Hotel Kashyyyk Evaluation

David Wheatley

# Meeting success criteria

Evaluating code is one of several key criteria that must be fulfilled to learn and improve. Our program has managed to fulfill all criteria required in the brief. This would not have been possible without a robust test table.

## Check in

See the table below for the requirements for the check in process, if they were met, how they were met and could be improved in future.

|  |  |
| --- | --- |
| **Stores surname and date of birth** | **Fully met** |
| Surname is stored correctly (max 63 chars), and DOB is stored and validated (not less than 1900, not more than 2020, plus valid day/month combinations, including leap year checks to allow for 29 Feb on leap years. We have, however, changed from using DD/MM/YY to YYYY-MM-DD. This is a more standardized date format. | |
| **Stores number of guests, incl. adults and children** | **Fully met** |
| The number of guests is stored correctly with each booking. We first ask for the total guest count, then ask for how many of those guests are 16 or under. These values are stored separately to then be used to provide discounts for the price of meals. | |
| **Stores board type (full, half, or B&B)** | **Fully met** |
| We used 1, 2, and 3 to represent the possible board types. These were hash-defined to prevent confusion across the codebase. | |
| **Stores length of stay in days** | **Fully met** |
| Our program validates that the user has entered valid number ( and correct type). It stores this under the same index as the Booking ID. | |
| **Stores if guest wants wake-up calls** | **Fully met** |
| Our program asks with a Y/N prompt if the guest wants daily wake-up calls. It repeats the prompt if they enter an invalid option. Their choice is then saved. | |

|  |  |
| --- | --- |
| **Shows rooms available for booking** | **Fully met** |
| When a room is booked, an array called BookedRooms is updated. The entry at the index of the room number is updated to show that it is unavailable (i.e. for Room 6, index 6 would be set to true). This array is checked during booking – the program will not display any room where this is set to true. The check in process also prevents booking one of these rooms if it is not displayed to the user as free. | |
| **Assigning a booking ID** | **Fully met (with issue)** |
| To generate a booking ID, the system takes the guest’s surname, then adds two random numbers to the end.  This system could be significantly improved as, at present, we don’t check if duplicate booking IDs exist, meaning, if two people with the same surname checked in, and happened to be assigned the same Booking ID, they would collide when attempting to book tables. | |

## Book a dinner table

See the table below for the requirements for the dinner table booking process, if they were met, how they were met and could be improved in future.

|  |  |
| --- | --- |
| **Input guest booking ID** | **Fully met** |
| We get the guest’s booking ID and iterate through the list of booking IDs until we find the ID they inputted, then save the index it was found at. All the other data about the booking can also be found at this index.  If the ID can’t be found, we let the user know and go back to the main menu. | |
| **Verify eligibility for dinner table (FB/HB only)** | **Fully met** |
| We check, using the index fetched in the previous step, the board type to make sure it is either Full or Half board. If it is not, we show an error message, then go back to the main menu. | |
| **Check free table count, and print available times/tables** | **Fully met** |
| We go through the current table bookings to check if any are free. We know how many guests are in the booking and assume they all want to eat together. If there is not enough space for them, we tell them. If there is, we display that time as an option (7 PM, 9 PM, or both).  The user can choose the time that they want, then we display the tables which are free, which they can select. | |

|  |  |
| --- | --- |
| **Ask user which table they would like to book** | **Fully met** |
| The user can choose between the 3 available tables. If they have more people than can fit on one table, we loop this part so that they can book multiple tables for their party. We show a “guests remaining to seat” counter, too, so they know how many people they have left to reserve a table for.  We could improve this by providing a way to override this counter so they can book a table for an amount less than what is in their booking. | |
| **Show a message confirming their table booking** | **Fully met** |
| After the table booking is complete, we confirm their booking(s) with a message, along with confirmation of the table name and time. | |
| **Update the tables available for that time** | **Fully met** |
| After the table is booked, we update the array which determines if the tables are available to be equal to their booking ID (for the table they reserved). | |

## Check out

See the table below for the requirements for the check out process, if they were met, how they were met and could be improved in future.

