# Courier.cpp

The most recent version

Generated by Doxygen 1.9.6

# Chapter 1

# Introduction

**This program create route file from a central city to all other cities. If the params aren't complete, the program will show an information about lacking params**

## 1.1 Manual

1) Download all files from my repository called Courier_cpp on the GitHub;
2) Launch command prompt;
3) Use command 'cd' and go to files "Courier_cpp", "Courier", "x64", "Debug";
4) Then type in arguments from point at the number 5. (Remember that instead input.txt" type in data file name;
5) Courier.exe -i input.txt -o output.txt;
6) Choose a city where should the centre be.

Author

**Karol Pitera**

Date

**23.02.2023**

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 edge Struct Reference

The scructure includes a information about a given neighbouring city.

```
#include <struct.h>
```

### Public Attributes

- double range
- std::string end

### 4.1.1 Detailed Description

**Parameters**

| | |
|---|---|
| *range* | The distance between the cities. |
| *end* | Neighbouring city. |

Definition at line 22 of file struct.h.

### 4.1.2 Member Data Documentation

#### 4.1.2.1 end

```
std::string edge::end
```

Definition at line 24 of file struct.h.

#### 4.1.2.2 range

```
double edge::range
```

Definition at line 23 of file struct.h.

The documentation for this struct was generated from the following file:

- struct.h

## 4.2 vertex Struct Reference

The structure include a information about the given city.

```
#include <struct.h>
```

### Public Attributes

- std::string previous
- double distance = max
- std::vector< edge > neighbors
- bool visited = false

### 4.2.1 Detailed Description

**Parameters**

| | |
|---|---|
| *previous* | The earlier city that was determined by the algorithm |
| *distance* | Dictance from the given city to the center. |
| *neighbors* | Vector of nieghbouring cities stcrutures. |
| *visited* | Bool value, that include the information about visiting the city. |

Definition at line 36 of file struct.h.

### 4.2.2 Member Data Documentation

#### 4.2.2.1 distance

```
double vertex::distance = max
```

Definition at line 39 of file struct.h.

#### 4.2.2.2 neighbors

```
std::vector<edge> vertex::neighbors
```

Definition at line 40 of file struct.h.

#### 4.2.2.3 previous

```
std::string vertex::previous
```

Definition at line 38 of file struct.h.

#### 4.2.2.4 visited

```
bool vertex::visited = false
```

Definition at line 41 of file struct.h.

The documentation for this struct was generated from the following file:

- struct.h

# Chapter 5

# File Documentation

## 5.1 features.cpp File Reference

complete features file (header, and feature body).

```
#include <iostream>
#include <cmath>
#include <vector>
#include <unordered_map>
#include <fstream>
#include <deque>
#include "features.h"
#include "struct.h"
```

Include dependency graph for features.cpp:

## 5.2 features.cpp

Go to the documentation of this file.

```
#include <iostream>
#include <cmath>
#include <vector>
#include <unordered_map>
#include <fstream>
#include <deque>

#include "features.h"
#include "struct.h"

void service_cmd(std::string & input, std::string& output, int argc, char* argv[]) {
    for (int i = 0; argc > i; i++)
    {
        if (strcmp(argv[i], "-i") == 0) {
            input = argv[i + 1];
        }
        if (strcmp(argv[i], "-o") == 0) {
            output = argv[i + 1];
        }
    }
}

void read_data(std::unordered_map <std::string, vertex> & cities, std::string input) {

    std::ifstream plik(input);
    std::string m1, m2;
    double dist;

    if(plik) {

        while (not plik.eof())
        {
            plik » m1 » m2 » dist;
            cities[m1].neighbors.push_back({dist, m2});
            cities[m2].neighbors.push_back({ dist, m1 });

        }
    }
}
void Dijkstra(std::unordered_map <std::string, vertex> & cities, std::vector <std::string> & unavailable) {

    for (int k = 0; k < cities.size(); k++) {
        double min = max;
        std::string current;

        for (const auto i : cities) {
            if (i.second.distance < min and !i.second.visited) {
                min = i.second.distance;
                current = i.first;
            }
        }

        for (int j = 0; j < cities[current].neighbors.size(); j++) {

            double current_way = cities[current].distance + cities[current].neighbors[j].range;
            std::string neighbour = cities[current].neighbors[j].end;

            if (current_way < cities[neighbour].distance) {
                cities[neighbour].distance = current_way;
                cities[neighbour].previous = current;
            }
        }
        cities[current].visited = true;
    }

        for(const auto i : cities) {
            if (!i.second.visited) {
                unavailable.push_back(i.first);
            }
        }
}
void typing_route(std::unordered_map <std::string, vertex> cities, std::string center,
                  std::vector <std::string>& unavailable, std::string output) {

    std::ofstream file(output);

    for (const auto i : cities) {

        if (i.second.distance < max) {

            std::deque <std::string> queue;

            if (i.first != center) {

                std::string last;
                last = i.second.previous;

                while (last != center) {
                    queue.push_front(last);
                    last = cities[last].previous;
                }
                if (file) {

                    file « center « " -> ";

                    for (int j = 0; j < queue.size(); j++) {
                        file « queue[j] « " -> ";
                    }
                    file « i.first « ": " « i.second.distance « std::endl;
                }
            }
        }
    }
    if (unavailable.size() > 0) {
        file « std::endl « "Cities without connecting to the center: " « std::endl;

        for (int i = 0; i < unavailable.size(); i++) {
            file « "- " « unavailable[i] « std::endl;
        }
    }
    file.close();
}
```

