PRNG Sequence Guessing

We often use *Random number generators* in our life. Java programming language has class `java.util.Random` for it. 64-bit parameter `seed` is maintained during the lifetime of an instance of this class. On creation it is initialized with random 48-bit value. Method `nextInt(n)` returns integer in range $[0, n)$. On every call of `nextInt(n)` seed is changed and then number consisting of bits from 17 to 47 of seed, inclusive, is returned:

```
int nextInt(int n) {
 seed = (seed * 0x5DEECE66D + 0xB) & ((1L << 48) - 1);
 return (int) ((seed >>> 17) % n);
}
```

Note that in Java on overflow computations are done modulo 2^{64}.

Operator `>>>` *in Java is zero fill right shift, it fills leading bits with zeros, even if number is negative. The behavior would be the same, if the variable were unsigned. (It exists, because Java doesn't have unsigned integer types)*

Given ten output values of `nextInt(1000)`, guess the next values to be output by the generator.

Input Format

The first line of input contains integer N indicating the number of test cases to follow.

Each of the next N lines contains ten integers $a_1, a_2, \ldots a_{10}$: sequentially generated numbers by the same Random instance's `nextInt(1000)` function.

Constraints

- $1 \leq N \leq 10$
- $0 \leq a_i < 1000$

Output Format

For each test case, output a line containing ten integers $a_{11}, a_{12}, \ldots, a_{20}$: the next ten numbers returned by `nextInt(1000)`.

Sample Input 0

```
2
643 953 522 277 464 366 321 409 227 702
877 654 2 715 229 255 712 267 19 832
```

Sample Output 0

```
877 633 491 596 839 875 923 461 27 826
101 966 573 339 784 718 949 934 62 368
```