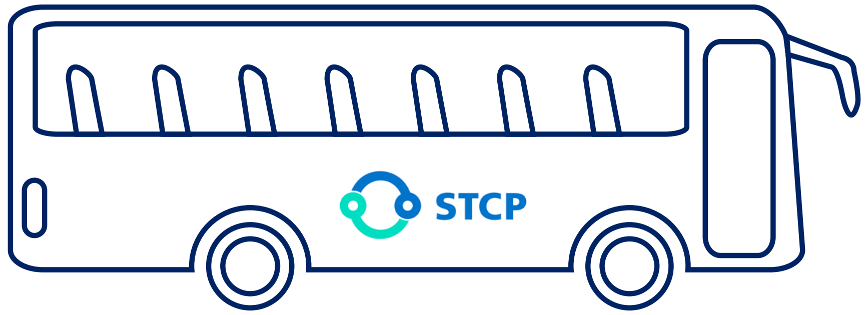
# NAVEGAÇÃO NOS TRANSPORTES PÚBLICOS DO PORTO

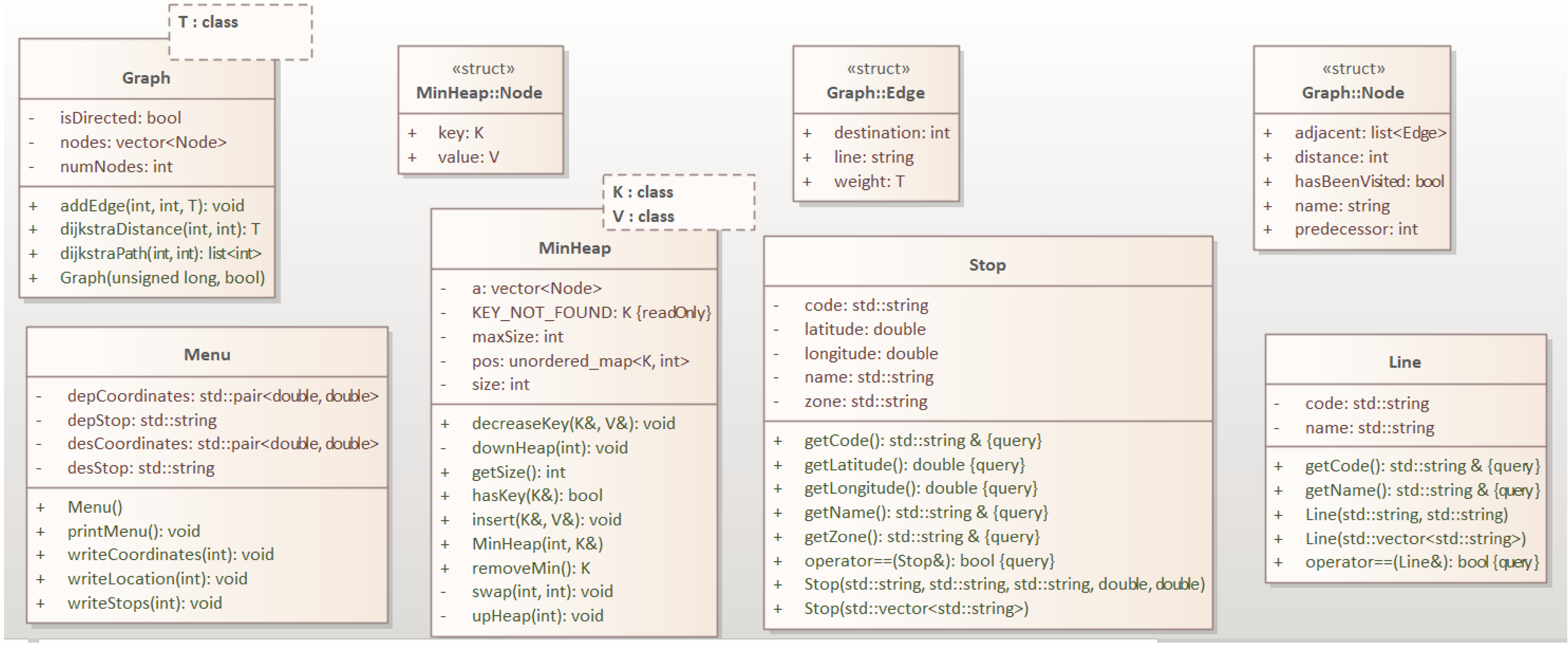
Trabalho prático 2
AED 21/22
L.EIC - 2ºano

