IUBAT— International University of Business Agriculture and Technology

Founded 1991 by Md Alimullah Miyan

Project Documentation

CSC – 284

Submitted To
Abdullah Mohammad Sakib
Lecturer
Department of CSE

Team	
Team Name	Anonymous
Total Member	1
Project Name	IUBAT Super Shop Billing System
Technology	C++

Member	
Name	Md. Wasif Hossain
ID	23303320
Department	BCSE
Section	G

Introduction and Overview

The **IUBAT Super Shop Billing System** is a comprehensive software solution designed to streamline the billing process in retail settings, specifically for a super shop. Developed using C++, the system automates common billing tasks, such as adding items to the cart, calculating totals, and applying discounts. By incorporating fundamental Object-Oriented Programming (OOP) principles, this project showcases the potential of C++ to solve real-world problems efficiently.

Problem Statement

In retail operations, managing transactions manually can lead to errors, time consumption, and inefficiencies. The **IUBAT Super Shop Billing System** aims to address these issues by automating the billing process, ensuring faster and more accurate calculations, and simplifying inventory management. This system allows for quick generation of bills, offering features such as item addition, price updates, and discounts—all with minimal human intervention.

Purpose

The main goal of this project is to create a user-friendly and scalable billing system that can be integrated into any super shop or small retail business. By providing essential functionalities like discount calculations, itemized bill generation, and data storage, the system enhances the customer experience and improves operational efficiency.

Key Features

- **Item Addition**: Allows users to add multiple items to the shopping cart by specifying item name, price, and quantity.
- **Discount Calculation**: Users can apply a discount on the total bill, with automatic calculation of the discounted amount.
- **Bill Generation**: Displays a detailed bill, including individual item costs, total price, and final total after discount.
- **File Handling**: Saves transaction data to a file, ensuring that the system retains all item details and purchase history.

Target Audience

This system is aimed at retail businesses, small supermarkets, and shops that want to automate their billing operations. It can easily be customized to accommodate specific business needs, such as adding tax calculations, loyalty points, or integration with barcode scanners.

Benefits

The project provides a seamless and efficient solution to automate billing operations, reducing errors and saving time. It is designed with scalability in mind, so future

enhancements—such as integrating customer accounts or incorporating advanced analytics—can be easily added. Additionally, the intuitive interface ensures that users, even with minimal technical expertise, can easily navigate and use the system.

🗁 Structure of the Code

The **IUBAT Super Shop Billing System** is designed following a modular approach, adhering to Object-Oriented Programming (OOP) principles. The structure is intended to keep the code clean, maintainable, and easy to scale. The system has been divided into different components—classes, functions, and file handling—to organize the project effectively.

1. Library Usage

a. #include <iostream>

- **Purpose**: The <iostream> library is used for input and output operations in C++. It provides the necessary functionalities for interacting with the user via the console.
- **Usage in the Project**:
    - **Output**: The std::cout object from this library is used extensively in your project to display messages to the user, such as prompting for inputs, displaying the total bill, and showing success or error messages.
        - Example: std::cout << "Item 'Laptop' added to your shopping list." << std::endl;
    - **Input**: The std::cin object is used to read user input from the console. This is critical for receiving the item details (name, price, quantity) and the discount percentage.
        - Example: std::cin >> itemName; for inputting the item name.

b. #include <fstream>

- **Purpose**: The <fstream> library is used for file handling operations, allowing the program to read from and write to files.
- **Usage in the Project**:
-
    - **File Writing**: The std::ofstream class is used to write item details (name, price, quantity) to the totalbill.txt file as the user adds items to their cart. This allows the system to store transaction data persistently.
        - Example:

          cpp
          Copy code
          std::ofstream outFile("totalbill.txt", std::ios::app);
          outFile << itemName << ": " << itemPrice << " x " << itemQuantity << " = " << itemTotalCost << std::endl;
          outFile.close()

- **File Reading**: The std::ifstream class can be used (if necessary) for reading previously saved item details from the file. In your current project, it may not be directly used to retrieve data, but it could be useful if you want to load previous bills or data.
    - Example (if implemented):

      cpp
      Copy code
      std::ifstream inFile("totalbill.txt");
      while (getline(inFile, line)) {
      // Process each line
      }
      inFile.close();

c . #include <sstream>

- **Purpose**: The <sstream> library provides functionalities to manipulate strings and perform input/output operations on strings. It is commonly used for converting data types or extracting specific parts from strings.
- **Usage in the Project**:
    - **String Manipulation**: The std::stringstream class can be used to parse or combine strings. For example, if you need to format a string by combining different types of data (such as item names and prices), std::stringstream allows you to easily convert values into strings.
        - Example:

          cpp
          Copy code
          std::stringstream itemDetails;
          itemDetails << itemName << ": " << itemPrice << " x " << itemQuantity;
          std::string formattedItemDetails = itemDetails.str();
          std::cout << formattedItemDetails << std::endl;

