```
#include <iostream>
#include <iomanip>
#include <fstream>
#include "hw61f.h"
using namespace std;
int main(int argc, char* argv[]){
    if (argc < 5){
        cout << endl;
        cout << "Error! You need to specify the initial value, the end-point,
number of subintervals and the output filename.\nNow exiting.\n" << endl;
        return 0;
    }

    double yinit = stod(argv[1]);
    double xinit = 0.0;
    double b = stod(argv[2]);

    int N = stoi(argv[3]);
    double h = (b-xinit)/N;
    ofstream output_file(argv[4]);

        if(!output_file.is_open()){
               cout << endl;
               cout << "Error! Unable to create the specified output file."
<< endl;
               return 0;
        }

        {
               double xval = 0.0;
               double yval = 0.0;

            output_file << setprecision(12) << xinit << " " << yinit;

               for (int i=0; i < N; i++){
                       yval = yinit + h*f(xval,yinit);
                       xval = xinit + h;
                       output_file << '\n' << setprecision(12) << xval << " "
<< yval;


                       yinit = yval;
                       xinit = xval;
               }
        }

        output_file << flush;
        output_file.close();
        cout << "Data file written successfully." << endl;
        cout << endl;
        return 0;
}
```

```cpp
#include <iostream>
#include <iomanip>
#include <random>
#include <functional>
#include "hw62f.h"

using namespace std;

int main(int argc, char* argv[]){

    if (argc < 4){
        cout << endl;
        cout << "Error! You need to specify the interval start and endpoint, and the number of points for Monte-Carlo simulation.\nNow exiting.\n" << endl;
        return 0;
    }

    double start = stod(argv[1]);
    double end = stod(argv[2]);
    long long int points = stoll(argv[3]);

    random_device rd;
    int seed = rd();
    mt19937 generator(seed);
    uniform_real_distribution<double> distribution(start, end);
    auto choose = bind(distribution, generator);

    double xi;
    double I = 0.0;
    double r = (end - start)/points;

    cout << endl;
    cout << "Running Monte-Carlo simulation, please hold on..." << endl;

    for (long long int i=0; i < points; i++){
        xi = choose();
        I += r * f(xi);
    }

        cout << "The approximate integral of the function is: " << setprecision(12) << I << endl;
        cout << endl;

        return 0;
}
```

```
#include <iostream>
#include <iomanip>
#include <random>
#include <cmath>
#include <functional>

using namespace std;
void randompoint(double &x, double &y);

int main(int argc, char* argv[]){

    if (argc < 4){
        cout << "Error! You need to specify the point\'s x and y coordinates,
and the number of trials for Monte-Carlo simulation.\nNow exiting.\n" <<
endl;
        return 0;
    }

    double xval = stod(argv[1]);
    double yval = stod(argv[2]);
    long long int trials = stoll(argv[3]);

    double avgdist = 0.0;
    cout << endl;
    cout << "Running Monte-Carlo simulation, please hold on..." << endl;

    double x;
    double y;
    for (long long int i=0; i < trials; i++){
        randompoint(x, y);
        avgdist += sqrt(pow(xval - x,2.0) + pow(yval - y, 2.0));
    }
    avgdist = avgdist / trials;

        cout << "The approximate expected distance to (" << xval << "," <<
yval << ") is: " << setprecision(12) << avgdist << endl;
        cout << endl;

        return 0;
}

void randompoint(double &x, double &y){
        x = 1.01; y=1.01;
        random_device rd;
    int seed = rd();
    mt19937 seedmaker(seed);
    uniform_real_distribution<double> distrib(-1, 1);
    auto guess = bind(distrib, seedmaker);
        while (pow(x, 2.0) + pow(y, 2.0) >= 1.0){
                x = guess();
                y = guess();
        }
}
```

```
#include <iostream>
#include <iomanip>
#include <random>
#include <cmath>
#include <functional>

using namespace std;

int main(int argc, char* argv[]){

    if (argc < 4){
        cout << endl;
        cout << "Error! You need to specify the point\'s x and y coordinates, and the number of trials for Monte-Carlo simulation.\nNow exiting.\n" << endl;
        return 0;
    }

    double xval = stod(argv[1]);
    double yval = stod(argv[2]);
    long long int trials = stoll(argv[3]);

    random_device rd;
    int seed = rd();
    mt19937 seedmaker(seed);
    uniform_real_distribution<double> distrib(0, 1);
    auto guess = bind(distrib, seedmaker);

    double theta;
    double radius;
    double avgdist = 0.0;
    double pi = acos(-1);

    cout << endl;
    cout << "Running Monte-Carlo simulation, please hold on..." << endl;

    for (long long int i=0; i < trials; i++){
        theta = 2 * pi * guess();
        radius = sqrt(guess());
        double pointX = radius*cos(theta);
        double pointY = radius*sin(theta);
        avgdist += sqrt(pow(xval - pointX,2.0) + pow(yval - pointY, 2.0));
    }
    avgdist = avgdist / trials;

        cout << "The approximate expected distance to (" << xval << "," << yval << ") is: " << setprecision(12) << avgdist << endl;
        cout << endl;

        return 0;
}
```

```
#include <iostream>
#include <iomanip>
#include <random>
#include <cmath>
#include <functional>

using namespace std;

int main(int argc, char* argv[]){

    if (argc < 4){
        cout << endl;
        cout << "Error! Please specify number of dimensions, total number of
steps, and number of simulation trials.\nNow exiting.\n" << endl;
        return 0;
    }

    int dim = stoi(argv[1]);
    long long int steps = stoll(argv[2]);
    long long int trials = stoll(argv[3]);

    if(dim == 0){
        cout << "Number of dimensions must be at least 1.\nNow exiting.\n" <<
endl;
        return 0;
    }

    random_device rd;
    int seed = rd();

    mt19937 bern_generator(seed);
    mt19937 int_generator(seed);

    bernoulli_distribution bern_dist(.5);
    auto pick_direction = bind(bern_dist, bern_generator);

    uniform_int_distribution<int> int_dist(0, dim-1);
    auto pick_axis = bind(int_dist, int_generator);

    cout << endl;
    cout << "Running Monte-Carlo simulation, please hold on..." << endl;

    double avgdist = 0.0;
    int axis;
    for (long long int i=0; i < trials; i++){
        double distance = 0.0;
        vector<int> location (dim);
        for (long long int j=0; j < steps; j++){
            if (dim > 1){
                axis = pick_axis();
            }else{
                 axis = 0;
            }
            int direction = 2*pick_direction() - 1;
            location[axis] = location[axis] + direction;
        }
        for (int k=0; k < dim; k++){
            distance += pow(location[k],2.0);
        }
        distance = sqrt(distance);
        avgdist += distance;
    }
    avgdist = avgdist / trials;

        cout << "The average distance to the origin after " << trials <<  "
trials is: " << setprecision(12) << avgdist << endl;
        cout << endl;

        return 0;
}
```