Afonso Pinto - up202008014
Bruna Marques - up202007191
Miguel Curval - up201105191



U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# Diagrama de classes



```
T : class
Graph
-  isDirected: bool
-  nodes: vector<Node>
-  numNodes: int
+  addEdge(int, int, T): void
+  dijkstraDistance(int, int): T
+  dijkstraPath(int, int): list<int>
+  Graph(unsigned long, bool)
```

```
Menu
-  depCoordinates: std::pair<double, double>
-  depStop: std::string
-  desCoordinates: std::pair<double, double>
-  desStop: std::string
+  Menu()
+  printMenu(): void
+  writeCoordinates(int): void
+  writeLocation(int): void
+  writeStops(int): void
```

```
«struct»
MinHeap::Node
+  key: K
+  value: V
```

```
K : class
V : class
MinHeap
-  a: vector<Node>
-  KEY_NOT_FOUND: K {readOnly}
-  maxSize: int
-  pos: unordered_map<K, int>
-  size: int
+  decreaseKey(K&, V&): void
-  downHeap(int): void
+  getSize(): int
+  hasKey(K&): bool
+  insert(K&, V&): void
+  MinHeap(int, K&)
+  removeMin(): K
-  swap(int, int): void
-  upHeap(int): void
```

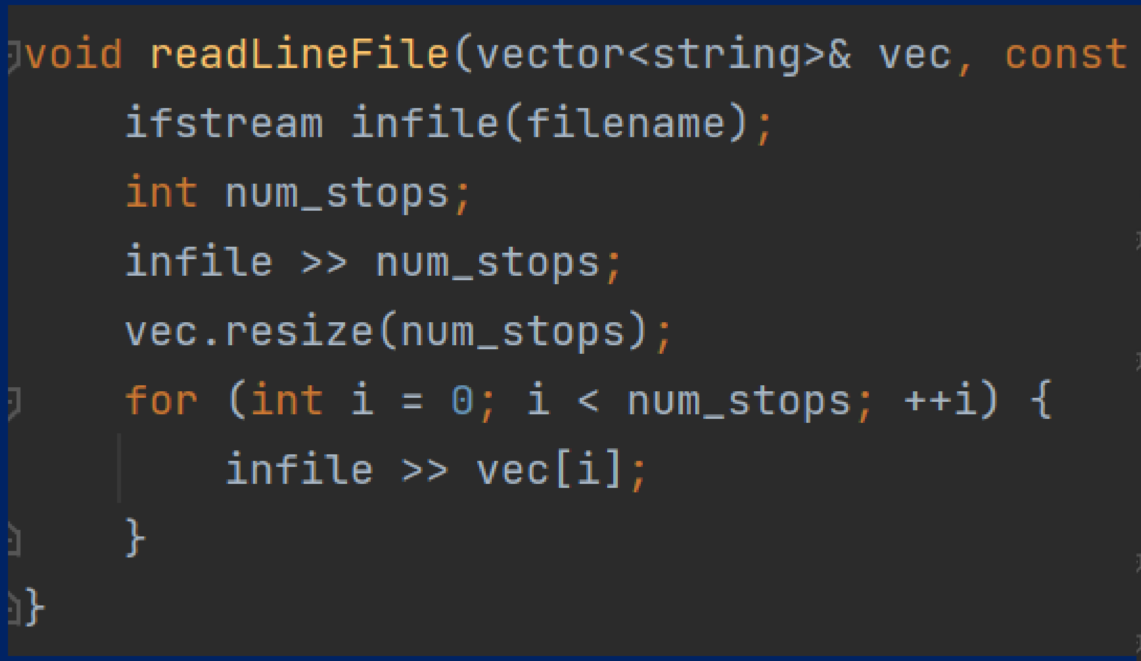
```
«struct»
Graph::Edge
+  destination: int
+  line: string
+  weight: T
```

```
Stop
-  code: std::string
-  latitude: double
-  longitude: double
-  name: std::string
-  zone: std::string
+  getCode(): std::string & {query}
+  getLatitude(): double {query}
+  getLongitude(): double {query}
+  getName(): std::string & {query}
+  getZone(): std::string & {query}
+  operator==(Stop&): bool {query}
+  Stop(std::string, std::string, std::string, double, double)
+  Stop(std::vector<std::string>)
```

