```
/* (c).1993.11.13.John.C.McCarthy
   "S(n).h"

   Example Implementation of A Simple Algorithm to Calculate S(n),
   The Smarandache Function:

   Because there are more people familiar with C than with C++, this module
   has been written entirely in C (apart from "//" style comments).  The
   module was compiled using Borland C++ version 3.1.

   For efficiency, n is constrained to the limits of an unsigned long.  Hence,
   0 <= n <= 2^32 - 1 (= 4,294,967,295).  ("^" represents exponentiation).
   Although catering for n of vast magnitude is possible, it imposes heavy
   storage and processing overheads.  The range of an unsigned long therefore
   seems a reasonable compromise.

   The algorithm depends on the most elementary properties of S(n):
   1) Calculate the STANDARD FORM (SF) of n:
      In SF: n = +/-(p1^a1)*(p2^a2)*...*(pr^ar) where p1, p2,...pr denote the
      distinct prime factors of n and a1, a2,...ar are their respective
      multiplicities.
   2) S(n) = max[ S(p1^a1),...,S(pr^ar)].
   3) S(p^a), where p is prime, is given by:
      3.1) a<=p   ==>  S(p^a) = p*a.
      3.2) a>p    ==>  S(p^a) = x < p*a.  In this case, fortunately rare, x
           is the smallest integer such that p appears as a factor in the
           list of all integers > 1 and <= x at least a times.  Let the no.
           of times p appears as a factor in the list of all integers > 1
           and <= y be f(y, p).  Then:
           f(y, p) = Σ[int(y/(p^i))] for i>0 while y>=(p^i).
           Hence, x is the smallest integer such that f(x, p)>=a.
           Note that between succesive integer multiples of p there are no
           integers which have p as a factor.  The trick here is to look for
           the largest multiple of p (call it c), such that f(p*c, p)<=a
           (so that x = p*c, if f(p*c, p) = a, else x = p*(c+1)):
           3.2.1) c = a-2  (largest possibilty for c since f(p*(a-1), p)>=a
                  when a>p (Note: f(p*(a-1), p)=a is not sought for slight
                  performance gain)).
           3.2.2) z = f(p*c, p).
           3.2.3) While(z>a):
                  3.2.3.1) d = no. of times p appears as a factor of p*c
                             = (no. of times p appears as a factor of c) + 1.
                  3.2.3.2) c = c-1  (next largest possibility for c).
                  3.2.3.3) z = z-d  (= f(p*c, p)).
           3.2.4) If(z<a), x = p*(c+1).
           3.2.5) Else     x = p*c.

   To calculate the prime factors of all 32-bit n requires use only of primes
   < (2^16) (i.e. all primes expressible as an unsigned short integer).  This
   is because any factor of n remaining after division of n by all its prime
   factors < (2^16) is simply a prime.  Since there are only 6542 16-bit
   primes, the program first creates a list of these (which only takes about
   4 seconds on my 20 MHz 386DX PC) so that they never have to be
   recalculated, thus saving much time.
*/
```

```
#define PRIMES16 6542    // The number of 16-bit primes
#define MAX_SFK 9  /* max. distinct primes in the SF of n.  The smallest
    number with more than 9 distinct primes is the product of the 10 smallest
    primes (= 6,469,693,230), which is substantially more than the largest
    integer expressible as an unsigned long.  Hence, 9 distinct primes are
    more than ample.
*/

typedef unsigned long u_long;
typedef unsigned int  u_int;
typedef enum {false, true} boolean;

struct SF_struct {
  int    sfk;              // no. of distinct primes
  u_long sfp[MAX_SFK];     // the distinct primes
  int    sfa[MAX_SFK];     // respective multiplicities
};

extern u_int prime[PRIMES16+1];  // list of all 16-bit primes
                                 // plus terminating zero.

void make_primes(void);  // construct list of all 16-bit primes (prime[]).
                         // Must be called before calls to getSF() or S().
void getSF(u_long n, struct SF_struct *SF); // calc. SF of n and store in SF
u_long S(u_long n);      // calc. S(n)
u_long Spa(u_long p, int a);  // calc. S(p^a) where p is prime
int f(int x, int p);  /* the number of times the prime p appears as a factor
    in the integers from 1 to x inclusive.  This function is only called from
    Spa(p, a) when a>p with x=p*(a-2) (refer to item (3) of algorithm outline
    above).  Max value of (a) occurs when p is a minimum, n is a maximum and
    (p^a)=n.  So, (2^max(a))=max(n)=(2^32)-1.  Hence max(a)<32.  So, x<60
    when (a) is at its max.  Max value of p (and x) occurs when a=p+1 and
    (p^a)=max(n).  So, max(p)^(max(p)+1)=(2^32)-1.  The upshot is that
    max(p)=9 when a=10.  Hence, max(x)=72.  This explains why it is safe for
    x, p and the return value of f(x,p) to be passed as ints.
*/
```