Vendula Pokludová

Documentation – Tram Timetables

User Part

Brief Description

The program uses a graphical user interface to display types of tram timetables. The user can choose between three actions: the full timetable for a selected tram, departures for a chosen tram and specified stop, and the next ten departures from the chosen stop for a selected time.

Detailed Description of Use

The user has dropdown menus available, from which they can select a tram, stop, day (weekday, Saturday, Sunday), hour, minute, and action. After pressing the enter button, the selected action is executed, and the output is displayed in the same window (application). The program takes the currently selected options from all dropdown menus and performs the corresponding action. After starting the program, the first element from the list of options is automatically selected for each dropdown menu, and it can never happen that any menu does not have a valid choice selected. The output is displayed in a textbox in the application, and because some outputs are very long, the user can scroll up or down and left or right in the text box.

Programming Part

Chosen Algorithm

Since the program does not have access to a large amount of memory, it was important to store as little data as possible for each tram. The data had to be stored in a way that allowed quick timetable generation and in a way that allowed simple creation of the data files.

Representation of Input Data and Preparation of Input Data

For each tram we must know:

- All departures from the first terminal (*startA*)
- All departures from the second terminal (*startB*)
- The duration of the ride (in minutes) from *startA* to each stop between *startA* and *startB*
- The duration of the ride (in minutes) ze *startB* to each stop between *startB* and *startA*

Since the current data for trams in Prague is available at www.pid.cz in the form of a complete timetable, it was fairly easy to copy all departure times from the terminal stops and also relatively easy to calculate the duration of the ride to each stop.

It is important to note that some tram stops are only present for trams going in one direction of travel. In the data files this is addressed using the character *x* in place of duration time to that stop.

The structure of the data files was chosen using the above information and reasoning and therefore the data files look like this:

Tram Number

Start A
Start B

Start A ; Duration from start A ; Duration from start B
…Analogously for all stops that occur between A and B…
Start B ; Duration from start A ; Duration from start B

All departure times from start A on weekdays
All departure times from start A on Saturdays
All departure times from start A on a Sundays

All departure times from start B on weekdays
All departure times from start B on Saturdays
All departure times from start B on Sundays

Chosen Algorithm (Continued)

According to the file structure, which is shown above, it is easy to see how the program generates the timetable:

1. As input parameters, it takes the current values in the dropdown menus and uses those to determine which type of timetable and for what values (tram, stop, time) the user wants to print information for.
2. Then the program will iterate over the departure times in the corresponding data file(s) and
    a. For each stop the duration of ride is added to the departure time to get the time of departure for that stop

Representation of Output Data and its Interpretation

The output appears in a graphical interface in a separate window, which is implemented by the wxWidgets library. In the application, the user can input a tram, stop, day, time, and action they want to be performed. Once the user has chosen, they can press the "Enter" button, and the result will appear on the lower half of the same window.

Program

Classes:

Time

Is used for calculations involving times and also for saving time elements in data structures.

Properties

- *hours* – integer [0, 24]
- *minutes* – integer [0, 59]

Methods

- Operator +

  Allows us to calculate times of departure from non-terminal stops by adding duration of travel to the time of departure from terminal stop.

- Operator <,>,=,>=

  Instances of this class must be comparable, so that we can use a map data structure (of type <*Time*, *std::string*>) during the search for the 10 closest departures from a given stop at the specified time.
- *toString* – returns an instance of *std::string,* which looks like: *hours : minutes*

Data

enum *day_type* — to determine the type of day (weekday, Saturday, Sunday); is used when choosing which lines of departure times to read/ process in the data file(s).

Tram structure

startA je *std::string,* name of terminal stop A

startB je *std::string,* name of terminal stop B

departuresA a map, where keys are *day_type* and values are a vector of all departures from ze *startA* on that day

departuresB, analogous to previous property, except from *startB*

stops is a vector of all stops from s*tartA* to *startB*

timesA is an *unordered_map<std::string, int>,* for quick access to durations of rides (values) from *startA* to the given stop (key)

timesB, analogous to *timesA,* but from *startB*

Properties:

stops — a map where keys are different tram stops and the value is a list of trams (identified by their number) that stop at that stop.

trams — a map where keys are integers (tram identifiers) and the corresponding value is the created instance of the *Tram* structure that relates to that tram number.

stopColumn The width of the first column in the output table (which holds names of tram stops); value is determined by the length of the longest tram stop name.

