Two Pass Linker

Problem Description

When the compiler has finished processing a module, they produce **an object module** that is almost runnable. The following picture shows four object modules.

```
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
```

Each object module contains **three parts**, a definition list, a use list, and the program text itself.

- **definition list** consists of a integer ***defcount*** followed by defcount pairs (S, R) where S is the string representing the symbol being defined and R is the offset. **The real value of a symbol is offset plus the number of instructions before the definition of this symbol.**
  - For example, `z` is definded in the module 4, and the number of insturctions before module 4 is $13(=5+6+2)$. So `z = 13 + 2 = 15`. `xy` is definded in the module 1, and the number of insturctions before module 1 is 0. So `xy = 0 + 2 = 2`. And the real value of the symbols are global and unchanged.
- **use list** consists of a integer ***usecount*** followed by usecount symbols that are referred to in this module.
- **program text** consists of a count ***codecount*** followed by codecount pairs (type, instr), where instr is a 4-digit instruction (integer) and type is a single character indicating **I**mmediate, **A**bsolute, **R**elative, or **E**xternal.

Our task is to change the modules into memory maps.

```
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
```

An instruction (4 decimals digits) is composed of an **opcode (leftmost digit)** and **an operand (rightmost 3 digits)**. The opcode always remains unchanged by the linker.

The operand is modified/retained based on the instruction type in the program text as follows:

1. (I) an immediate operand is unchanged;
2. (A) an absolute address is unchanged;
3. (R) a relative address is relocated by adding the number of the instructions before this module.
4. (E) an external address is 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. If the use list is `2 f g`, then an instruction `E 7000` refers to `f`, and an instruction `E 5001` refers to `g`. `E 3002` can't refer anything which will be regarded as an error. **Before you relocate the address, you must know the real value of the symbols.**

Moreover, we need to find out the errors and the warnings. For the sake of simplicity, we only define some kinds of errors and warnings.

1. Error 1: We defined some symbols twice.
    - If `xy` appears in definition list twice.
    - For Error 1, we ignore the second definition, and output `Error 2: xy` before memory map. `xy` is the symbol name.
2. Error 2: We used some symbols not being definded.
    - If `xy` never appears in definition list.
    - If the module's usecount is 2, but the program text has `E 5002`.
    - For Error 2, we just ignore the instructions and the memory map, and output `Error 2`.
3. Warning 1: The instruction is larger than 9999.
    - If larger, we set its memory map to 9999.
    - Output `Warning 1` after memory map.
4. Warning 2: The symbols in the use list are never used in program text.
    - Output `Warning 2: xy` after the module. `xy` represents the name of symbol.
5. Warning 3: The symbols that are defined are never used in program text.

- Output `Warning 3: xy` after the program. `xy` represents the name of symbol.

Intput

The first line is the number of programs. Each program consists of several modules.

For every program, the first line is the number of modules.

$$\sum defcount \leq 16$$

$$usecount \leq 16$$

$$\sum codecount \leq 512$$

Output

The number of memory map with the errors and warnings.

Intput sample

```
2
1
2 z 2 xy 1
2 z xy
5 R 10002 I 10002 I 5000 I 0 R 3001
4
2 z 2 xy 2
1 z
5 R 0001  I 5678  E 2000  R 8002  E 7000
0
1 z
6 R 8001  E 1000  E 1000  E 3000  R 1002  A 1010
0
1 z
2 R 5001  E 4000
2 z 4 xy 4
1 z
3 A 8000  E 1000  E 2000
```

Output example

```
9999
Warning 1
9999
Warning 1
5000
0000
3001
Warning 2: xy
Warning 2: z
Warning 3: xy
Warning 3: z
Error 1: xy
Error 1: z
0001
5678
2002
8002
7002
8001
1002
1002
3002
1002
1010
5001
4002
```

```
8000
1002
2002
Warning 3: xy
```