|  |  |
| --- | --- |
| **Input guest booking ID** | **Fully met** |
| We get the guest’s booking ID and iterate through the list of booking IDs until we find the ID they inputted, then save the index it was found at. All the other data about the booking can also be found at this index.  If the ID can’t be found, we let the user know and go back to the main menu. | |
| **Calculates price of booking** | **Fully met** |
| Our program correctly calculates the cost of the guest’s booking and displays it to them. | |
| **Calculate subtotals** | **Fully met** |
| Our program shows the intermediary costs and subtotals (for the room, board, wake-up calls) and displays these. | |
| **10% off room cost if 65+** | **Fully met** |
| We use the DOB entered at check in to determine if they are 65+. If they are, we provide a 10% discount on the room rate only. This discount is displayed under the combined subtotal, but before the final total. | |
| **Displays booking ID in bill** | **Fully met** |
| We display the booking ID at the start of the receipt. | |
| **Clear booked rooms after check out** | **Fully met** |
| We iterate through the rooms in the booking and set their “booked” status to false in the RoomsBooked variable which is checked against during check in. | |

## Quit program

See the table below for the requirements for the quit program process, if they were met, how they were met and could be improved in future.

|  |  |
| --- | --- |
| **Returns to menu – doesn’t actually exit** | **Fully met** |
| Currently this option is only displayed on the main menu, so having it loop back to the main menu is almost useless. We weren’t sure where else to put it. | |

# Evaluating my code

Our code is robust. Combining our team’s different knowledge created an initial prototype program of each section which I then merged. The initial prototype code was quite easy to attack and cause crashes, so I slowly added verification checks.

While adding these verifications, I found several other oddities which had to be worked around, meaning that my final code ended with ‘hack’ on top of ‘hack’, with several individual layers of ‘hack’, along with a lovely side dish of ‘hack’. Reflecting on this, I should have done more research into C input validation so that I did not use 3+ different forms of it in the program.

From my research after the fact, it seems to be a consensus that the best way to perform input validation is using fgets() and combining this with sscanf\_s() to validate data types. I will try to use this in the future in situations where a program not crashing is critical. For some programs, a simple scanf() is still better: there needs to be a balance between complexity and time.

The trickiest part of this project for me was merging the code. If my team members were more confident, I would have preferred to use a form of version control, such as git, backed up by a remote, such as GitHub, to manage our code, however this wasn’t a realistic option for us. I had created a data dictionary file, explaining the variables we’d use and their data types, however my team seemed to misunderstand this and converted the 2D arrays into 1D arrays, or 1D arrays into “standard” variables. This frustrated me, but I should take most of the blame. If they did not understand it, it indicates my documentation was not clear enough, but they should still ask me.

Eventually I renamed the variables they had used and, when the function is called by the menu subroutine, these variables get populated from the correct index of the global variables I had made. The check out functionality was easiest to almost fully rewrite than convert to the new method, so I ended up doing this, too.

During this project, I didn’t plan as much as I should, resulting in strange bugs and long debugging sessions. If I had written more pseudocode, or made more flowcharts, these issues would have been less likely to arise.

## Targets

1. Perform more research before writing code.
2. Use a version control system to make merging our different code more easily.
3. Plan more!

# Evaluating our team

Initially I had some concerns as my team was not communicating well, however everything worked out fine in the end. I decided not to start writing my code until after I received my team’s code. My section, table booking, seemed to be the most difficult at the start, but, with hindsight, I think that the other sections were more complex, not in concept, but in execution: validating input; performing seemingly simple calculations (65+ discount); etc.

The trickiest thing was the way my team initially provided their code. The check in function was completed first, then the check out feature was added on top of that, rather than a separate file or function. This made the merging stage more difficult as I could not just add a call to a subroutine and a new menu item. Because of this, I decided to refactor the check in and check out code into separate files with their own function calls. This made it much easier to merge overall.

Deadlines were an issue internally within our team. As I had postponed writing my code without informing my team, when they handed their code over to me (at the last moment), I hardly had enough time to merge and refactor their code, write my own, test, fix bugs, and finalise a release version. Furthermore, one team member did not know how to complete one part of the brief, so I did this for them, too.

## Targets

1. Regularly contact my team members to check their progress.
2. Ensure there are no miscommunications or misunderstandings within my team.
3. Rather than implementing a feature myself, try to explain the concept and how I would write it to my team member(s) so they have a learning experience.