Rand

Time Limit	Memory Limit
1 second	256 MB

Flavour Text

You find a strange website. All that's on it is a single text box, asking for a number.

```
Number? ____________ [submit]
>
```

Intrigued, you type in 1729, a rather dull number, and hit submit. After a moment, a message appears underneath.

```
Number? _______1729_ [submit]
> WA. Correct answer was 1804289383.
```

You try that number.

```
Number? _1804289383_ [submit]
> WA. Correct answer was 846930886.
```

You keep trying numbers, yet each time you get it wrong, and a different number appears at the bottom. Giving up, you go to bed. That night, you have a very strange dream, but all you remember in the morning is the number 1681692777.

```
Number? _1681692777_ [submit]
> AC
```

In your excitement at the sight of AC, you accidentally refresh the page.

```
Number? ____________ [submit]
>
```

Devastated, you decide to dedicate the rest of your life to guessing the correct random number. Of course, the odds aren't that bad, right?

Problem Statement

Your goal is to guess the number that will be next output by a pseudo-random number generator (PRNG).

Subtasks, Constraints and Scoring

The code for all of these PRNGs will be included at the end of this document, and will be identical to their implementations in both the provided grader and the sample grader.

Subtask	Points	PRNG name	Maximum calls to `guess`
1	7	`stdrand`	2
2	11	`hashed`	2
3	14	`xor16`	17
4	25	`lcg`	4
5	18	`xorshift`	6
6	25	`twister`	630

Your score for each subtask will be the minimum score across all testcases for that subtask in a particular run.

Interface

You must write and submit one file named `solution.cpp`. This file should begin with the line `#include "grader.h"`. Further, it should implement the function `solution`, which the grader will call.

```
void solution(int subtask);
```

The parameter `subtask` is the subtask number. This function should call `guess`.

```
uint64_t guess(uint64_t x);
```

The parameter `x` is your guess for the next number. If you guess correctly, the program exits. Otherwise, it returns the actual next number.

If the number of calls to `guess` exceeds the limit for that subtask (as specified above), the program terminates and you score 0.

Experimentation

A sample grader `grader.cpp` has been provided. The PRNGs implemented in it are identical to the ones used when judging.

Compilation

If you have `grader.h`, `grader.cpp`, and `solution.cpp` in the same folder, then you can compile your code with this command:

`g++ -DEVAL -O2 grader.cpp solution.cpp -o a.out`

This will produce an executable called `a.out`, which you can run with `./a.out`.

Input

The sample grader will read a single integer specifying the subtask number.

Sample Session

One possible interaction with the sample grader is shown below:

Grader	Student	Description
`solution(1)`		The grader calls your code.
	`guess(0)`	You guess that the next number is 0, but the correct answer was 1.
`guess` returns 1		
	`guess(2)`	You guess that the next number is 2, which is correct.

Because you have guessed correctly, the grader prints the number of calls to `guess` and exits.

PRNG Code

In all of these excerpts, `get_seed` represents a function which returns a uniformly random `uint64_t`. The return value will likely differ between runs of the program.

stdrand

```
uint64_t stdrand()
{
    static bool first = true;
    if(first) {
        std::srand(get_seed());
        first = false;
    }
```

```
    return std::rand();
}
```

hashed

```
uint64_t hashed()
{
    static std::minstd_rand rng(get_seed());
    // what if i just hash it a bunch
    return std::hash<size_t>()(std::hash<size_t>()(std::hash<uint64_t>()(rng())));
}
```

xor16

```
uint64_t xor16()
{
    static const uint64_t xors[16] = {
        2439027017699422455ull,
        956324577728359837ull,
        1707502905498873193ull,
        1393543965786758572ull,
        273899800980839037ull,
        1692732161622750117 5ull,
        1532402782698744333 3ull,
        269310320846006713 8ull,
        867236539875521141 1ull,
        400230189017975920 4ull,
        495687279412305895 4ull,
        153902581294426795 24ull,
        541949959137721753 6ull,
        138295317705086875 22ull,
        960949044087312456ull,
        195057201286594652 9ull,
    };
    static const uint64_t modulo = int(1e9) + 7;
    static uint64_t state = get_seed();
    static int upto = 0;
    // advance state
    state ^= xors[upto];
    upto = (upto + 1) % 16;
    return state % modulo;
}
```

lcg

```
uint64_t lcg()
{
    // initialise state
    static const uint64_t m = int(1e9) + 7;
    static uint64_t a = (get_seed() % (m/2)) + m/4;
    static uint64_t c = (get_seed() % (m/2)) + m/4;
    static uint64_t r = get_seed() % m;
    // advance
    r = (a*r + c) % m;
    return r;
}
```

xorshift

```
uint64_t xorshift()
{
    static uint64_t state = get_seed();
    uint64_t out = state & 0xffff;
    state = state ^ ((state << 11) * 3);
    state = (state >> 16) | (state << 48);
    return out;
}
```

twister

```
uint64_t twister()
{
    static std::mt19937 rng(get_seed());
    return rng();
}
```