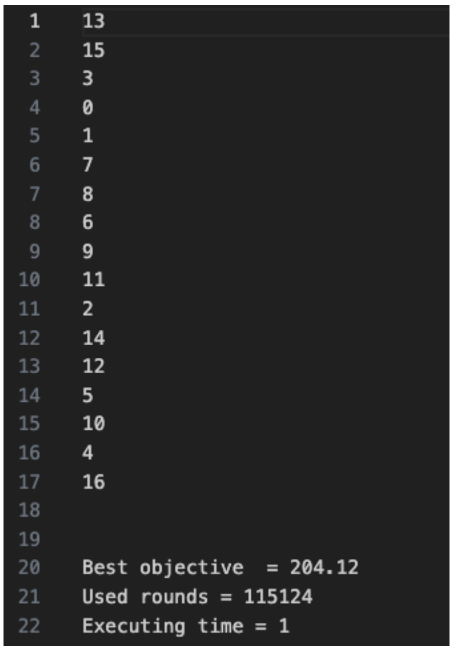
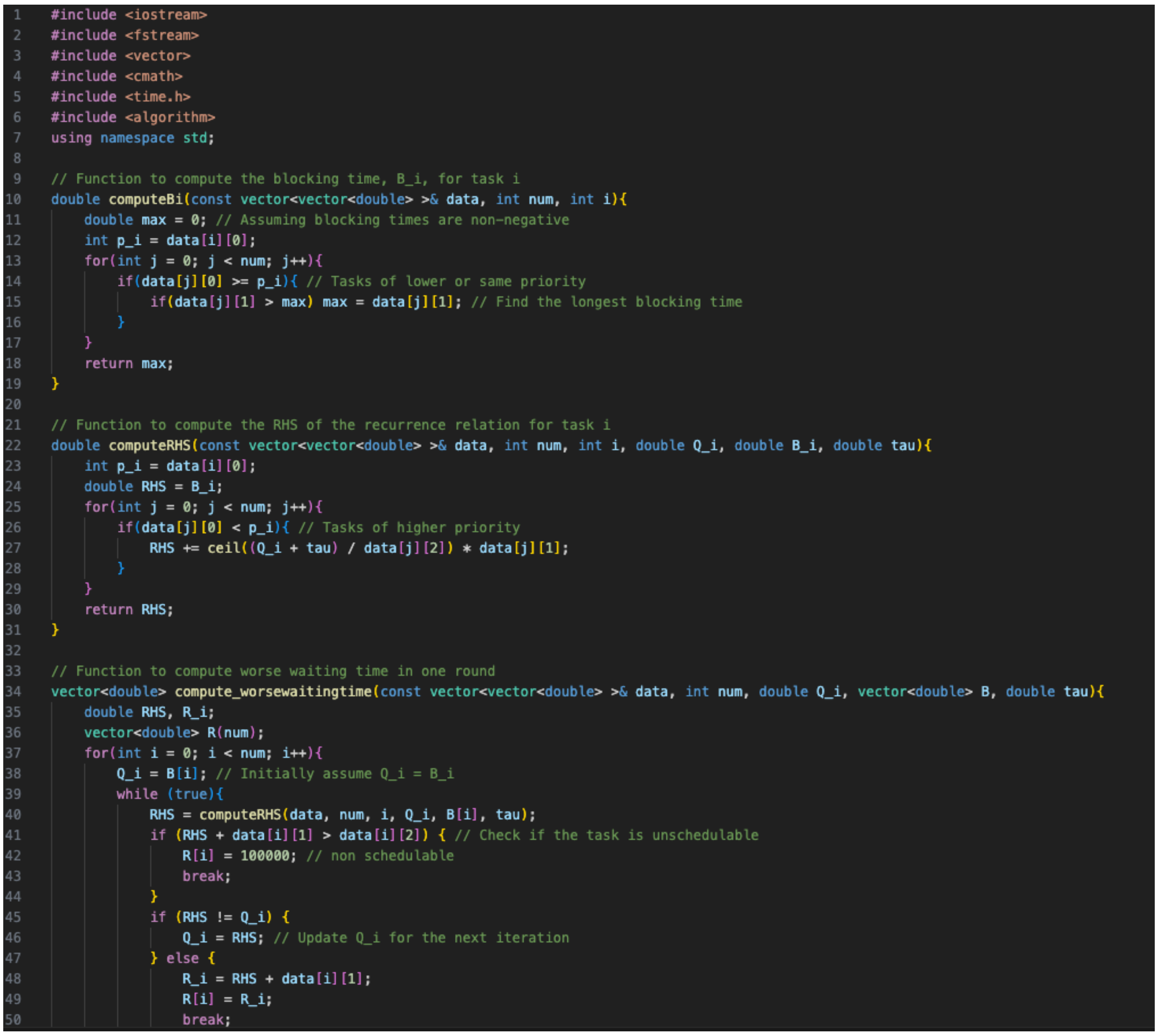
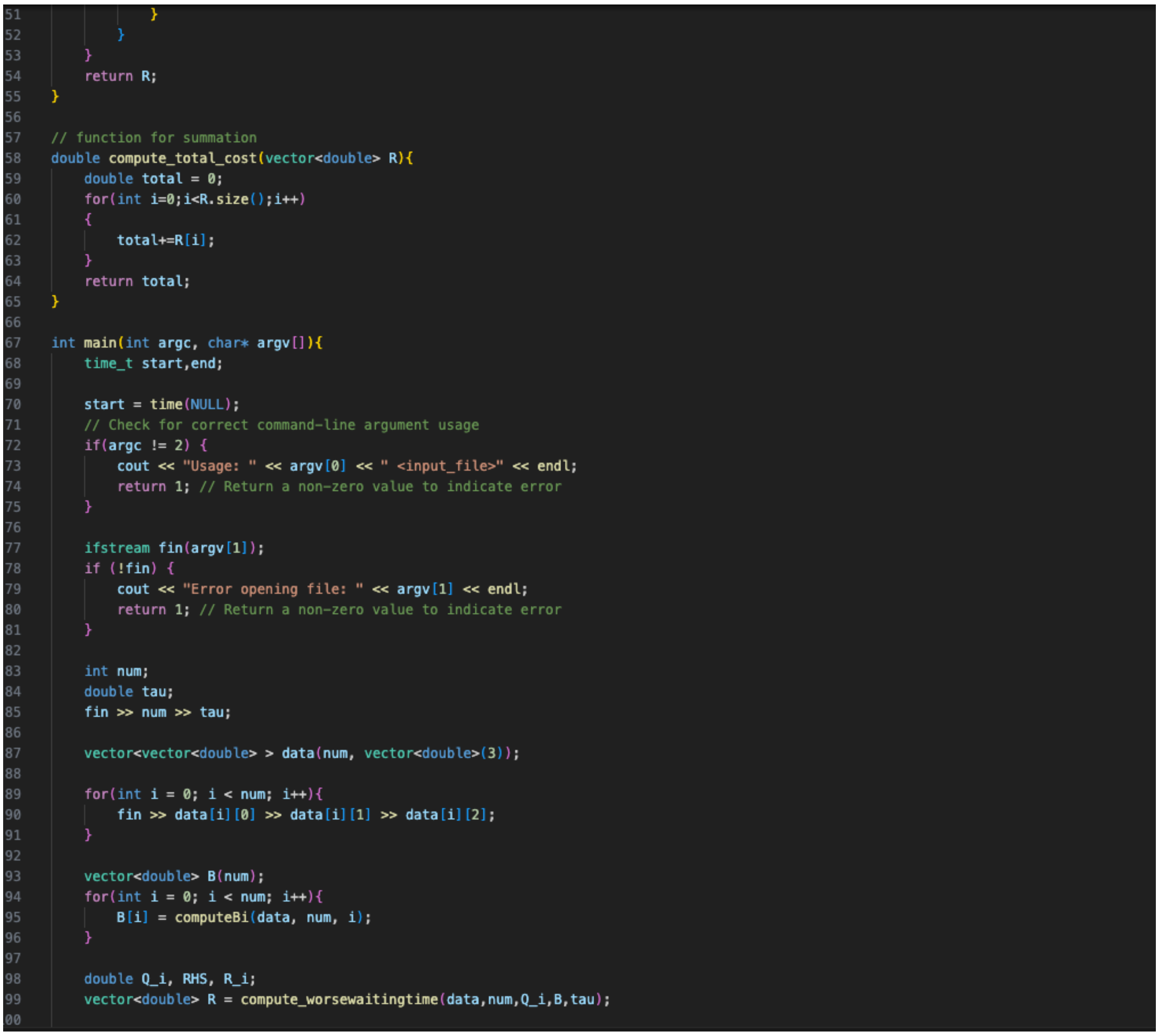


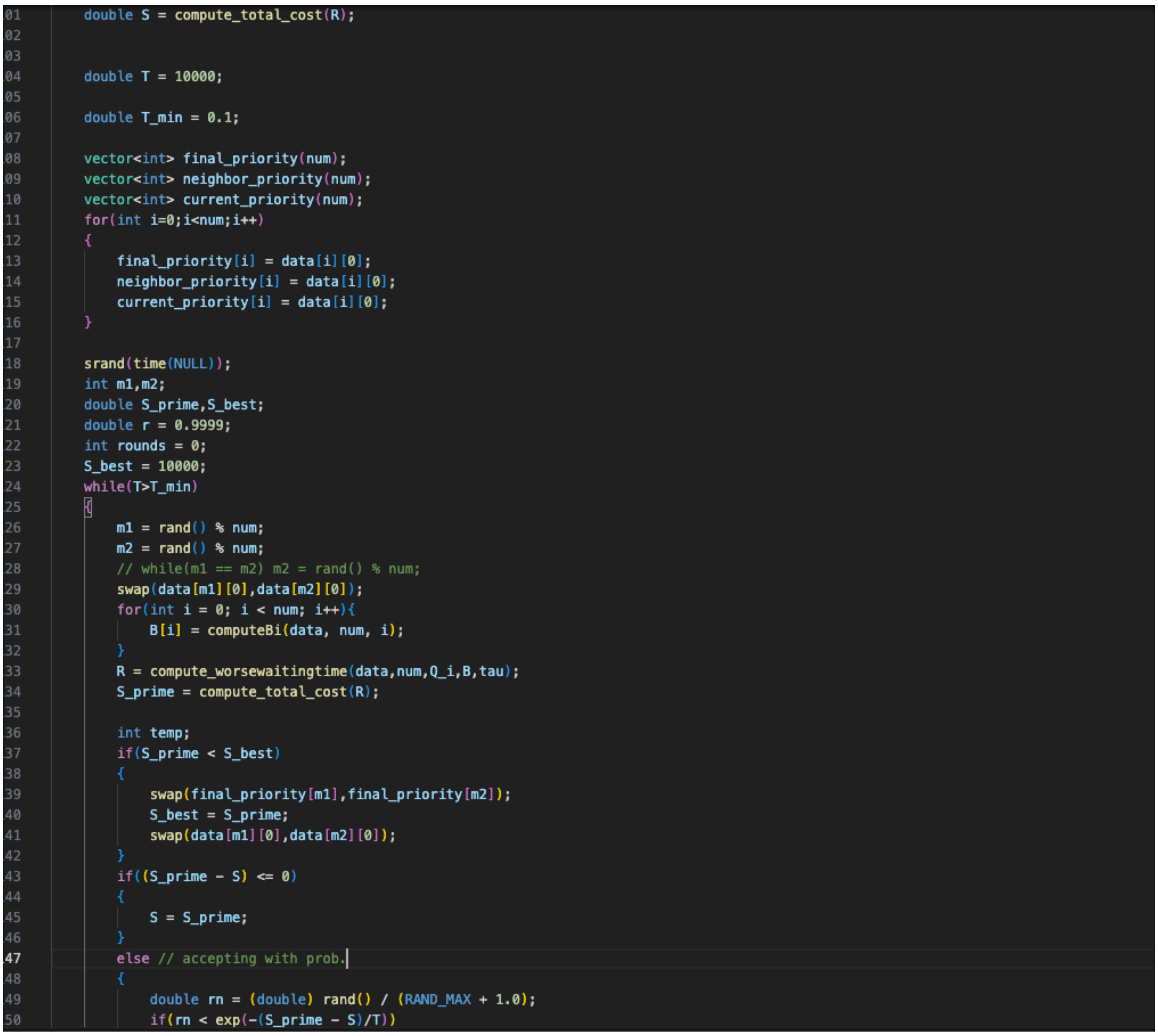
```
13
15
3
0
1
7
8
6
9
11
2
14
12
5
10
4
16


Best objective  = 204.12
Used rounds = 115124
Executing time = 1
```



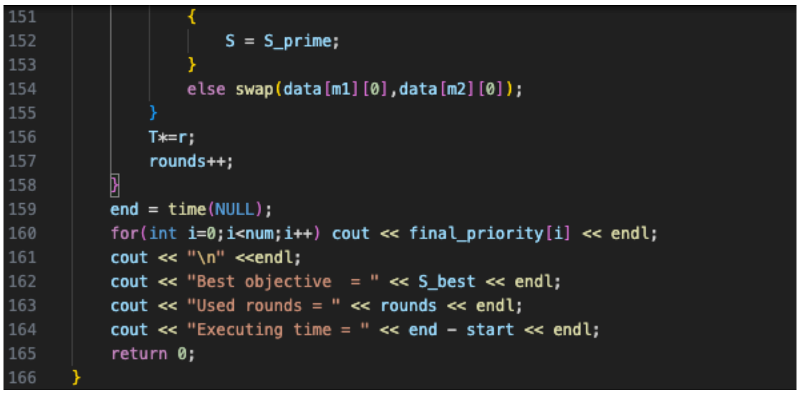
```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <time.h>
#include <algorithm>
using namespace std;

// Function to compute the blocking time, B_i, for task i
double computeBi(const vector<vector<double> >& data, int num, int i){
    double max = 0; // Assuming blocking times are non-negative
    int p_i = data[i][0];
    for(int j = 0; j < num; j++){
        if(data[j][0] >= p_i){ // Tasks of lower or same priority
            if(data[j][1] > max) max = data[j][1]; // Find the longest blocking time
        }
    }
    return max;
}

// Function to compute the RHS of the recurrence relation for task i
double computeRHS(const vector<vector<double> >& data, int num, int i, double Q_i, double B_i, double tau){
    int p_i = data[i][0];
    double RHS = B_i;
    for(int j = 0; j < num; j++){
        if(data[j][0] < p_i){ // Tasks of higher priority
            RHS += ceil((Q_i + tau) / data[j][2]) * data[j][1];
