Vallero Marzio s249920

Report for the Algorithms and Programming Exam of 22nd February 2019

I have created two main structures to handle the museums and the visitors data. Both data structures are implemented using Binary Search Trees, in order to undergo the requirement for worst case logarithmic speed in the processing of reservation requests and of printing all reservations made by any one visitor.

The *struct wrapper_m_t* is used to handle the struct museum_t in the operations regarding BST search and insertion, and to point to the actual *struct museum_t*. I did this in order to not modify the original *struct museum_t* for BST manipulation, even though I intended to treat *struct museum_t* in a similar way to *struct visitor_t*. Within the *struct museum_t*, we find the musuem's name and other nested structures, first of which is used to save an array *struct days_t* "calendar". This array has an entry for each day of the year, that is from 0 to 364. Within it we have the day's number (which in fact is not needed since the day is equal to the calendar array's index, but useful if we wanted to represent dates as strings, or with format dd/mm) and a pointer to a list of *struct time*, that contains a list with all the time slots for a museum.
The *struct_time* *timeslotsheadand holds the time slots threshold and time inteval data.

The *struct visitor_t* contains all the data regarding a given visitor's reservation. It uses "left" and "right" pointers to manipulate the visitorsBST, pointing to reservations made by other people, while using a pointer "next" to enlist all other reservations made by the same person.

The *main()* function can be subdivided into three parts:

I. Checks for the input filename to be present, then initializes the *char* and *int* variables used throughout main and then initializes the visitorsBST to NULL and dynamically allocates loads the museumsBST with the data present in the source file through the function *load_system()*.

II. A switch menu has been put in place of the if-then-else statements used in the written exam for ease of correction and of use. The menu asks to the user what he/she wants to do, and calls the relative functions to perform such operations. *memset()* and *fflush()* functions have been added to improve security and avoid long runtime errors, with the buffer array getting messy and what not.
A "Close" command has been added. These modifications with respect to the original written exam do not interfere with how the program works, but rather are there for the sake of completeness and for ease of correction purposes.

III. Once the menu is closed, the data structures are freed from memory and the program closes.

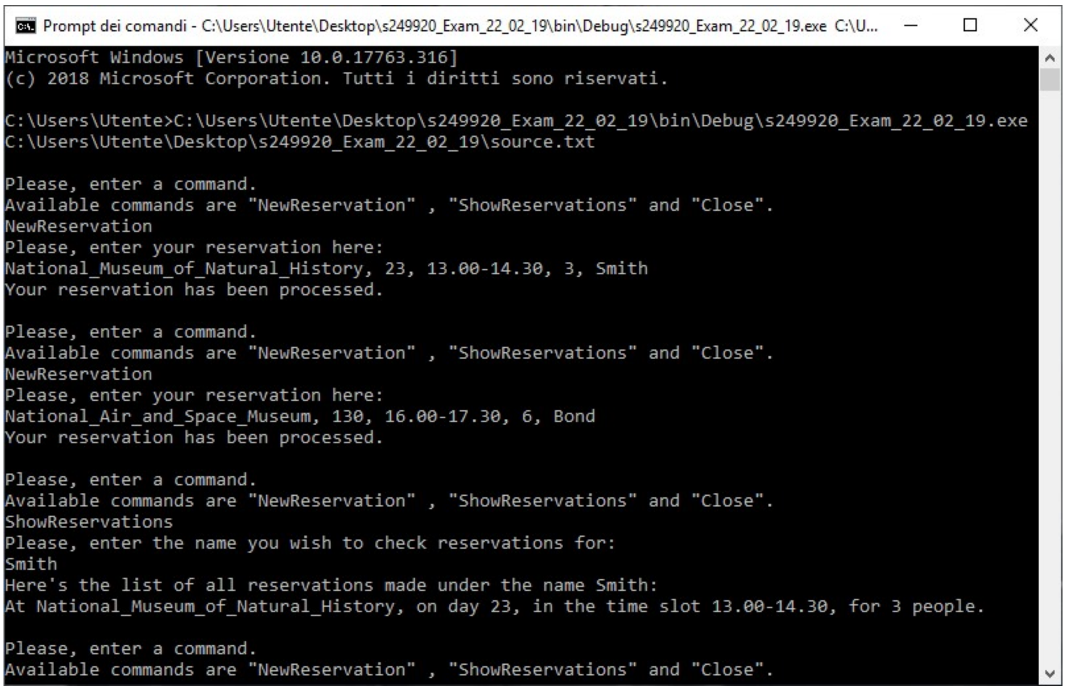
In case the user wants to add a new reservation, the program asks for the data needed, then formats the buffer for ease of data collection from *sscanf()*, by substituting all punctuation characters which are not an Underscore Character with Whitespace Characters. The data is then passed to the *add_reservation()* function, along with the museumsBST and the visitorsBST. The *add_reservation()* function checks if the data is malformed. If it is, a respective error is printed on stdout, returning 0. This is different from the written exam, as I felt it would have been easier to test the program if it didn't hard-close with *exit(EXIT_FAILURE)* every time it recieves malformed data. Functionally, the code is the same.
On the other hand, if the data is not malformed, and there are enough tickets left for the desired musem, day and time slot, the *insert_visitor()* function is called to search the correct spot in the visitorsBST and either add the new reservation to the Tree or enqueue the reservation to other previous reservations ma-de by the same person. *insert_visitor()* also updates the threshold of the given museum's day and time slot, by removing the placed tickets from threshold. Then, *add_reservation()* returns 1.

