```
#include "linda.h"
#define LIMIT 1000     /* upper bound for limit */

void worker() {
  int primes[LIMIT] = {2,3};  /* table of primes */
  int numPrimes = 1, i, candidate, isprime;

  /* repeatedly get candidates and check them */
  while(true) {
    if (RDP("stop"))  /* check for termination */
      return;
    IN("candidate", ?candidate);     /* get candidate */
    OUT("candidate", candidate+2);  /* output next one */
    i = 0; isprime = 1;
    while (primes[i]*primes[i] <= candidate) {
      if (candidate%primes[i] == 0) { /* not prime */
        isprime = 0; break;
      }
      i++;
      if (i > numPrimes) {  /* need another prime */
        numPrimes++;
        RD("prime", numPrimes, ?primes[numPrimes]);
      }
    }
    /* tell manager the result */
    OUT("result", candidate, isprime);
  }
}

real_main(int argc, char *argv[]) {
  int primes[LIMIT] = {2,3};  /* my table of primes */
  int limit, numWorkers, i, isprime;
  int numPrimes = 2, value = 5;
  limit = atoi(argv[1]);  /* read command line */
  numWorkers = atoi(argv[2]);

  /* create workers and put first candidate in bag */
  for (i = 1; i <= numWorkers; i++)
    EVAL("worker", worker());
  OUT("candidate", value);

  /* get results from workers in increasing order */
  while (numPrimes < limit) {
    IN("result", value, ?isprime);
    if (isprime) {   /* put value in table and TS */
      primes[numPrimes] = value;
      OUT("prime", numPrimes, value);
      numPrimes++;
    }
    value = value + 2;
  }
  /* tell workers to quit, then print the primes */
  OUT("stop");
  for (i = 0; i < limit; i++)
    printf("%d\n", primes[i]);
```

```
}
```

**Figure 7.16** Prime number generation in C-Linda.