        }
    }
    return RHS;
}

// Function to compute worse waiting time in one round
vector<double> compute_worsewaitingtime(const vector<vector<double> >& data, int num, double Q_i, vector<double> B, double tau){
    double RHS, R_i;
    vector<double> R(num);
    for(int i = 0; i < num; i++){
        Q_i = B[i]; // Initially assume Q_i = B_i
        while (true){
            RHS = computeRHS(data, num, i, Q_i, B[i], tau);
            if (RHS + data[i][1] > data[i][2]) { // Check if the task is unschedulable
                R[i] = 100000; // non schedulable
                break;
            }
            if (RHS != Q_i) {
                Q_i = RHS; // Update Q_i for the next iteration
            } else {
                R_i = RHS + data[i][1];
                R[i] = R_i;
                break;
```



```cpp
            }
        }
    }
    return R;
}

// function for summation
double compute_total_cost(vector<double> R){
    double total = 0;
    for(int i=0;i<R.size();i++)
    {
        total+=R[i];
    }
    return total;
}

int main(int argc, char* argv[]){
    time_t start,end;

    start = time(NULL);
    // Check for correct command-line argument usage
    if(argc != 2) {
        cout << "Usage: " << argv[0] << " <input_file>" << endl;
        return 1; // Return a non-zero value to indicate error
    }

    ifstream fin(argv[1]);
    if (!fin) {
        cout << "Error opening file: " << argv[1] << endl;
        return 1; // Return a non-zero value to indicate error
    }

    int num;
    double tau;
    fin >> num >> tau;

    vector<vector<double> > data(num, vector<double>(3));

    for(int i = 0; i < num; i++){
        fin >> data[i][0] >> data[i][1] >> data[i][2];
    }

    vector<double> B(num);
    for(int i = 0; i < num; i++){
        B[i] = computeBi(data, num, i);
    }

    double Q_i, RHS, R_i;
    vector<double> R = compute_worsewaitingtime(data,num,Q_i,B,tau);

```



```cpp
    double S = compute_total_cost(R);


    double T = 10000;

    double T_min = 0.1;

    vector<int> final_priority(num);
    vector<int> neighbor_priority(num);
    vector<int> current_priority(num);
    for(int i=0;i<num;i++)
    {
        final_priority[i] = data[i][0];
        neighbor_priority[i] = data[i][0];
        current_priority[i] = data[i][0];
    }

    srand(time(NULL));
    int m1,m2;
    double S_prime,S_best;
    double r = 0.9999;
    int rounds = 0;
    S_best = 10000;
