CMPE 230 SPRING 2024 PROJECT 1 - FAVOR FOR THE RINGMASTER REPORT

Berk Göktaş 2022400039
Bora Aydemir 2021400054

April 2024

1 Purpose

Our main purpose was to implement the project with a simple and plain concept to maintain the code's reusability and reduce the effort needed while implementing and optimizing the code. We had a well-thought-out, structured, and organized design process to achieve this.

2 Design

As stated in the purpose part, the design concept was crucial for the project's maintainability. Before starting to code, we thoroughly thought about the kind of implementation that would make our work simpler. Most of the thought process was on how to implement parsing of the sentences. At first, we wanted to use a recursive function that parses both action sentences and condition sentences, however, we quickly realized that that would cause more problems as the structure of the action sentences and condition sentences are different, implementing a recursive code aiming to correctly parse two different structures together would be a challenging task. We then decided to implement a global reader that calls two different parse functions by determining the type of the next sentence using the return type of the parser function used before. This change in design increased our efficiency tremendously. In the design process, we also determined the functions that will operate on global variables by changing them and the functions that will only operate inside and globally return a value. We designed structs that will hold the components of a sentence, the properties of characters, the properties of items, and the properties of locations. After having a clear design process, we split the tasks and began implementing the project merging the changes we made on our code in a shared repository and effectively communicating with each other in key aspects that might affect the integrity of the project.

3 Implementation

At first, we split the tasks in two to reduce our workload. We implemented basic functions such as creating a location, creating a subject, assigning a subject to a location, etc. as if we correctly parsed all of the sentences. So the first part implemented was the sentence execution part. For the parsing part, we implemented two different parsing functions, one parser for condition sentences and one parser for action sentences that after parsing returns an integer which is used as an indicator to which parsing function the global reader function will call. For example, if an action sentence is being parsed and the next sentence is a condition sentence, the action parser would return the value 1 indicating the next sentence will be a condition sentence and hence will be parsed by the condition parser. When the value 2 is returned, the global reader function will stop as it indicates we are at the end of the sequence. To parse the sentences, we used a method that checks the position of the word relative to the keyword of the sentence to determine if the type of the word is valid. For all sentences, both parser functions will parse subjects first until they reach a keyword, after that, depending on the keyword, it will either parse items or a location. Since different sentences and items are both separated with and, there needs to be the keyword "and" in specific locations inside a sentence. This is being checked in the parser functions using the modulo operation which takes modulo 3 of the position of the word after the keyword relative to the keyword. This ensures the sentence is indeed valid, and if it's not allows us to detect the syntax error and terminate the process earlier. We also use this modulo method to check if an item is valid as the first component of an item should be an integer and the rest is the same with the subject and locations - checking if the string is valid. Before parsing the sentence, we create a sentence struct that will be used to fill the properties of the sentence while parsing. We do parsing and filling the properties simultaneously decreasing the time needed thus improving the efficiency. After creating the sentence structures, we store them in an array that will be used to execute the sentences. We traverse through the array locating the condition sentences, checking the validity of the condition sentences, and executing the action sentences until that condition sentence if it's valid. This ensures a wrong action will never be executed thus eliminating the possibility of taking an action executed back. The questions are handled separately with a separate question parser as the questions are never used with other sentence types together. If we were to give an example of one of the problems we faced in terms of implementation, it was the program running differently due to the differences in system OS. We got a full grade on a Mac on autograder, however, on my PC that uses Linux, we faced a segmentation fault. We fixed this using memory debugger Valgrind to detect memory leaks and faults. We realized that due to the different memory handling by different OS's, uninitialized variables were the cause of the problem. We fixed this problem by initializing all the variables we used beforehand.

