C++ code (Basic code for guidance on Optimized and Non-Optimized Route Algorithms)

```
#include <iostream>
using namespace std;
#include <string>
#include <fstream>
#include <time.h>
#include <conio.h>

void routeChosen(int n);
class WasteLocations{
public:
 string locations[8] = {"Waste Collector", "Al Ain", "Abu Dhabi", "Dubai", "Sharjah", "Ajman",
                        "Fujairah", "RAK"};
   int distances[8][8] = {{0,10,12,11,17,20,20,15},
                          {10,0,5,10,16,19,21,25},
                          {12,5,0,5,11,14,16,21},
                          {11,10,5,0,6,9,11,16},
                          {17,16,11,6,0,3,5,10},
                          {20,19,14,9,3,0,2,7},
                          {20,21,16,11,5,2,0,5},
                          {15,25,21,16,10,7,5,0}};

   int random_allocation [8] = {0, (rand() % 100 + 1),(rand() % 100 + 1),(rand() % 100 +
1),(rand() % 100 + 1),(rand() % 100 + 1),(rand() % 100 + 1),(rand() % 100 + 1)};

   void Unoptimized(){
      ofstream myfile;
      myfile.open ("Unoptimized.txt");

      // Variables to calculate relevant costs and cumulative costs
      int x = 0; int dist; int total_dist = 0; double time; double total_time = 0; double fuel;
      double total_fuel = 0; double wage; double total_wage = 0; int counter = 0;

      // Printing the path for >= 40% waste level
      myfile << "Our complete Route of the day \n"<< locations[0] << " -> ";
      for(int i = 1; i < 8; i++){
         if(random_allocation[i] >= 40){
           myfile << locations[i];
           myfile << " -> ";
           counter++;
         }
      }
      // If no location bigger than or equal 40, then no path to be served
      if(counter == 0){
         myfile << " No locations to be served";
      }else{
         myfile << " "<< locations[0]<<endl;
      }
      for(int j = 1; j < 8; j++){
         if(random_allocation[j] >= 40){
            //Initially Checking distance from waste location to first point
            myfile << "\n"<< locations[x] << " to "<< locations[j]<<endl;
```

```
            // Checking matrix for the distance
            dist = distances[x][j];
            time = dist * 1.5;
            fuel = dist * 1.5;
            wage = (time/60) * 5.77;
            myfile << "Distance =  "<<dist<< " km "<<endl;
            myfile << "Time to destination = "<<time<< " minutes"<<endl;
            myfile << "Fuel consumption to destination = "<<fuel<< " RM"<<endl;
            myfile << "Wage for the trip = "<<wage<< " RM\n"<<endl;

            total_dist+=dist;
            total_time+=time;
            total_fuel+=fuel;
            total_wage+=wage;
            x = j;
            counter++;
          }
        }
        if(counter > 0){
          myfile << "\n" <<locations[x] << " to "<< locations[0]<<endl;
          dist = distances[x][0];
          time = dist * 1.5;
          fuel = dist * 1.5;
          wage = (time/60) * 5.77;
          myfile << "Distance =  "<<dist<< " km "<<endl;
          myfile << "Time to destination = "<<time<< " minutes"<<endl;
          myfile << "Fuel consumption to destination = "<<fuel<< " RM"<<endl;
          myfile << "Wage for the trip = "<<wage<< " RM\n"<<endl;

          total_dist+=dist;
          total_time+=time;
          total_fuel+=fuel;
          total_wage+=wage;
          //printing out cumulative costs
          myfile << "Route Costs: "<<endl;
          myfile << "Total Distance Covered = "<< total_dist << " km, " << "Total Time Spent =
" << total_time/60 << " Hours, "<<
          "Total Fuel Consumption = "<< total_fuel << " RM, "<< "Total Wages "<< total_wage
<< " RM"<<endl;
        }
    };
};
class OptimizedRoute : public WasteLocations {
public:
    // Optimized Route
    void optimized(){
        ofstream myfile;
        myfile.open ("Optimized.txt");
        int x = 0; int dist; int total_dist = 0; double time; double total_time = 0; double fuel;
        double total_fuel = 0; double wage; double total_wage = 0; int counter = 0;

        myfile << "Our complete Route of the day \n"<< locations[0] << " -> ";
        for(int i = 1; i < 8; i++){
          if(random_allocation[i] >= 60){
            myfile << locations[i];
```

```
            myfile << " -> ";
            counter++;
        }
    }
    if(counter == 0){
        myfile << " No locations to be served";
    }else{
        myfile << " "<< locations[0]<<endl;
    }
    for(int j = 1; j < 8; j++){
        if(random_allocation[j] >= 60){
            myfile << "\n" <<locations[x] << " to "<< locations[j]<<endl;
            dist = distances[x][j];
            time = dist * 1.5;
            fuel = dist * 1.5;
            wage = (time/60) * 5.77;
            myfile << "Distance =  "<<dist<< " km "<<endl;
            myfile << "Time to destination = "<<time<< " minutes"<<endl;
            myfile << "Fuel consumption to destination = "<<fuel<< " RM"<<endl;
            myfile << "Wage for the trip = "<<wage<< " RM\n"<<endl;

            total_dist+=dist;
            total_time+=time;
            total_fuel+=fuel;
            total_wage+=wage;
            x = j;
            counter++;
        }
    }
    if(counter > 0){
        myfile << "\n" << locations[x] << " to "<< locations[0]<<endl;
        dist = distances[x][0];
        time = dist * 1.5;
        fuel = dist * 1.5;
        wage = (time/60) * 5.77;
        myfile << "Distance =  "<<dist<< " km "<<endl;
        myfile << "Time to destination = "<<time<< " minutes"<<endl;
        myfile << "Fuel consumption to destination = "<<fuel<< " RM"<<endl;
        myfile << "Wage for the trip = "<<wage<< " RM\n"<<endl;
        total_dist+=dist;
        total_time+=time;
        total_fuel+=fuel;
        total_wage+=wage;

        myfile << "Route Costs: "<<endl;
        myfile << "Total Distance Covered = "<< total_dist << " km, " << "Total Time Spent =
" << total_time/60 << " Hours, "<<
            "Total Fuel Consumption = "<< total_fuel << " RM, "<< "Total Wages "<<
total_wage << " RM"<<endl;
    }
    myfile.close();
  }
};
int main() {
```

```
    int c;
    char x;
    bool quit = false;

    srand (time(nullptr));
    while (!quit){
 cout << "Choose the route to be displayed (number)"<<endl<<"0 - Unoptimized Route"<<endl<< "1 - Optimized Route";
        cin >> c;
        routeChosen(c);
        if (c == 0){
            ifstream f("Unoptimized.txt");
            if (f.is_open()) {
                cout << f.rdbuf();
            }
        }else if (c == 1){
            ifstream f("Optimized.txt");
            if (f.is_open()) {
                cout << f.rdbuf();
            }
        }
        // Terminating system statement
        cout << "Press X to quit the system.. Y to continue";
        cin >> x;
        if (x == 'X' || x == 'x'){
            quit = true;
        }else if(x == 'Y' || x == 'y'){
            continue;//a
        }
    }
    return 0;
}

void routeChosen(int n){
        OptimizedRoute R;
    if(n == 0){
        R.Unoptimized();
    }else if(n == 1){
        R.optimized();
    }else{
        cout << "Invalid Choice.."<<endl;
    }
}
```

---------- End of **CW1** Sample Code Guidance ----------