Project 1

Fundamentals of programming II

Gússem Yahia-Cheikh and Sara Díez

April 2025

First steps

First of all, before starting programming directly, we wanted to adress the whole problem and see what were the structure of creation to archive the desired result. So we started by making an schema/flowchart of the whole program and the steps we wanted to follow.

Strategy and Steps

We decide to divide the work by functionality provided each 'exercice' in the pdf, so the three main areas to work on are:

- Books
- Plates
- Shopping

Argument Management

So from there we started by the initial phase of the program, the input and the managment of it. The function `void CheckArguments (int argc, char *argv)` checks the number of arguments is correct and that the second argument $\in \mathbb{R}$. If the conditions are satisfied we stored the value in `EventNumbers`.

Event Loop Setup

Then we attacked the `void SimulationLoop(int EventNumbers)`, the structure we followed to code the function was:

1. Loop over the number of events
2. Generate a random event type (0,1 or 2)
3. On each iteration we delegate it corresponding type:
    1. Type 0: Package sortment
    2. Type 1: Package classification
    3. Type 2: Shopping
4. And finally simulating the time passing after each event (`UpdateShoppingQueue()`)

Task 1: Package managment

Now let's address the package managment block, here we will work with sorted linked lists ordered by `suppliers` (taking in count that the `RobotPackage` contains a `supplier`, `id` and `year`). We have three functions working in this block: `PrintRobotPackages()`, `SearchRobotPackage()` and `SimulateManagingRobotPackages()`.

Task 2: Package Classification (Stacks)

For this task, we implemented three stacks representing small, medium, and large packages. Each package has a randomly assigned type and color. When a package arrives, the system decides which stack it belongs to and pushes it there.

We defined a `Stack` structure using a singly linked list. Each time a package is classified, we check whether the stack has reached its `MAX_CAPACITY`. If it has, the robot simulates transporting all packages by removing (freeing) all nodes in the stack.

This allows us to manage space dynamically and simulate real-world capacity constraints.

Task 3: Shopping (Queue and Time)

This system manages a shopping queue. Each robot is created with a shopping list (a number of items to buy) and added to the queue.

Time is simulated through the event loop: one unit of time passes for each event (sorting, classification, or shopping). We track how many items each robot still has to buy. When a robot's count reaches zero, it is removed from the queue (and memory is freed).

This approach mimics how robots would queue up and shop asynchronously, with their progress depending on the total activity in the system.

Fine Tuning (Cleanup and Testing)

Problems and criticisms

One of the biggest challenges was managing dynamic memory correctly. It was easy to forget freeing certain structures, especially when dealing with stacks and queues.

Another difficulty was simulating the shopping time using event counts. We had to ensure that time progressed accurately and that no robot was skipped or served out of order.

Finally, handling the string operations and ordering in the package management system required careful use of `strcmp()` and array boundaries.

Output Sampling