    while(T>T_min)
    {
        m1 = rand() % num;
        m2 = rand() % num;
        // while(m1 == m2) m2 = rand() % num;
        swap(data[m1][0],data[m2][0]);
        for(int i = 0; i < num; i++){
            B[i] = computeBi(data, num, i);
        }
        R = compute_worsewaitingtime(data,num,Q_i,B,tau);
        S_prime = compute_total_cost(R);

        int temp;
        if(S_prime < S_best)
        {
            swap(final_priority[m1],final_priority[m2]);
            S_best = S_prime;
            swap(data[m1][0],data[m2][0]);
        }
        if((S_prime - S) <= 0)
        {
            S = S_prime;
        }
        else // accepting with prob.
        {
            double rn = (double) rand() / (RAND_MAX + 1.0);
            if(rn < exp(-(S_prime - S)/T))
```



```cpp
            {
                S = S_prime;
            }
            else swap(data[m1][0],data[m2][0]);
        }
        T*=r;
        rounds++;
    }
    end = time(NULL);
    for(int i=0;i<num;i++) cout << final_priority[i] << endl;
    cout << "\n" <<endl;
    cout << "Best objective  = " << S_best << endl;
    cout << "Used rounds = " << rounds << endl;
    cout << "Executing time = " << end - start << endl;
    return 0;
}
```