Metody:

addTram Takes the filename as an input argument and tries to open the file with that name and process the data in the file into a new instance of the *Tram* structure, which is then added to the *Data* class, if the file is written correctly.

get_times From the string that was read from the file in the *addTram* method, creates and returns a vector of instances of the *Time* class. The input strings should be in the form "HH:MM HH:MM … HH:MM HH:MM", which is a list of times where the delimiter is a single space.

get_tokens Splits an input string into segments or tokens based on a specified delimiter and adds these tokens to a vector, which is then returned. This provides an efficient way to organize and handle segmented string data which is present in the data files.

tram_full_timetable

The input parameters are tram number and day type, which determine which departure times should be used when generating the timetable string. Takes the appropriate departures from *startA* and for each stop adds journey duration. These values are formatted to create a string, that when printed, resembles a timetable. The same is done for departures from *startB* and the resulting strings are concatenated and returned.

fill_first_column

Modifies the input string (which already contains the name of the stop) to be of the correct length by adding spaces. The correct length is the same length as the longest stop name. The method is called in the *tram_full_timetable* method to aid in formatting the output string.

Tram_stop_timetable

The method takes tram number, day type, and stop name as input. It begins by addressing the travel direction from *startA* to *startB*. All departures from *startA* are gathered, and travel time to the selected stop is added to determine departure times from that stop in the *startA* to *startB* direction. The output is a string formatted like a timetable, with the first column showing hours and rows

listing minutes of tram departures from the stop in one direction. An analogous process is applied for the reverse direction, generating a similar timetable string. These strings are then concatenated and returned, providing a comprehensive timetable view for both directions.

stop_time_timetable

The method takes stop name, day type, and time as inputs. It iterates through trams serving the specified stop, using the *find* method to pinpoint the first ten departures for each tram. Once all pertinent trams are processed, their departures are collated into a map, keyed by departure time with values combining tram number and travel direction. The method then extracts the earliest ten departures from this map. The output is a string, formatted like a timetable, utilizing the key-value pairs from the map and presenting them as a consolidated schedule, which is then returned.

find The method uses a *Tram* instance, stop name, day type, and time as inputs to create a map, aiming to have up to ten entries. Departure times serve as keys, and the values are strings indicating travel direction. It populates the map with the first ten departures from the chosen stop, calculating these by adding the travel time from the terminal. The process stops if it reaches the end of the departure list, which can result in less than ten entries.

get_trams_vector, get_stops_vector, get_days_vector

Produces vectors of strings suitable for use in wxChoice instances within the graphical interface.

To implement the graphical UI, the following classes are required from the wxWidgets library:

App It represents an application instance, referred to as *wxApp* in the *wxWidgets* library. This instance features an *OnInit*() method that initializes the *MainFrame*. For this initialized *MainFrame*, it employs *SetClientSize*, *Center*, and *Show* methods, which respectively adjust the window's size, center it on the screen, and make the application visible.

MainFrame

Constructor:

Creates instances of panels, *panel1* (which contains objects used by the user to input data) and *panel2* (which includes *wxTextCtrl* for output), a button, *wxButton*, to signal the program that the user wants the selected action to be executed. It also creates all *wxChoice* objects, which allow the user to select the tram, stop, type of day, time (hours, minutes), and action.

OnButtonClicked

The method is called when the user presses the button labeled “Enter”. It saves all currently selected options from all *wxChoice* objects and, based on the chosen action, calls the corresponding method from the *Data* class with the correct parameters. Then, the *setOutput* method is called with the resulting string.

setOutput

Erases the existing string in the *wxTextCtrl* object using the *Clear* method, then appends the text provided in the input parameter.

Overall Project Evaluation

Reflecting on the overall project process, the most significant challenge was the optimal preparation of input files and efficient data storage within the program, ensuring the setup wasn't overly complex while maintaining ease and speed in calculations. The rest of the project unfolded more smoothly. Utilizing the *wxWidgets* library, which boasts comprehensive documentation and a diverse range of objects, greatly facilitated implementation. Nonetheless, some aspects of the project could have been managed differently. This includes better time allocation for specific tasks, pinpointing areas where effort could have been either scaled back or intensified, and more effectively identifying aspects that either met or fell short of expectations.