Report

TL; DR (overview summary):

This program receives as argument a number of events, and randomly assigns each of them to a type –sorting books, plates and shopping robots. Then, according to each type of event, the program handles it at if it were a kind of "robots" doing household tasks, which in reality is reflected as either a list, a stack or a queue being modified in various ways. Finally, the program outputs a report on the current state of each list and its elements before cleaning each of them.

Description of the strategy

The main function receives as arguments the number of events as a list of characters (**argv) and an integer number which holds the number of strings passed as arguments (argc).

Then it calls the CheckArguments() function with those same parameters, which is the one responsible of managing the possible users' errors when introducing the arguments thanks to the atoi() function. It might return error messages if passed the wrong kind of arguments, but it might also be able to transform them to positive integers and work with them anyway in case they are negative or decimal. This function will also assign the number of events passes as input to the global variable EventNumbers.

Next, if the EventNumbers are not –1, which means the previous function worked correctly, the main calls the SimulationLoop() function.

This function calls the GenerateEventType() function as many times as events there are, which will assign randomly a type to each event, and depending on which type is assigned, it will call the appropriate functions.

- **Sorting books**

When the event is of type "book", a book is created, with a random author and its corresponding book title, and a random year. Then, SimulateSortingBooks() is called. This function will basically search the position where the generated book must be and place it after this position.

The function that provides the position where the new book must be is SearchBook(). This will traverse through the list until it founds a book with the same author, and will return the pointer of this book. In case there is no other book of the same author, the new book will be placed at the end of the list.

- **Managing plates**

If the event is of type "plate", a plate is created, of a random type (dinner, soup or dessert plate) and random color. Then, SimulateManagingPlate() is called. This function will basically add each plate into a different stack depending on its type but will empty a stack when it reaches maximum capacity. *

It will do that by calling the PushPlate() function, which will add each plate at the top of its corresponding stack considering if it is empty. Then, if any stack has reached maximum capacity, the function RemoveStack() will be called, and it will empty it.

* The stack will immediately be emptied whenever it reaches the maximum capacity, so it's not possible to see a stack with maximum capacity plates in PrintPlates().

- **Go for shopping**

When the event is of type "shopping", a shopping robot will be created, with a random number of thigs to buy and an ID +1 higher than the previous shopping robot. The robot will be added to the queue. If the queue is empty, it will directly enter the shop, if not, it will wait after the last robot of the queue, waiting for its turn.

When a robot enters the shop, the counter eventsToConsume will be set to the robot's number of things to buy. Whenever an event of the previous both (books or plates) is called, this counter will be decreased by 1. Once it reaches 0, the next robot of the queue will enter the shop, starting the process again. If there are no robots in the queue, the counter will still decrease to even if it's negative, this will not cause any problem to the previous process.

- **Cleaning the simulation**

Finally, when all events are processed, the SimulationLoop() function will clean the simulation by calling three functions: RemoveSortingBooks(), which empty the list of books; CleanPlateStacks(), which will call the function RemovePlates to empty the stacks which haven't reached maximum capacity; and the CleanShoppingQueue(), which will empty the queue of remaining books. Each of these functions will also indicate how many elements of each list have been removed.

Although they are not used, if the functions PrintBooks(), PrintPlates() and PrintShopping() are called, each of them will print the current state of their corresponding list, giving information about the indexes of each list member.

Criticism of the problems that arose

We found many bugs during the project, but here are some interesting ones:

- Before, plates were added to the stack after comparing if the stack was full, so there were times when there were stacks at max capacity during the execution of the program. This didn't cause an error, but it changed the final statistics respect to adding the plate before checking if the stack was full. Finally, we kept this last version (adding the plate and then checking the stack), as it was more intuitive and fit more with the instruction pdf.

- Another bug we had is that the global variable eventsToConsume was always decreasing, reaching negative numbers. We didn't realise that, so when checking if it was equal to 0 in function UpdateShopping(), this gave always False, so robots were

never dequeued. When we found the bug, we decided to just change this comparison from "==" to "<=", instead of limiting eventsToConsume to 0. This may seem counterintuitive, the fact that there can be negative events to consume, but in the code, it just works fine, and it's even more efficient than limiting it to 0 every time it decreases.

- We spent some time wandering why the final statistics for shopping were not the same as the ones in the instruction pdf. We were made aware that this could not be a bug, but the fact that randomness, even if there was seed, could give different results to different computers. Also, it could be that our implementation of the functions makes the result change, but still being right.

Conclusion

To conclude, we enjoyed the freedom the instructions of the project gave us regarding the content of the functions, compared to the rigidity imposed by similar projects in Python last semester.

We also managed to adapt correctly to the mechanics of lists, stacks and queues, in part thanks to the fact that we had similar instances of code to work with from previous classes and exercises, but we must admit they might get harder to apply in an exam, where we can't access these kinds of information sources.

The hardest thing we had to deal with was the final stage of the project, in which we had to find the "semantic" or "logic" errors in our code (the ones that didn't give a visible bug, but made the result be wrong), identify our previous mistakes and solve them.

We also found interesting the prospect of learning more about code style in C and the inner workings of "seeds" in the case, for instance, of the rand() function.