## 5.3 features.h File Reference

Header file.

```
#include <iostream>
#include <cmath>
#include <vector>
#include <unordered_map>
#include <fstream>
#include "struct.h"
```

Include dependency graph for features.h: This graph shows which files directly or indirectly include this file:

### Functions

- void service_cmd (std::string &input, std::string &output, int argc, char *argv[ ])

  *Feature assign a proper params to variables input and output.*
- void read_data (std::unordered_map< std::string, vertex > &cities, std::string input)

  *Feature reads a data from a data file and writes it to the unordered map.*
- void Dijkstra (std::unordered_map< std::string, vertex > &cities, std::vector< std::string > &unavailable)

  *The feature executes the Dijkstra algorithm.*
- void typing_route (std::unordered_map< std::string, vertex > cities, std::string center, std::vector< std::string > &unavailable, std::string output)

  *The feature sorts the cities and previous citis, then the feature saves the program result to the new create file.*

### 5.3.1 Function Documentation

#### 5.3.1.1 Dijkstra()

```
void Dijkstra (
            std::unordered_map< std::string, vertex > & cities,
            std::vector< std::string > & unavailable )
```

Good to know, that at the beginning of the program:

- The city which was choosen the center is assigned the value zero. Whereas all other cities is assigned the distance variable equal to a maximal posible value;
- Every city includes a bool value equal to "false" in the structure variable called "visited", (The value "false" means that given city hasn't been visited by algorithm .

  The algorithm executuion:
  1) The closest unvisited city (from center) is serached and saved to a variable called "current";
  2) The routes from center to all other cities are cheacked, If the distance are shorter than current saved way, the distance is overwritten and city from variable "current" are assigned to structure variable "previaus";
  3) Then city from the variable "current" is marked as visited, and value true is assigned to variable "visited" of structere given city;
  4) The points from 1 to 3 are executed as many time as there are saved a cities in the container (unordered↩
  _map);
  5) The unvisited cities are saved in the vector called "unavailable".

**Parameters**

| | |
|---|---|
| *cities* | The unordered map, there are saved the cities structures, with information about a given cities. |
| *unavailable* | The cities, which haven't got any route connecting to the center. |

Definition at line 88 of file features.cpp.

#### 5.3.1.2 read_data()

```
void read_data (
            std::unordered_map< std::string, vertex > & cities,
            std::string input )
```

First, Data are saved to proper structures
If the structures don't exist then the feature creates them.
Second, Structures are saved to the unordered map.

**Parameters**

| | |
|---|---|
| *cities* | unordered map, include structures with a information about cities from the database |
| *input* | the variable includes a file name with input data. |

Definition at line 48 of file features.cpp.

#### 5.3.1.3 service_cmd()

```
void service_cmd (
            std::string & input,
            std::string & output,
            int argc,
            char * argv[] )
```

Param "-i" inform the feature that a next argument will be assign to the variable "input". The program will do similar with a param "-o", and the variable "output".

**Parameters**

| | |
|---|---|
| *input* | the variable includes a input file name with data |
| *output* | the variable includes output file name with a program result |

Definition at line 26 of file features.cpp.

#### 5.3.1.4 typing_route()

```
void typing_route (
            std::unordered_map< std::string, vertex > cities,
            std::string center,
            std::vector< std::string > & unavailable,
            std::string output )
```

1) A queue is created, then next city and previous cities are saved to the first position the queue until the center is saved;
2) Then, Route from the center, through previous cities and finished on the last city (saved in the variable "i"), the distence information is saved in the end;\ 3) The points from 1 to 2 are performed for each city from the unordered map;
4) All cities which haven't got conected with the center are saved type in to the file with a short information (about not connected);
5) The file is closed.