    - **Input Parsing**: It could be used for parsing user input if you expect multi-part input that needs to be split, such as entering item details in one line (name, price, quantity), then breaking it down.

d. #include <string>

- **Purpose**: The <string> library provides functionality for working with strings in C++. It defines the std::string class, which allows you to work with dynamically-sized strinng
- **Usage in the Project**:
    - **Storing Item Information**: The project heavily relies on std::string for storing item names and other text-based data (like the discount type or customer input). It ensures that string data is handled safely and efficiently.
        - Example:

```
cpp
Copy code
std::string itemName;
std::cout << "Enter item name: ";
std::cin >> itemName;
```

- **String Validation**: You can use std::string to validate inputs, check if strings are empty, or modify the content (e.g., trimming spaces or appending additional information).

2. Classes

The project makes use of two main classes to represent the core components of the billing system:

Bill Class:

- The **Bill** class serves as the base class for the billing system. It is responsible for handling basic operations such as storing item information (item name, price, and quantity) and calculating the total price of all items in the shopping cart.
- **Attributes**:
  - itemName: Stores the name of the item.
  - itemPrice: Stores the price of the item.
  - itemQuantity: Stores the quantity of the item purchased.
- **Methods**:
  - addItem(): Adds a single item to the cart, updating the item details and the total cost.
  - calculateTotalCost(): Computes the total cost of all items in the cart before applying any discounts.
  - getItemDetails(): Retrieves the item details (name, price, and quantity) for display.

DiscountBill Class:

- The **DiscountBill** class is derived from the **Bill** class. It extends the functionality of the base class to incorporate discount calculations.
- **Attributes**:
  - discountPercentage: Represents the percentage of discount to be applied to the total bill.
- **Methods**:
  - calculateDiscountedTotal(): This method overrides the calculateTotalCost() method from the **Bill** class. It applies the discount to the total cost and returns the final amount after the discount has been subtracted.
  - applyDiscount(): Sets the discount percentage for the current transaction.

3. Functions

The project utilizes several functions to handle the operations related to item addition, bill generation, and user interaction. Here is a breakdown of the important functions:

addItemToCart(Bill&):

- **Purpose**: Adds an item to the customer's shopping cart.
- **Parameters**: Takes a reference to a **Bill** object to add the item to the cart.
- **Functionality**:
    - Prompts the user for the item name, price, and quantity.
    - Validates the user input to ensure proper data entry (e.g., ensures the price and quantity are numeric).
    - Adds the item to the **Bill** object by calling the addItem() method.
    - Updates the total cost by invoking calculateTotalCost() from the **Bill** class.

printTotalBill():

- **Purpose**: Displays the total bill, including itemized details and the discount calculation (if applicable).
- **Parameters**: None.
- **Functionality**:
    - Loops through the items in the cart and prints each item's details (name, price, quantity, and total cost).
    - Calculates the total before and after applying the discount (if any).
    - Displays the final bill, including:
        - List of all items and their individual prices.
        - Total price before the discount.
        - Total price after the discount (if applicable).
        - Final discounted total.
    - The function ensures a neat and organized display of the bill summary, enhancing user experience.

main():

- **Purpose**: Controls the flow of the program and provides the user interface.
- **Parameters**: None.
- **Functionality**:
    - Displays a menu to the user with options to add items to the cart, view the total bill, or exit the program.
    - Calls **addItemToCart()** to handle item additions and **printTotalBill()** to display the final bill.
    - Manages user input, ensuring the correct selection of menu options and prompting for re-entry in case of invalid inputs.
    - Handles program termination by clearing any stored data in totalbill.txt and displaying a thank-you message.

4. File Handling

To ensure smooth data management, the project incorporates basic file handling for saving and retrieving transaction details.

totalbill.txt:

- **Purpose**: Stores the item details (name, price, quantity, and total cost) of each transaction.
- **Functionality**:
  - As items are added to the cart, their details are saved to the totalbill.txt file.
  - This allows the system to keep track of all purchased items and their total cost for future reference.
  - When the user exits the program, the file is deleted to clear all stored data, ensuring the system remains clean for the next transaction.

The use of file I/O ensures that the system's data remains persistent, even after the program terminates. However, this approach can be expanded to use more advanced data storage methods in future versions, such as databases or cloud-based storage, if needed.

OOP Concepts Used

This project utilizes several fundamental **OOP principles**:

1. Encapsulation:

- Private and protected attributes like itemName, itemPrice, and itemQuantity in the Bill class ensure data security.
- Getter and setter methods provide controlled access to these attributes.

2. Inheritance:

- The DiscountBill class inherits from the Bill class, enabling code reuse and extension of functionality.

