```
/*
 Muti Dimension integration using Monte Carlo
method
 Integration of f(x1, x2, ... xn) on (a[i],b[i])
(i=1,n) intervals
 AG: February 2007, modified by me in 2013
 */
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <vector>

using namespace std;

double f(vector<double>, int);
double int_mcnd(double(*)(vector<double>,int),
vector<double>, vector<double>, int, int);

int main()
{

/*  vector<double> a(6,0.0);
     vector<double> b(6,1.0);
     const int n = 6;       // define how many
```

```
integrals
*/

    int k,n;
    cout << "Enter the dimension of integral" << 
endl ;
    cin >> n ;


    vector<double> a,b;
    double temp;

    cout << "Enter lower limits a_k" << endl;
    for (k=0; k<n; k++)
    {
        cin >> temp ;
        a.push_back(temp);
    }
    cout << "Enter upper limits b_k" << endl;
    for (k=0; k<n; k++)
    {
        cin >> temp ;
        b.push_back(temp);
    }
    double result;
    int i, m;
   int ntimes;

    cout.precision(6);
    cout.setf(ios::fixed | ios::showpoint);
    // current time in seconds (begin
calculations)
    time_t seconds_i;
    seconds_i = time (NULL);

/*   int m =1024;
    result = int_mcnd(f, a, b, n, m);
     cout << result <<endl;
 */

    m = 2;                // initial number of
intervals
    ntimes = 20;          // number of random
points with nmax=2^ntimes
    cout << setw(12) << n <<"D Integral" << endl
;
    for (i=0; i <=ntimes; i=i+1)
    {
        result = int_mcnd(f, a, b, n, m);
        cout << setw(10)  << m << setw(12) <<
result <<endl;
```

```
         m = m*2;
    }

         cout << " " << n <<"D Integral = " <<
result << endl;

     // current time in seconds (end of
calculations)
    time_t seconds_f;
    seconds_f = time (NULL);
    cout << endl << "total elapsed time = " <<
seconds_f - seconds_i << " seconds" << endl <<
endl;



     system ("pause");
     return 0;
}

/*
 *  Function f(x1, x2, ... xk)
 */
double f(vector<double> x, int n)
{
     double y;
     int j;
     y = 0.0;
     /* a user may define the function explicitly
like below */
     //    y = pow((x[0]+x[1]+x[2]+x[3]+...x[n-1]
),2);
     /* or for a specific function like (x1 + x2
+ ... xn)^2
     use a loop over n */
     for (j = 0; j < n; j = j+1)
     {
         y = y + x[j];
     }
     y = pow(y,2);

     return y;
}

/*================================================
================
 Muti Dimension integration of f(x1,x2,...xn)
 method: Monte-Carlo method
 ------------------------------------------------
-----------------
 input:
 fn  - a multiple argument real function
(supplied by the user)
 vector a - left end-points of the interval of
integration
```

```
 vector b - right end-points of the interval of
integration
 n  - dimension of integral
 m  - number of random points
 output:
 r      - result of integration
 Comments:
 be sure that following headers are included
 #include <cstdlib>
 #include <ctime>
 ==============================================
==================*/

double int_mcnd(double(*fn)(vector<double>,int),
vector<double> a, vector<double> b, int n, int m
)

{
    double r,  p;
    vector<double> x (n,0.0); // initialization
of a n-dimensional vector (0.0, 0.0 ,.....0.0,
0.0)
    int i, j;

    srand(time(NULL));  /* initial seed value
(use system time) */
```

```
    r = 0.0;
    p = 1.0;

    // step 1: calculate the volume of the
region of integration
    for (j = 0; j < n; j = j+1)
    {
        p = p*(b[j]-a[j]);
    }

    // step 2: integration
    for (i = 1; i <= m; i=i+1)
    {
        //      calculate random x[] points
        for (j = 0; j < n; j = j+1)
        {
            x[j] = a[j] + (b[j]-a[j])*rand()/
(RAND_MAX+1);
        }
        r = r + fn(x,n);
    }
    r = r*p/m;
    return r;
}

/* Test output
 since the initial seed vary with time the
```

```
results
 may also vary

 6D Integral
 2    7.214014
 4   12.257787
 8   11.070598
 16   11.636560
 32   10.282998
 64    9.203717
 128    9.717596
 256    9.923973
 512    9.921504
 1024    9.665128
 2048    9.670398
 4096    9.632915
 8192    9.632082
 16384    9.549368
 32768    9.530320
 65536    9.527094
 131072    9.500749
 262144    9.500489
 524288    9.504665
 1048576    9.503214
 2097152    9.500961

 */
```