# Lucky Numbers

Lucky numbers are subset of integers. Rather than going into much theory, let us see the process of arriving at lucky numbers,

Take the set of integers
1,2,3,4,5,6,7,8,9,10,11,12,14,15,16,17,18,19,……

First, delete every second number, we get following reduced set.
1,3,5,7,9,11,13,15,17,19,……

Now, delete every third number, we get
1, 3, 7, 9, 13, 15, 19,……..

Continue this process indefinitely……
Any number that does NOT get deleted due to above process is called "lucky".

Therefore, set of lucky numbers is 1, 3, 7, 13,……

Now, given an integer 'n', write a function to say whether this number is lucky or not.

```
    bool isLucky(int n)
```

**Algorithm:**
Before every iteration, if we calculate position of the given no, then in a given iteration, we can determine if the no will be deleted. Suppose calculated position for the given no. is P before some iteration, and each Ith no. is going to be removed in this iteration, if P < I then input no is lucky, if P is such that P%I == 0 (I is a divisor of P), then input no is not lucky.

**Recursive Way:**

```
#include <stdio.h>
#define bool int

/* Returns 1 if n is a lucky no. ohterwise returns 0*/
```

```
bool isLucky(int n)
{
  static int counter = 2;

  /*variable next_position is just for readability of
     the program we can remove it and use n only */
  int next_position = n;
  if(counter > n)
    return 1;
  if(n%counter == 0)
    return 0;

 /*calculate next position of input no*/
  next_position -= next_position/counter;

  counter++;
  return isLucky(next_position);
}

/*Driver function to test above function*/
int main()
{
  int x = 5;
  if( isLucky(x) )
    printf("%d is a lucky no.", x);
  else
    printf("%d is not a lucky no.", x);
  getchar();
}
```

**Example:**

Let's us take an example of 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,15,17,18,19,20,21,.....
1,3,5,7,9,11,13,15,17,19,.....
1,3,7,9,13,15,19,.....
1,3,7,13,15,19,.....
1,3,7,13,19,.....

In next step every 6th no .in sequence will be deleted. 19 will not be deleted after this step because position of 19 is 5th after this step. Therefore, 19 is lucky. Let's see how above C code finds out:

| Current function call | Position after this call | Counter for next call | Next Call |
|---|---|---|---|
| isLucky(19 ) | 10 | 3 | isLucky(10) |
| isLucky(10) | 7 | 4 | isLucky(7) |
| isLucky(7) | 6 | 5 | isLucky(6) |
| isLucky(6) | 5 | 6 | isLucky(5) |

When isLucky(6) is called, it returns 1 (because counter > n).

**Iterative Way:**

Please see this comment for another simple and elegant implementation of the above algorithm.