Z
D SUBX 05
C 06 51 52 53 54 55 56
Z
$
$
```

Sample Output

```
Case 1: checksum = 0078
 SYMBOL   ADDR
--------  ----
END       0105
MAIN      0100

Case 2: checksum = 548C
 SYMBOL   ADDR
--------  ----
ENTRY     0104
SUBX      0126 M
SUBY      ????
```