4 Examples and How To Use The Program

After typing "make" into the console, the program will compile and the ringmaster executable will be created. After running the executable, the sentences will be fed from the console as input. If the sentence is valid, it will be executed, and "OK" will be printed to the console, otherwise no action will be taken, and "INVALID" will be printed. Type "exit" into the console to terminate the program. Here are some examples:

```
bayd@Boras-MacBook-Air-2 bouncmpe230-assignment1 % ./ringmaster
>> Garavel and Emrata and Ceku buy 100 Nenya and 100 Narya and 100 Vilya
OK
>> Bob and SydneySweeney buy 10 Vilya and 10 Narya from Ceku
OK
>> Ceku total ?
100 Nenya and 80 Narya and 80 Vilya
>> Bob total ?
10 Vilya and 10 Narya
>> Bob and Ceku sell 10 Nenya to Logar
OK
>> Logar total Nenya ?
0
>> Bob and Ceku sell 10 Vilya to Logar
OK
>> Logar total Vilya ?
20
>> Bob and Ceku total Narya ?
90
>> exit
bayd@Boras-MacBook-Air-2 bouncmpe230-assignment1 %
```

Figure 1: Example 1

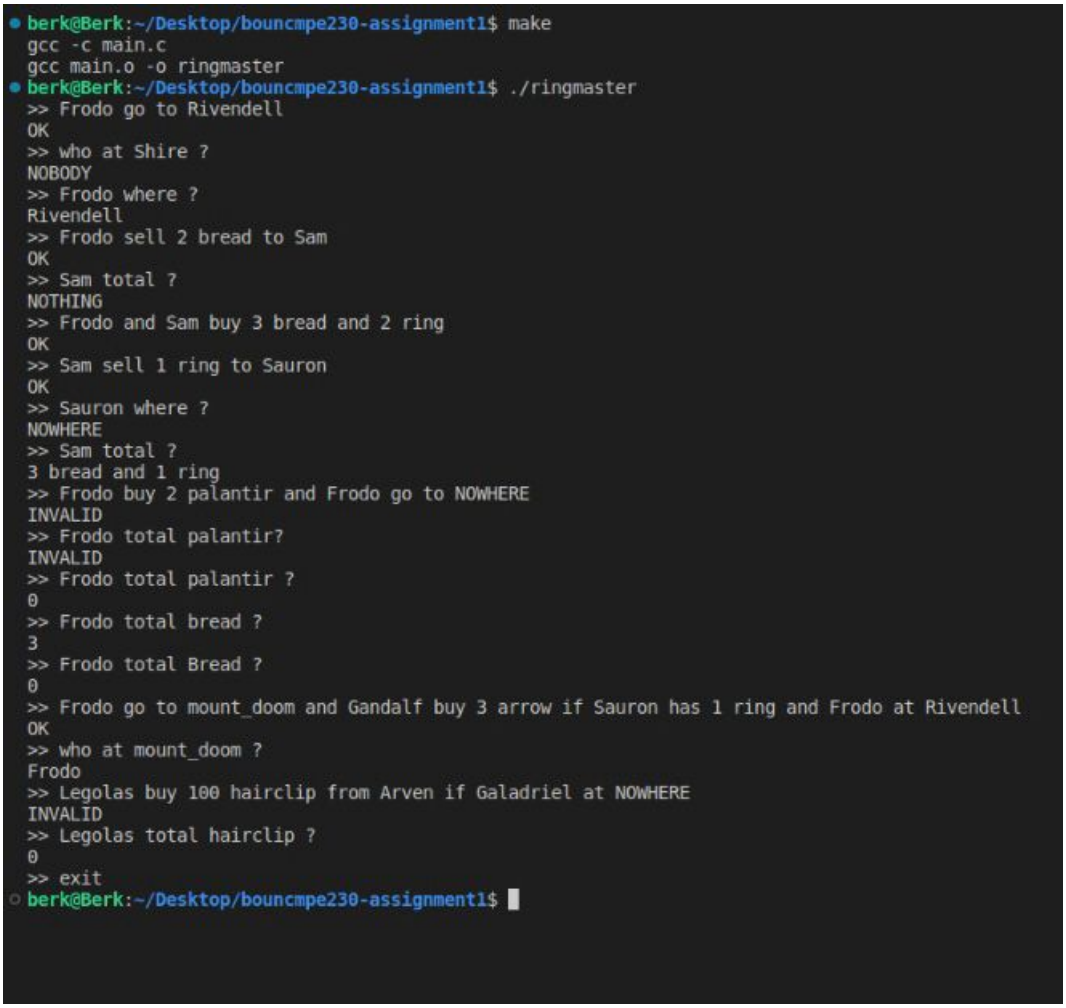

Figure 2: Example 2

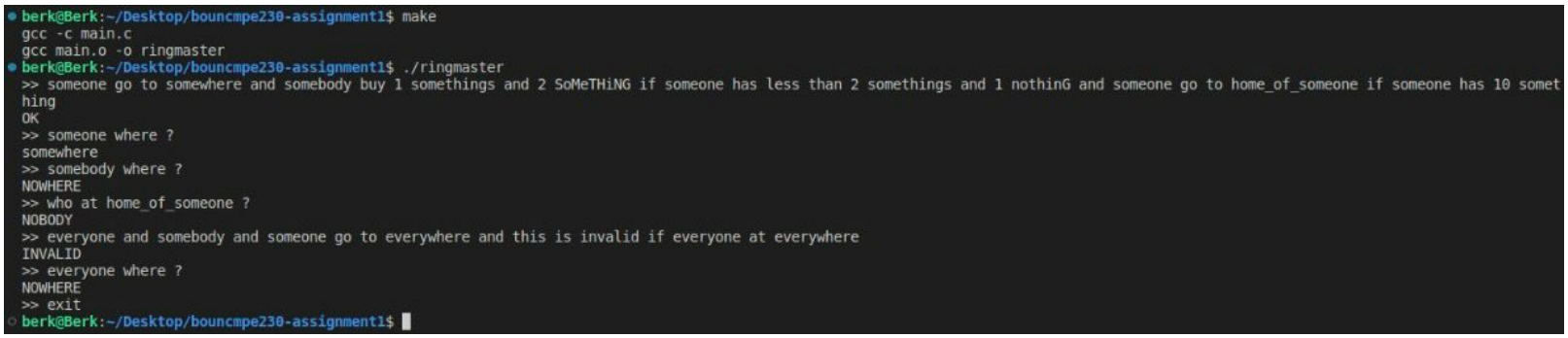

Figure 3: Example 3

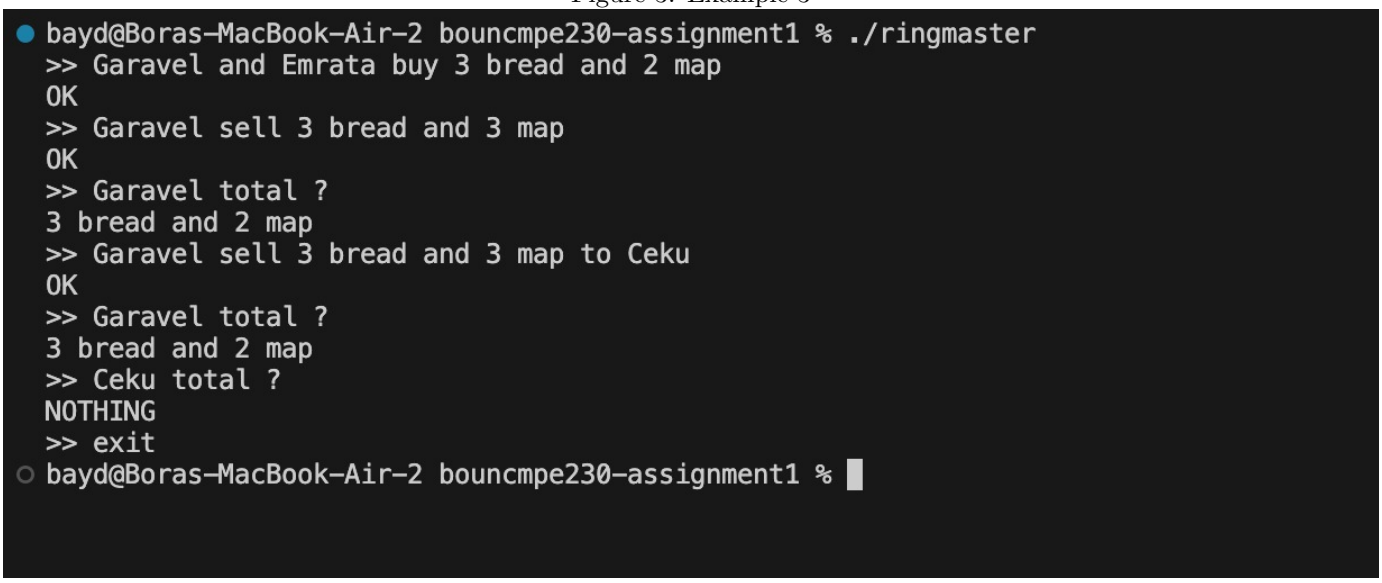

Figure 4: Example 4