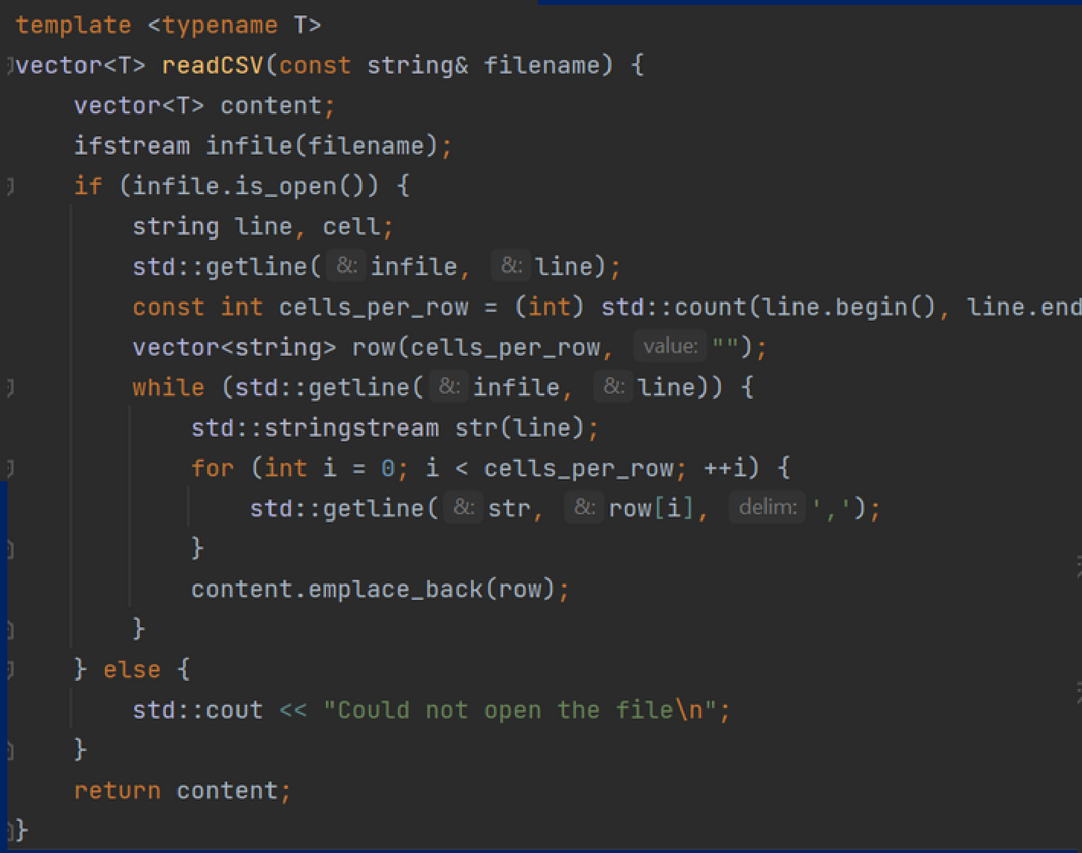
```
«struct»
Graph::Node
+  adjacent: list<Edge>
+  distance: int
+  hasBeenVisited: bool
+  name: string
+  predecessor: int
```

```
Line
-  code: std::string
-  name: std::string
+  getCode(): std::string & {query}
+  getName(): std::string & {query}
+  Line(std::string, std::string)
+  Line(std::vector<std::string>)
+  operator==(Line&): bool {query}
```

