```c
/* ================================================= */
/*  FILE:  for_demo.c                                */
/* ================================================= */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define MAX_NUMS 2048
#define FNAME "nums"

/* ================================================= */
void allocate_A( int * * Ap )
{
    int * A ;

    A = (int *) malloc( MAX_NUMS * sizeof(int) ) ;
    if ( A == NULL ) {
       fprintf(stderr,"malloc failed !!\n" ) ;
       exit(1) ;
    }
    *Ap = A ;
}

/* ================================================= */
void read_file( char * fname, int A[], int * np )
{
   FILE * fp    ;
   int n, r, x  ;

   fp = fopen( FNAME, "r" ) ;
   if ( fp == NULL ) {
       fprintf(stderr,"unable to read file '%s'\n", FNAME ) ;
       exit(2) ;
   }
   n = 0 ;
   do {
       r = fscanf( fp, "%d", &x ) ;
       if ( ( r != EOF ) && ( n < MAX_NUMS ) ){
           A[n] = x ;
           n++ ;
       }
   } while ( r != EOF ) ;
   fclose( fp ) ;
   *np = n ;
}
```

```
/* ================================================= */
/*       M A I N ()                                  */
/* ================================================= */
int main()
{
/* Shared variables. */
   int * A ;
   int n   ;
   int total ;

/* Private variables. */
   int ptotal         ;
   int j              ;
   int id, nprocs     ;

#pragma omp parallel shared(n,A,total) private(id,nprocs,j,ptotal)
   {
      nprocs = omp_get_num_threads() ;
      id = omp_get_thread_num()     ;

#pragma omp master
      {
            /* The master thread does this. */
            allocate_A( &A ) ;
            read_file( FNAME, A, &n ) ;

            printf("%d numbers found.\n", n ) ;
            printf("using %d procs.\n", nprocs ) ;

            total = 0 ;   /* Initialize the total. */
      }  /* end section: read-in data */


     /* Every process must wait until thread 0 has updated A, n and total. */
#pragma omp barrier

/*   In this example, we rely on '#pragma omp for' to distribute the */
/*   iteration space.                                                */

   /* Every process computes a local sum. */
   ptotal = 0 ;
#pragma omp for schedule(static,n/nprocs)
   for ( j = 0 ; j < n ; j++ ) {
       ptotal += A[j] ;
   }
   printf( "(%d):  local sum = %d\n", id, ptotal ) ;

/*  Every process contributes its private total to the shared total. */
#pragma omp critical
{
   total += ptotal ;
}

/*  Wait for every process to finish updating 'total'. */
#pragma omp barrier

#pragma omp master
{
     printf("Total = %d\n", total ) ;
}

}  /* End parallel section. */

} /* End main() */
```

```
---------------------------- Sample Session -----------------------------
gottlieb% make
gcc -c -fopenmp -O3 for_demo.c
gcc -o for_demo -fopenmp -O3 for_demo.o

gottlieb% setenv OMP_NUM_THREADS 1
gottlieb% for_demo
1024 numbers found.
using 1 procs.
(0):  local sum = 519632
Total = 519632
gottlieb% setenv OMP_NUM_THREADS 8

gottlieb% for_demo
1024 numbers found.
using 8 procs.
(0):  local sum = 64993
(2):  local sum = 64870
(3):  local sum = 65854
(7):  local sum = 63393
(6):  local sum = 68886
(1):  local sum = 67108
(4):  local sum = 63587
(5):  local sum = 60941
Total = 519632
```