**Parameters**

| | |
|---|---|
| *cities* | The unordered map, there are saved the cities structures, with information about a given cities. |
| *center* | A city that chose as the center. |
| *unavailable* | The cities, which haven't got any route connecting to the center. |
| *output* | the variable includes output file name with a program result |

Definition at line 135 of file features.cpp.

## 5.4 features.h

Go to the documentation of this file.

```
#pragma once

#include <iostream>
#include <cmath>
#include <vector>
#include <unordered_map>
#include <fstream>

#include "struct.h"
void service_cmd(std::string& input, std::string& output, int argc, char* argv[]);

void read_data(std::unordered_map <std::string, vertex> & cities, std::string input);

void Dijkstra(std::unordered_map <std::string, vertex> & cities, std::vector <std::string> &
      unavailable);

void typing_route(std::unordered_map <std::string, vertex> cities, std::string center, std::vector
      <std::string>& unavailable,
                       std::string output);
```

## 5.5 main.cpp File Reference

File with main feature.

```
#include <iostream>
#include <cmath>
#include <vector>
```

```
#include <unordered_map>
#include <fstream>
#include <deque>
#include "features.h"
#include "struct.h"
```
Include dependency graph for main.cpp:

## Functions

- int main (int argc, char ∗argv[ ])

### 5.5.1 Detailed Description

**Parameters**

| | |
|---|---|
| *argc* | number of typed arguments. |
| *argv* | params contents. |

Definition in file main.cpp.

### 5.5.2 Function Documentation

#### 5.5.2.1 main()

```
int main (
            int argc,
            char ∗ argv[ ] )
```

Definition at line 38 of file main.cpp.

## 5.6 main.cpp

Go to the documentation of this file.
```
00001 #include <iostream>
00002 #include <cmath>
00003 #include <vector>
00004 #include <unordered_map>
00005 #include <fstream>
00006 #include <deque>
00007
00008 #include "features.h"
00009 #include "struct.h"
00010
00038 int main(int argc, char* argv[]) {
00039     std::string input;
00040     std::string output;
00041
00042     service_cmd(input, output, argc, argv);
00043
00044     if (input.empty() || output.empty()) {
00045         std::cout « "no parameters" « std::endl;
```

```

    }

    else {
        std::unordered_map <std::string, vertex> cities;
        std::vector <std::string> unavailable;
        std::string centre;

        read_data(cities, input);

        std::cout « "The program calculate the shortest route between centre and all different cities"
« std::endl « std::endl;
        std::cout « "Availble cities: " « std::endl;

        for (const auto i : cities) {
            std::cout « "- " « i.first « std::endl;
        }

        std::cout « std::endl « "Type city name where should the centre be: ";
        bool exist = false;
        do
        {
            std::cin » centre;
            std::cout « std::endl;

            for (const auto i : cities) {

                if (i.first == centre) {
                    exist = true;
                }
            }
            if (!exist)
                std::cout « "This city was not found, type in correct a city name: ";

        } while (!exist);
        cities[centre].distance = 0;

        Dijkstra(cities, unavailable);

        typing_route(cities, centre, unavailable, output);
    }
}
```

## 5.7 struct.h File Reference

Structures file.

```
#include <iostream>
#include <cmath>
#include <vector>
#include <unordered_map>
```

Include dependency graph for struct.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct edge

  *The scructure includes a information about a given neighbouring city.*
- struct vertex

  *The structure include a information about the given city.*

### Variables

- const double max = std::numeric_limits<double>::max()

### 5.7.1 Detailed Description

**Parameters**

| | |
|---|---|
| *max* | The maximal value mean a infinity |

Definition in file struct.h.

### 5.7.2 Variable Documentation

#### 5.7.2.1 max

```
const double max = std::numeric_limits<double>::max()
```

Definition at line 13 of file struct.h.

## 5.8 struct.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <iostream>
00004 #include <cmath>
00005 #include <vector>
00006 #include <unordered_map>
00007
00013 const double max = std::numeric_limits<double>::max();
00014
00015
00022 struct edge {
00023     double range;
00024     std::string end;
00025 };
00026
00027
00036 struct vertex {
00037
00038     std::string previous;
00039     double distance = max;
00040     std::vector <edge> neighbors;
00041     bool visited = false;
00042 };
```

# Index