In case the user wants to find all previous reservations made by a particular person, the program asks for such a person's name, then calls the function *print_reservations()*. The function *print_reservations()* calls the *search_visitor()* function, searching the given person's name within the visitor names in the visitorsBST. If the person is found, all of his/hers reservations are printed from the reservations list implemented with *visitors_t* *next. Otherwise, if the person is not found, an error is printed on stdout.

Most of the sub-functions present in the program are library functions from "treePublic.h" that have been adapted to accomodate for the custom data structures have used. Within these functions there are:

- *search_museum()*, *search_visitor()*. Search in their respective BST for the desired data, returning the object's pointer if found or NULL if not.
- *insert_museum()*, *insert_timeslot()*, *insert_visitor_r()* (called by the wrapper function *insert_visitor())*. These functions, with different data structures, do the same thing, that is searching for the right spot in the BSTs and placing there the data given as function argument .
- *format_input()*. Goes through the buffer with the user's input and removes all puntuation characters except for Underscores. This makes the subsequent read from *sscanf()* easier. Underscores are not removed since they are part of the museums' names.
- *initialize_time_slots()*. Recursively allocates memory for, and enlists the list of "n" time slots for a museum at the time of adding a museum to the museumsBST. All the fields are initialized at -1, to later distinguish where to put the time slots read from the source file in the lines that follow the name of a museum. Returns the head pointer to the list, that is assiged to *time* *timeslotshead.
- *visitorsBST_dispose()*, *museumsBST_dispose()*, *museum_struct_dispose()*, *days_struct_dispose()*, *time_struct_dispose()*. These functions are used to free the memory that has been dynamically allocated by *load_system()* for the museumsBST and by *insert_visitor()* for the visitorsBST.

The whole code is commented line by line for ease of correction.
Here follows an example of the program's interface when run through Window's command prompt.


Prompt dei comandi - C:\Users\Utente\Desktop\s249920_Exam_22_02_19\bin\Debug\s249920_Exam_22_02_19.exe C:\U...

```
Microsoft Windows [Versione 10.0.17763.316]
(c) 2018 Microsoft Corporation. Tutti i diritti sono riservati.

C:\Users\Utente>C:\Users\Utente\Desktop\s249920_Exam_22_02_19\bin\Debug\s249920_Exam_22_02_19.exe
C:\Users\Utente\Desktop\s249920_Exam_22_02_19\source.txt

Please, enter a command.
Available commands are "NewReservation" , "ShowReservations" and "Close".
NewReservation
Please, enter your reservation here:
National_Museum_of_Natural_History, 23, 13.00-14.30, 3, Smith
Your reservation has been processed.

Please, enter a command.
Available commands are "NewReservation" , "ShowReservations" and "Close".
NewReservation
Please, enter your reservation here:
National_Air_and_Space_Museum, 130, 16.00-17.30, 6, Bond
Your reservation has been processed.

Please, enter a command.
Available commands are "NewReservation" , "ShowReservations" and "Close".
ShowReservations
Please, enter the name you wish to check reservations for:
Smith
Here's the list of all reservations made under the name Smith:
At National_Museum_of_Natural_History, on day 23, in the time slot 13.00-14.30, for 3 people.

Please, enter a command.
Available commands are "NewReservation" , "ShowReservations" and "Close".
```