Ahram Canadian University

Restaurant Management system

Basmalla Mohamed 42410550

Mawada Moustafa 42410098

Lojain Tamer 42410055

Malak Mohamed 42410066

Yasmeen Emad 42410094

Haneen Mohamed 42410388

Habiba Mohamed 42410389

Haneen Tamer 42410580

This code starts by defining a constant ***maxOrder*** to limit the maximum number of orders to 100. The ***orders*** array is initialized with five food items: pasta, fries, corndog, pizza and burger. The variable ***orderSize*** is set to 5, indicating the current number of orders. The function ***displayOrders*** checks if there are any orders to display. If there are none, it informs the user with a message saying "No orders to display!" If there are orders, it lists them out numbering each item.

The ***addOrder*** function allows users to add food items to their order list, which initially contains five items. It first checks if the order list is full. if so, it displays a message and exits. If there is space, it shows the current orders and prompts the user to select an item by entering a number. Based on the input, the corresponding food item is added to the order list, and the total order count is updated then a confirmation message is displayed.

The *deleteOrder* function takes an array of orders and the size of the array as arguments. It first checks if there are any orders to delete; if not, it informs the user and exits. The function displays all current orders and prompts the user to enter the index of the order to delete. It validates the user input to ensure the index is within the correct range. If valid, it shifts all orders after the specified index one position left to overwrite the .deleted order, reducing the size of the array. Finally, it confirms deletion

The *editOrder* function in C++ is used to modify an order in a list. It first checks if the list is empty, displaying a message and exiting if no orders exist. If the list is not empty, it uses the *displayOrders* function to show all current orders, then prompts the user to select an order to edit by entering its index. The function validates the index to ensure it falls within the correct range. If the input is valid, the user is asked to enter the updated order. The use of *getline()* ensures proper handling of multi-word inputs, preventing input errors. The specified order is then updated with the new value, and a success message is shown.

The *searchOrders* function is used to find a specific order in a list. It first checks if the list is empty and stops if there are no orders. The user is asked to type the name of the order they want to find. The function uses *cin.ignore()* and *getline()* to handle names with multiple words. It then goes through the list one by one to check for a match. If the order is found, it shows the position and name of the order and stops the search. If no match is found, it tells the user the order was not found.

The *calculateSales* function calculates and displays sales details for a list of orders. It takes the list of orders, their prices, quantities, and the total number of orders as inputs. The function initializes *totalSales* to 0.0. A loop goes through each order, calculates the sales for that order by multiplying its price with its quantity, and adds it to *totalSales*. For each order, it prints the name, quantity sold, and sales amount. After the loop, it displays the total sales for all orders.

The deleteAllOrders function removes all orders from a list. It takes the array of orders and the size of the list as inputs. A loop runs through the list, setting each order to an empty string ("") to clear it. After clearing the array, the size of

the list is set to 0, indicating that no orders remain. A message is displayed to confirm that all orders have been successfully deleted.

The *displayOrderFrequency* function counts and displays how often each type of order appears in a list. It initializes counters (*pizzaCount, friesCount, burgerCount, pastaCount, corndogCount*) to zero. A loop goes through all orders, checking each one against the predefined order types using *if-else* statements. When a match is found, the corresponding counter is increased. After completing the loop, the function prints the frequency of each order type, showing how many times each order appears in the list. This function provides a summary of order trends.

The *menu* function displays a user-friendly menu for a restaurant order management system. It prints a list of numbered options, each .representing a specific action

Add an order. .1
Delete an order. .2
Edit an order. .3
Search for an order. .4
Calculate total sales. .5
Display all orders. .6
Display the frequency of each order. .7
Delete all orders. .8
Exit the system. .9

At the end, the function prompts the user to input their choice by displaying `"Enter your choice:"`.

In the `main` function, the program sets up the essential components for managing restaurant orders. The number of dishes available is defined as 5, and an array of dish names is created to represent the menu items, including "Pasta," "Fries," "Corndog," "Pizza," and "Burger." Each dish has a corresponding price stored in a separate array, with values like 30, 20.50, 45, 89.99, and 75 for the respective dishes. An array of quantities is also initialized, with all values set to zero to indicate no sales have been made initially. These arrays allow the program to track menu items, their prices, and quantities sold, which are used for operations like calculating sales and updating the order list.

In the `main` function, the program offers a variety of actions based on user input, handled by a `switch` statement.

Case 1 (Add Order): This case calls the *addOrder* function, which allows the user to add an order to the list of orders.
Case 2 (Delete Order): Here, the *deleteOrder* function is called, enabling the user to delete a specific order from the list.
Case 3 (Edit Order): The *editOrder* function is triggered, allowing the user to modify an existing order.
Case 4 (Search Order): This calls the *searchOrders* function, which helps the user search for a specific order in the list.

Case 5 (Calculate Sales): The user is prompted to input the quantities of sold dishes for each item on the menu. The *calculateSales* function then computes the total sales based on the prices and quantities.
Case 6 (Display Orders): This calls the *displayOrders* function to show all current orders.
Case 7 (Display Order Frequency): The *displayOrderFrequency* function is triggered to display how many times each dish has been ordered.
Case 8 (Delete All Orders): The *deleteAllOrders* function is used to remove all orders from the list, clearing the system.
Case 9 (Exit): This option exits the program, displaying a message that the system is closing.

If the user enters an invalid choice, the program displays an error message and prompts them to try again. The loop continues running until the exit option (*case 9*) is selected.