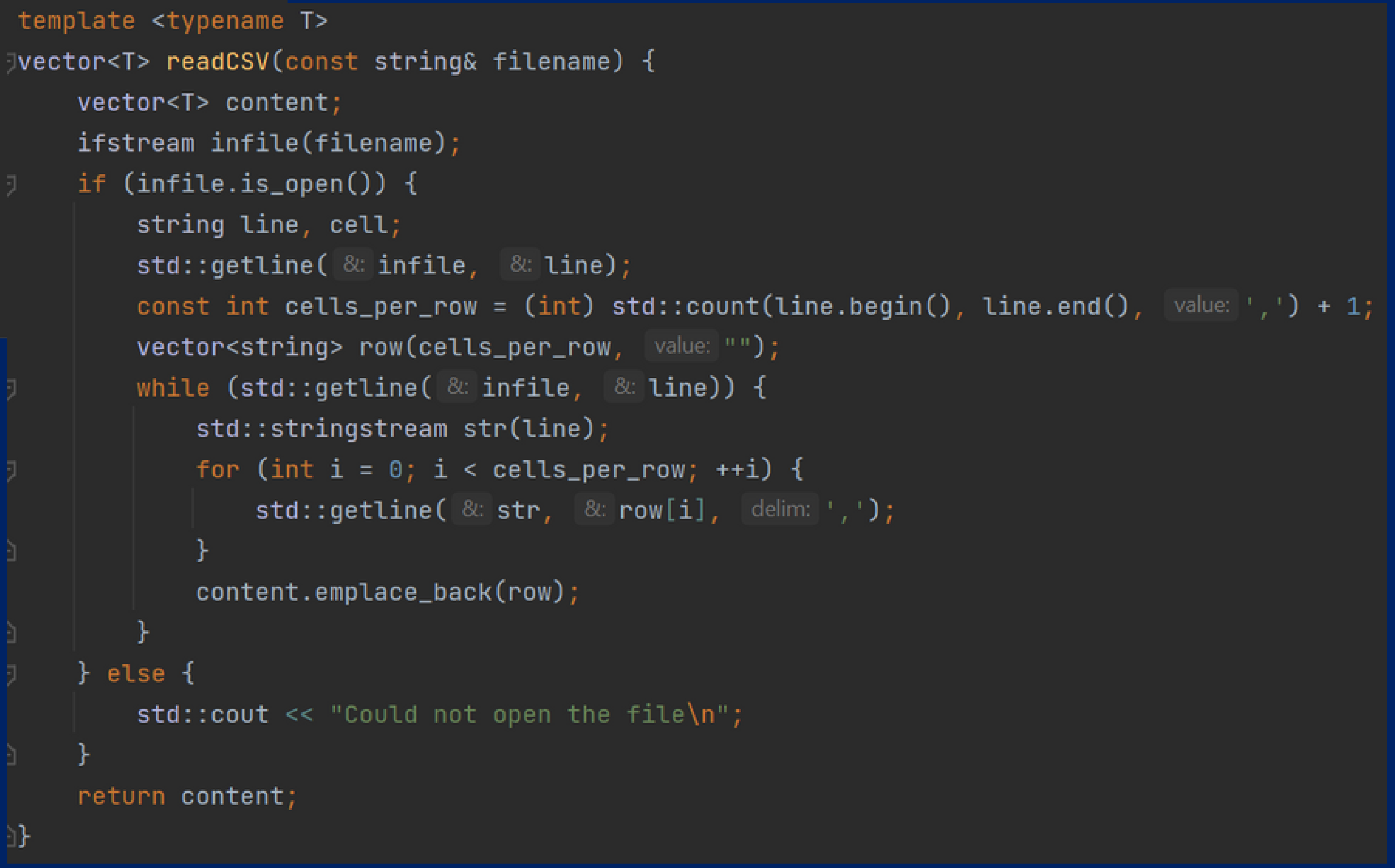
# Leitura do dataset



```
void readLineFile(vector<string>& vec, const string& filename) {
    ifstream infile(filename);
    int num_stops;
    infile >> num_stops;
    vec.resize(num_stops);
    for (int i = 0; i < num_stops; ++i) {
        infile >> vec[i];
    }
}
```