3. Polymorphism:

- The calculateTotalCost() method is overridden in the DiscountBill class to include discount logic.

4. Abstraction:

- High-level methods like addItemToCart() and printTotalBill() abstract complex logic, making the code cleaner and easier to maintain.

💡 Benefits of This Project

1. **Efficiency:**
   - Automates the billing process, reducing human error and saving time.
2. **Flexibility:**
   - Allows users to add multiple items and apply customizable discounts.
3. **Scalability:**
   - Can be easily extended to include features like taxes, loyalty points, or barcode scanning.
4. **User-Friendly:**
   - Simple menu-driven interface ensures ease of use.

Workflow of the IUBAT Super Shop Billing System

The **IUBAT Super Shop Billing System** is designed to automate the billing process for a super shop or retail business. It performs several key operations such as adding items to the cart, calculating the total price, applying discounts, and generating a final bill. The system is structured in a way that it guides the user through a simple menu interface and provides output in the form of clear, readable bills.

The workflow of this system can be broken down into several stages, with specific interactions between the user and the program. Each stage involves calls to functions, class methods, and user input, followed by appropriate outputs. Below is a step-by-step walkthrough of how the code works, from start to finish.

Step 1: Program Initialization

When the program is run, the first thing that happens is the execution of the **main()** function. The program will display a menu to the user, allowing them to choose between various options, such as adding items to the cart, viewing the total bill, or exiting the program.

```
std::cout << "Welcome to IUBAT Super Shop Billing System!" << std::endl;
std::cout << "1. Start Shopping" << std::endl;
std::cout << "2. View Total Bill" << std::endl;
std::cout << "3. Exit Program" << std::endl;
std::cout << "Please select an option: ";
```

Step 2: User Input Handling

The program will wait for the user to input a choice. This is done via the **std::cin** object, which reads the user's input. Depending on the choice, the program will either allow the user to start shopping, view the total bill, or exit the program.

```
int choice;
std::cin >> choice;
```

Step 3: Start Shopping - Adding Items to Cart

If the user selects the option to **Start Shopping** (choice 1), the program will proceed to prompt the user for item details. The user will be asked to input the **item name**, **price**, and **quantity** for each item they wish to add to their shopping cart.

a. Item Details Input

```
std::string itemName;
double itemPrice;
int itemQuantity;
std::cout << "Enter item name: ";
std::cin >> itemName;
std::cout << "Enter item price: ";
std::cin >> itemPrice;
std::cout << "Enter item quantity: ";
std::cin >> itemQuantity;
```

b. Adding the Item to the Cart

Once the item details are collected, the **addItemToCart()** function is called, passing a reference to a **Bill** object. The item is added to the cart, and the total cost is updated accordingly. This process is handled in the addItem() method of the Bill class.

```
Bill bill;  // Instance of Bill class
bill.addItem(itemName, itemPrice, itemQuantity);
```

The addItem() method adds the item to the cart and updates the total cost of the shopping cart. The updated total cost is stored in the **totalCost** attribute of the Bill object.

```
void Bill::addItem(std::string name, double price, int quantity) {
    itemName = name;
    itemPrice = price;
    itemQuantity = quantity;
    totalCost += price * quantity;
}
```

c. Saving Item Details to File

After adding the item, the item details are saved to a file, totalbill.txt, using **std::ofstream** for file output. This ensures that all transaction data is stored persistently.

```
std::ofstream outFile("totalbill.txt", std::ios::app);
outFile << itemName << ": " << itemPrice << " x " << itemQuantity << " = " <<
itemPrice * itemQuantity << std::endl;
outFile.close();
```

The user will then see a message confirming the item has been added:

```
std::cout << "Item '" << itemName << "' added to your shopping list." << std::endl;
```

Step 4: View Total Bill

If the user selects the **View Total Bill** option (choice 2), the program will call the **printTotalBill()** function. This function performs several key tasks:

a. Displaying Item Details

First, it loops through the cart's items and displays the details of each item, including the item name, price, quantity, and total cost.

```
std::cout << "Items in your cart:" << std::endl;
for (const auto& item : items) {
    std::cout << item.name << ": " << item.price << " x " << item.quantity << " = " <<
item.price * item.quantity << std::endl;
}
```

b. Calculating the Total Cost

Next, the function calculates the total cost of all items in the cart by calling the **calculateTotalCost()** method. This method sums up the total price of all items.

```
double total = bill.calculateTotalCost();
std::cout << "Total before discount: $" << total << std::endl;
```

c. Applying Discount (Optional)

If the user wishes to apply a discount, they can enter a percentage value. The program will then calculate the total cost after applying the discount by invoking the **calculateDiscountedTotal()** method from the **DiscountBill** class. The DiscountBill class overrides the calculateTotalCost() method to apply the discount.

```
double discountPercentage;
std::cout << "Enter discount percentage (if any): ";
std::cin >> discountPercentage;
DiscountBill discountBill;
```

```
discountBill.applyDiscount(discountPercentage);
double finalAmount = discountBill.calculateDiscountedTotal();
```

d. Displaying the Final Bill

Finally, the program will display the total cost after applying the discount (if any).

```
std::cout << "Total after discount: $" << finalAmount << std::endl;
```

The final output will show the item list, the total before and after discount, and the discounted total.

