```c
/* File:    trap.c
 * Purpose: Calculate definite integral using trapezoidal
 *          rule.
 *
 * Input:   a, b, n
 * Output:  Estimate of integral from a to b of f(x)
 *          using n trapezoids.
 *
 * Compile: gcc -g -Wall -o trap trap.c
 * Usage:   ./trap
 *
 * Note:    The function f(x) is hardwired.
 *
 * IPP:     Section 3.2.1 (pp. 94 and ff.) and 5.2 (p. 216)
 */
#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include <math.h>
#include <omp.h>

double f(double x);    /* Function we're integrating */
double Trap(double a, double b, int n, double h, double*
total_result);

int main(int argc, char* argv[]) {
   double  integral;   /* Store result in integral   */
   double  a, b;       /* Left and right endpoints   */
   int     n;          /* Number of trapezoids       */
   double  h;          /* Height of trapezoids       */
   int opt = 0;
   int num_thread;
   double total_result = 0.0;

   while ((opt = getopt(argc, argv, "a:b:n:t:")) != -1) {
       switch (opt) {
           case 'a':
               a = strtoul(optarg, (char**) NULL, 10);
               break;

           case 'b':
               b = strtoul(optarg, (char**) NULL, 10);
               break;

           case 'n':
               n = strtoul(optarg, (char**) NULL, 10);
```

```
                break;

            case 't':
                num_thread = strtoul(optarg, (char**) NULL, 10);

            default:
                break;
        }
    }

    h = (b-a)/n;
# pragma omp parallel num_threads(num_thread)

    integral = Trap(a, b, n, h, &total_result);

    printf("With n = %d trapezoids, our estimate\n", n);
    printf("of the integral from %f to %f = %.15f\n",
       a, b, integral);

    return 0;
}  /* main */

/*--------------------------------------------------------------------
 * Function:    Trap
 * Purpose:     Estimate integral from a to b of f using trap rule
 and
 *              n trapezoids
 * Input args:  a, b, n, h
 * Return val:  Estimate of the integral
 */

 double gauss(double x)
 {
      const double A1 = 8000,               a1 = 2;
      const double A3 =  100,  x03 =   1,   a3 =  2;
      const double A4 =   80,  x04 =   1,   a4 =  2;
      const double A5 =   60,  x05 = .01,   a5 =  2;
      const double A6 =   40,  x06 = .01,   a6 =  2;
      const double A7 =   20,  x07 = .01,   a7 =  2;

       return A1 * sin( x / a1 )
           +  A3 * exp( -pow( x - x03, 2.0 / a3 ) )
           +  A4 * exp( -pow( x - x04, 2.0 / a4 ) )
           +  A5 * exp( -pow( x - x05, 2.0 / a5 ) )
           +  A6 * exp( -pow( x - x06, 2.0 / a6 ) )
           +  A7 * exp( -pow( x - x07, 2.0 / a7 ) );
```

```
 91   }
 92
 93  double Trap(double a, double b, int n, double h, double*
  •  total_result) {
 94     double integral;
 95     int k;
 96     double thread_a, thread_b;
 97     int thread_number = omp_get_thread_num();
 98     int thread_n = n/omp_get_num_threads();
 99     thread_a = a + thread_number*thread_n*h;
100     thread_b = thread_a + thread_n*h;
101
102     integral = (gauss(thread_a) + gauss(thread_b))/2.0;
103     for (k = 1; k <= thread_n; k++) {
104       integral += gauss(thread_a+k*h);
105     }
106     integral = integral*h;
107  # pragma omp critical
108     *total_result += integral;
109  }  /* Trap */
110
```