📚 COP 4338 Assignment 3: Restaurant Management System - Testing Guide

✅ Execution Permissions Notice

If you **do not** have execute permissions for instructor-provided framework files, you can manually assign the necessary permissions to specific files or to all files within a directory using the following command on a **Linux system**:

> **Note:** When files are uploaded or transferred to Ocelot server (`ocelot-bbhatkal.aul.fiu.edu`), execute permissions **may be stripped** due to security restrictions. In such cases, you must explicitly grant **read (r)**, **write (w)**, and **execute (x)** permissions.

Granting Permissions to All Items in the Current Directory

```
# Check current permissions
ls -l

# Grant read, write, and execute permissions (safe and preferred - applies only what's missing)
chmod +rwx *

# Assign full permissions (read, write, execute) to all users for all files in the current directory
# ⚠️ Not preferred - use only as a last resort!
chmod 777 *

# Verify that permissions were applied
ls -l
```

🔄 How to Use the Provided Test Cases - *for manual tesing*:

This section outlines the recommended process for manually validating your implementation against sample test cases.

1. Simple Test Case

1. ***Copy the contents of*** `testcases_simple.txt` ***into*** `TESTCASES.txt`.
2. ***Run the instructor's sample executable to generate the expected output***:

   ⚠️ The file `TESTCASES.txt` must be present in the directory where `A3_sample` is located.

   💡 The file `TESTCASES.txt` should contain the test cases you wish to run to generate the `EXPECTED_OUTPUT.txt`.

   ```
   ./A3_sample > EXPECTED_OUTPUT.txt
   ```

3. ***Run your implementation (compiled via the provided*** `Makefile`***) to generate your output:***

   ```
   ./restaurant > STUDENT_OUTPUT.txt
   ```

4. ***Compare your output with the expected output:***

```
diff STUDENT_OUTPUT.txt EXPECTED_OUTPUT.txt
```

✅ **No differences means a 100% match and your implementation is correct. Congratulations!**

2. Moderate and Rigorous Test Cases

- Repeat the steps above using the corresponding testcases:
  - Replace the contents of `TESTCASES.txt` with the contents of `testcase_moderate.txt` and `testcase_rigorous.txt`, respectively.
  - Regenerate the corresponding `EXPECTED_OUTPUT.txt` using `./A3_sample > EXPECTED_OUTPUT.txt`.
  - Rerun your implementation and perform the comparison: `diff STUDENT_OUTPUT.txt EXPECTED_OUTPUT.txt`.

⚠️ **Important:** The test cases provided (simple, moderate, rigorous) are designed to help verify the functional correctness of your solution. However, **instructor will use additional complex and comprehensive test cases** during grading. Therefore, it is **mandatory** that your implementation passes all three provided test cases—simple, moderate, and rigorous — **to maximize the likelihood that it will also pass the instructor's test cases during final grading**.

🤖 How to use the autograders

The autograder scripts is available to facilitate automated testing throughout your development process **at any stage**. A correct implementation will earn **90 out of 100 points** through the autograders. The remaining **10 points** will be awarded based on:

- Adherence to submission guidelines
- Code structure and quality
- Code documentation

*** Please note that instructor will use the same autograders for the final grading.**

1. Required Files and Directory Structure

Ensure that the following files are located in the same directory:

```
Assignment_3/
├── restaurant.h                                    # Header file (DO NOT MODIFY)
├── driver.c                                        # Main program (DO NOT MODIFY)
├── Makefile                                    # Build system (DO NOT MODIFY)
├── autograder_restaurant_management_system.sh  # Autograder
├── batchgrader_restaurant_management_system.sh # Batch autograder
├── TESTCASES.txt                               # Test input data
├── EXPECTED_OUTPUT.txt                         # Expected output
├── copyString.c                                # Your implementation file
├── findMenuItem.c                              # Your implementation file
├── calculateSum.c                              # Your implementation file
├── addMenuItem.c                               # Your implementation file
├── processOrder.c                              # Your implementation file
├── sortMenu.c                                  # Your implementation file
└── Harry_Potter.ZIP                            # Your final submission ZIP folder
```

2. Executing the Autograder

Run the autograder script using the following command:

```
# Run autograder
./autograder_restaurant_management_system.sh
```

This script will compile your code, run the test cases, and provide a detailed score report.

3. Executing the Batch Autograder

⚠ **Remember:** Your final submission **ZIP folder** must be in your current working directory to run the batch autograder.

Run the batch autograder script using the following command:

```
# Run batch autograder
./batchgrader_restaurant_management_system.sh
```

This script will compile your code, run the test case, provide a detailed score report, and **flags error for any missing required files**.