Step 5: Exit Program

If the user chooses to exit the program (choice 3), the program will clear the saved data from the totalbill.txt file by deleting it. This ensures that all transaction data is wiped out, maintaining data privacy for the next session.

```
std::remove("totalbill.txt");
std::cout << "Thank you for shopping! Your transaction has been completed." <<
std::endl;
```

The program will then terminate gracefully, concluding the shopping session.

Step 6: Invalid Input Handling

At any point, if the user enters invalid input (e.g., selecting a menu option outside the allowed range or entering a non-numeric value for item price or quantity), the program will catch the error and prompt the user to re-enter the correct value.

For example, if the user selects a menu option that is not between 1 and 3, the program will display an error message and prompt them to try again.

```
if (choice < 1 || choice > 3) {
    std::cout << "Invalid option! Please select a valid option." << std::endl;
}
```

🎥 Outputs Explained

1. Start Shopping

Input: The user enters item details (name, price, quantity).
Output: The program saves the item details (name, price, quantity, and total cost) to totalbill.txt and displays a success message confirming the addition of the item.
Example:

- User adds a "Laptop" (price: $1000, quantity: 2).
- The program saves "Laptop: $2000" and displays: "Item 'Laptop' added to your shopping list."

2. View Total Bill

Input: The user enters a discount percentage (e.g., 10%).
Output: The program lists all purchased items with their details and calculates the total before and after applying the discount.
Example:

- Items:
  - "Laptop: $1000 x 2 = $2000"
  - "Phone: $500 x 3 = $1500"
- Total before discount: $3500
- After 10% discount: $3150
  The program outputs the item list and the total amounts.
-

3. Exit Program

Action: The program terminates, and the totalbill.txt file is deleted, clearing all saved shopping data.
Example: A message like "Thank you for shopping" confirms the program has ended and data has been cleared.

Sample Outputs

```
========================================
            IUBAT SUPER SHOP
 ========================================
  [1] Start Shopping
  [2] View Total Bill
  [3] Exit Program
 ========================================
 Enter your choice:
```

```
========================================
            Add to Cart Menu
========================================
 [1] Add Item to Cart
 [2] Return to Main Menu
========================================
Enter your choice:
```

```
========================================
            Add Item Details
 ========================================
 Enter item name: Apple
 Enter item price: 450
 Enter item quantity: 12
```

```
========================================
            Items Purchased
 ========================================
 Apple - 450 x 12 = 5400
 ========================================
 Enter discount percentage: 5
 Total Amount (Before Discount): 5400
 Discount Applied: 5%
 Total Amount (After Discount): 5130
 ========================================
 ========================================
            View Total Bill
 ========================================
  [1] View Bill
  [2] Return to Main Menu
 ========================================
 Enter your choice:
```

❗ Handling User Inputs

Case 1: Invalid Menu Option

- **Input:** A number outside the menu range (e.g., 4).
- **Output:** Displays an error message and prompts for re-entry.

Case 2: Non-Numeric Input for Numeric Fields

- **Input:** Entering text for price or quantity.
- **Output:** Program crashes (can be handled by adding input validation).

Case 3: Empty Input for Item Name

- **Input:** Pressing Enter without entering a name.
- **Output:** Item added with an empty name (not ideal; can be enhanced).

🔗 Project Repository

You can access the full project repository for the Super Shop Billing System on GitHub by clicking the link below.
Super Shop Billing System Github Repository > username – wasif-h

Link - https://github.com/wasif-h/Super-Shop-Billing-System.git

🏆 Conclusion

The workflow of the **IUBAT Super Shop Billing System** ensures that the entire billing process is streamlined and user-friendly. By using OOP principles, the system organizes the tasks into well-defined classes and functions, improving code readability and maintainability. The program guides the user through a simple, clear sequence of interactions, from adding items to generating the total bill and applying discounts. The use of file handling ensures that transaction data is saved and can be retrieved if necessary, making the system efficient and reliable.

The code allows the user to manage the entire shopping and billing process from start to finish with minimal input, providing quick and accurate results. It can be extended in the future to incorporate additional features, such as tax calculations, loyalty programs, and even barcode scanning, to make it a complete solution for small and medium-sized retail businesses.

Thank You