Executable Format

Time limit: 1.0s **Memory limit:** 64M

The Windows/MS-DOS world was never really designed. Rather, it evolved. The whole world is really a mess. For example, there were many executable formats back then.

The first executable format is `COM`, used by MS-DOS as it was compatible with CP/M, with extension `.com`. This format has absolutely no format. The entire file is loaded at offset `0x100` in the current segment and simply executed by jumping to the first byte with a `jmp 100h`.

The next format is the `MZ` format, used by MS-DOS, with extension `.exe`, but `.com` works too. This format has a header that stores information about the code, and most importantly, it allows the code to be relocated to any place in memory, instead of being forced at `0x100`. This format is identified by the magic number `MZ`, word `0x5A4D`, or bytes `4D 5A`. (x86 is little-endian.) MZ, of course, is the initials of the legendary Mark Zbikowski, an MS-DOS developer.

And then there is `NE` (new executable), the executable format of Windows 1.0 through 3.x, with the extension `.exe` or `.com`. This format is designed to be compatible with `MZ`, so it starts with an `MZ` header and the letters `MZ`. So how would one identify it? Luckily, the MZ header has a field `e_lfanew`, at the offset `0x3C`. This is a `DWORD`, or unsigned 32-bit integer (4 bytes) that points to the start of a new header, in this case `NE`. Since the x86 is a little-endian platform, the integer is also little-endian, meaning that the least significant bytes are listed first. You can already see this from the previous example of `0x5A4D`, which is stored as `4D 5A` as bytes. The NE header starts with the bytes `NE`. **That is, the first 2 bytes starting at the offset represented by `e_lfanew` form the header name.**

Similarly, OS/2 used the `LE` format (linear executable). It also made use of `e_lfanew`, but instead of `NE`, `LE` is found at that offset.

Finally, the most common executable format of today, but not even publicly released in 1992, is the `PE` format (portable executable). It is the executable format of modern Windows. Similar to both `LE` and `NE`, it used `e_lfanew`, but the bytes at that offset is `PE` (`0x4550`, `50 45`).

Your assignment is to determine the format of an executable from its hexdump.

Sidebar

How do you know if something that starts with `MZ` is not just a `COM` file which happened to start with `MZ`? We know for sure because this is what `MZ`, or `4D 5A` disassembles to:

```
0100    4D    DEC    BP
0101    5A    POP    DX
```

`DEC BP` decrements a register whose value is undefined. No one in the right mind would do that. But wait, it gets worse. `POP DX` underflows the stack, because there is nothing to pop off the stack. This is a very serious bug, because

the stack pointer overflows (stack grows downward) and would wrap around to `0`.

Input Specification

The first line contains the integer N such that $1 \leq N \leq 131\,072$, the number of bytes in the hex dump.

The next $\lceil \frac{N}{16} \rceil$ lines contain the hex dump, each containing 16 (possibly less for the last line) bytes of the code in hexadecimal, separated by spaces.

Output Specification

Output `COM`, `MZ`, `NE`, `LE`, or `PE`, depending on the format detected. If `e_lfanew` points outside the file or to an unknown value, output `MZ`.

Sample Input

```
256
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00
9D 68 BA 89 D9 09 D4 DA D9 09 D4 DA D9 09 D4 DA
D0 71 41 DA D8 09 D4 DA D0 71 50 DA DB 09 D4 DA
D0 71 47 DA DE 09 D4 DA D9 09 D5 DA F1 09 D4 DA
D0 71 57 DA CF 09 D4 DA D0 71 40 DA D8 09 D4 DA
D0 71 45 DA D8 09 D4 DA 52 69 63 68 D9 09 D4 DA
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 50 45 00 00 4C 01 04 00
1C AC 88 54 00 00 00 00 00 00 00 00 E0 00 03 01
```

Sample Output

```
PE
```

Explanation

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ.........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 E8 00 00 00  ............è...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  9D 68 BA 89 D9 09 D4 DA D9 09 D4 DA D9 09 D4 DA  .hº‰Ù.ÔÚÙ.ÔÚÙ.ÔÚ
00000090  D0 71 41 DA D8 09 D4 DA D0 71 50 DA DB 09 D4 DA  ÐqAÚØ.ÔÚÐqPÚÛ.ÔÚ
000000A0  D0 71 47 DA DE 09 D4 DA D9 09 D5 DA F1 09 D4 DA  ÐqGÚÞ.ÔÚÙ.ÕÚñ.ÔÚ
000000B0  D0 71 57 DA CF 09 D4 DA D0 71 40 DA D8 09 D4 DA  ÐqWÚÏ.ÔÚÐq@ÚØ.ÔÚ
000000C0  D0 71 45 DA D8 09 D4 DA 52 69 63 68 D9 09 D4 DA  ÐqEÚØ.ÔÚRichÙ.ÔÚ
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0  00 00 00 00 00 00 00 00 50 45 00 00 4C 01 04 00  ........PE..L...
000000F0  1C AC 88 54 00 00 00 00 00 00 00 00 E0 00 03 01  .¬ˆT........à...
```

Credits to the Python Software Foundation because sample input is `python.exe`'s first 256 bytes, and also to Microsoft whose compiler generated that file.