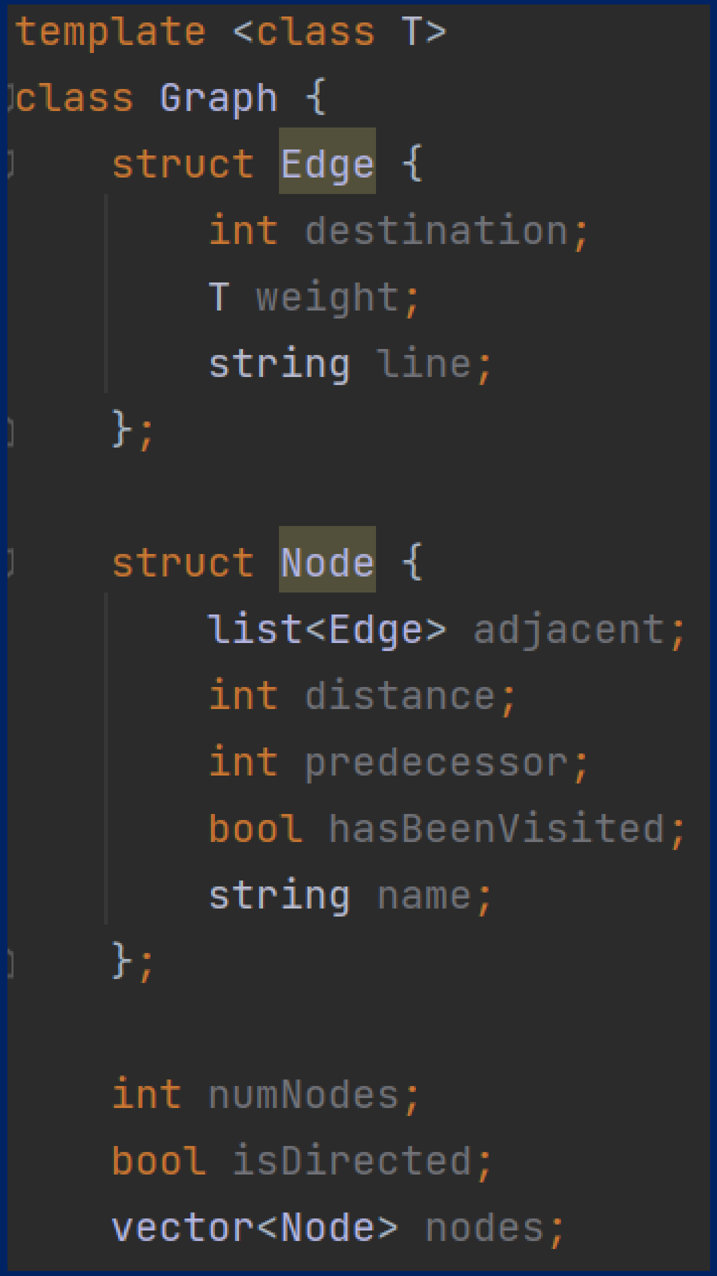


```
template <typename T>
vector<T> readCSV(const string& filename) {
    vector<T> content;
    ifstream infile(filename);
    if (infile.is_open()) {
        string line, cell;
        std::getline( &: infile,  &: line);
        const int cells_per_row = (int) std::count(line.begin(), line.end(),  value: ',') + 1;
        vector<string> row(cells_per_row,  value: "");
        while (std::getline( &: infile,  &: line)) {
            std::stringstream str(line);
            for (int i = 0; i < cells_per_row; ++i) {
                std::getline( &: str,  &: row[i],  delim: ',');
            }
            content.emplace_back(row);
        }
    } else {
        std::cout << "Could not open the file\n";
    }
    return content;
}
```

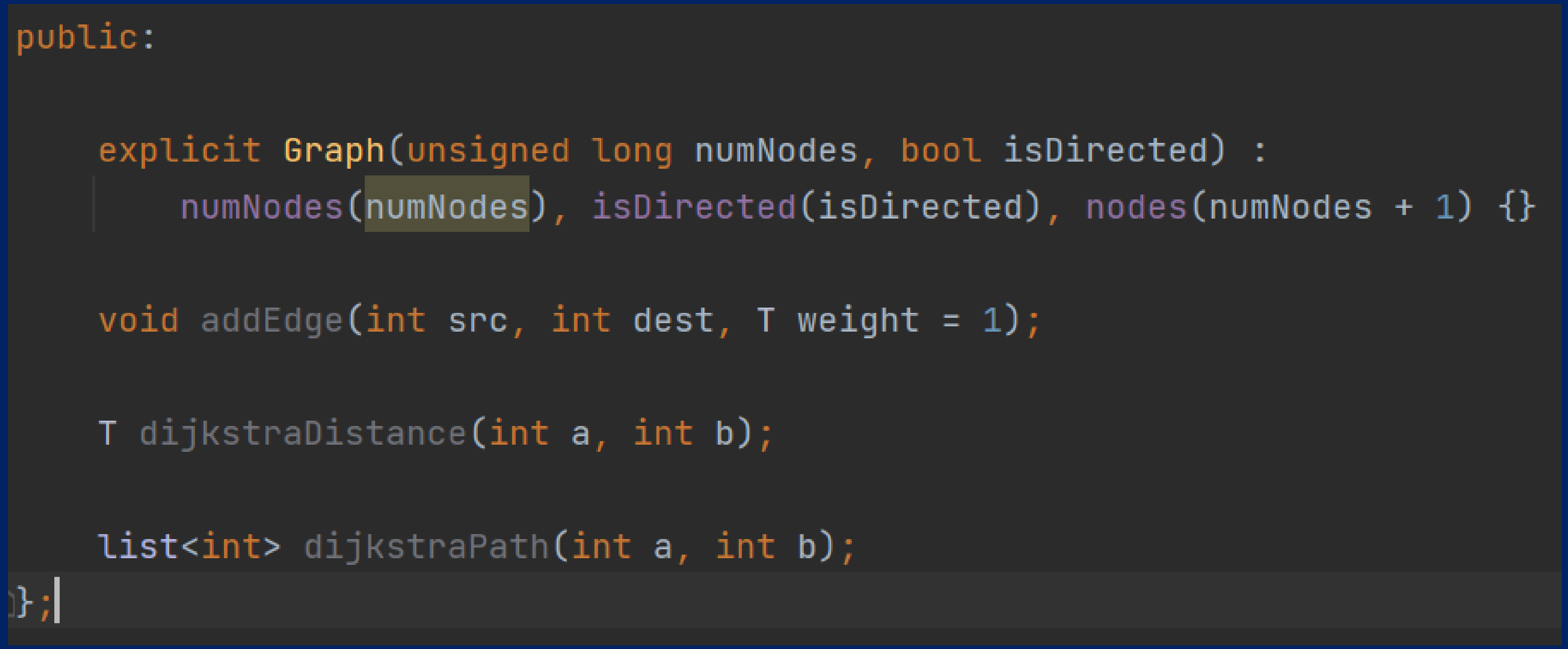


```
template <typename T>
vector<T> readCSV(const string& filename) {
    vector<T> content;
    ifstream infile(filename);
    if (infile.is_open()) {
        string line, cell;
        std::getline( &: infile,  &: line);
        const int cells_per_row = (int) std::count(line.begin(), line.end(),  value: ',') + 1;
        vector<string> row(cells_per_row,  value: "");
        while (std::getline( &: infile,  &: line)) {
            std::stringstream str(line);
            for (int i = 0; i < cells_per_row; ++i) {
                std::getline( &: str,  &: row[i],  delim: ',');
            }
            content.emplace_back(row);
        }
    } else {
        std::cout << "Could not open the file\n";
    }
    return content;
}
```

# Grafo usado para representar o dataset

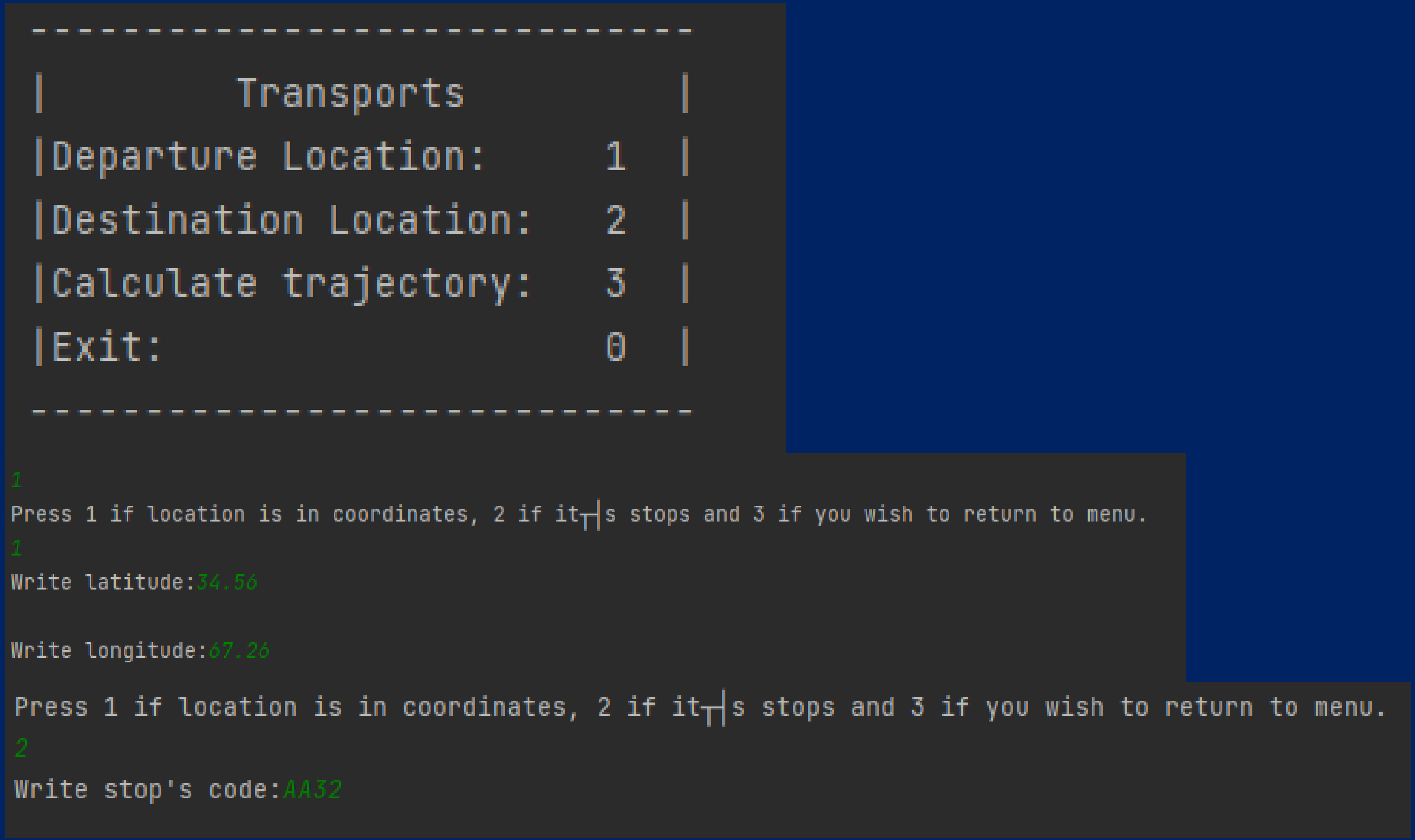


```cpp
template <class T>
class Graph {
    struct Edge {
        int destination;
        T weight;
        string line;
    };

    struct Node {
        list<Edge> adjacent;
        int distance;
        int predecessor;
        bool hasBeenVisited;
        string name;
    };

    int numNodes;
    bool isDirected;
    vector<Node> nodes;
```



```cpp
public:

    explicit Graph(unsigned long numNodes, bool isDirected) :
        numNodes(numNodes), isDirected(isDirected), nodes(numNodes + 1) {}

    void addEdge(int src, int dest, T weight = 1);

    T dijkstraDistance(int a, int b);

    list<int> dijkstraPath(int a, int b);
};
```

# Interface



```
-----------------------------

|        Transports         |
|Departure Location:      1  |
|Destination Location:    2  |
|Calculate trajectory:    3  |
|Exit:                    0  |

-----------------------------
```

```
1
Press 1 if location is in coordinates, 2 if it┬┤s stops and 3 if you wish to return to menu.
1
Write latitude:34.56

Write longitude:67.26
```

```
Press 1 if location is in coordinates, 2 if it┬┤s stops and 3 if you wish to return to menu.
2
Write stop's code:AA32
```

# Funcionalidades

# Destaque de funcionalidade

# Dificuldades e esforço dos elementos do grupo

dificuldade...

| | |
|---|---|
| Afonso Pinto | 33% |
| Bruna Marques | 33% |
| Miguel Curval | 33% |