Lab Exercise 6

Task 1: Implement and insert the values “BORROWROB” in the stack and identify if it’s a palindrome or not. Use the push and pop functions to accomplish this (Note: Use Arrays to accomplish this)

Task 2: You are required to design a simple Store Transaction Manager for a small shop using a stack implemented with a linked list in C++. The shopkeeper wants to keep track of all sales and refunds as transactions, where each transaction is stored as an object containing a unique transaction ID, amount, description, and a flag value to indicate whether it is a sale or a refund. A sale means the shop sold an item (money received), so the amount is positive, while a refund means the shop returned money to a customer, so the amount is negative. The flag is used to represent this: if the amount ≥ 0, set flag = 1 (Sale), and if the amount < 0, set flag = 0 (Refund).

When a new transaction is added (pushed) to the stack, it is given a unique ID starting from 1, and its description is checked — if it is longer than 20 characters, it should be cut short and “...” added at the end. Before storing a sale transaction, apply a discount on the amount based on the following rules:

- If the sale amount ≥ 500 PKR → give a 5% discount
- If the sale amount ≥ 1000 PKR → give a 15% discount
- If the sale amount ≥ 1500 PKR → give a 30% discount

This means that the final stored amount must be after the discount has been applied. Refund transactions are stored as negative amounts and are not discounted.

You are given a predefined list of 7 transactions (an array of objects), where some are sales and some are refunds. From this list, your program should randomly select transactions and store them in the stack (implemented with a linked list). When a transaction is removed (popped) from the stack, it means the shopkeeper has cancelled or reversed it. In that case, the amount should be multiplied by -1 to change its sign, the description should have " [REVERSED]" added at the end, and the flag should be updated to indicate it has been reversed.

For example, in the main() function, you can create a stack of transactions and randomly push a few entries such as:

- Sale of Rs. 1200 → "Sale: Blue Jacket" (after 15% discount = Rs. 1020)
- Sale of Rs. 450 → "Sale: Cotton Socks" (no discount)
- Refund of Rs. -300 → "Refund: Defective Shirt"
- Sale of Rs. 1700 → "Sale: Leather Jacket" (after 30% discount = Rs. 1190)

After adding them, display all transactions in the stack. Then perform infix to postfix conversion and evaluation for a mathematical expression like (100 + 20) * 0.9 - 5, which should convert to postfix form 100 20 + 0.9 * 5 - and evaluate to 103.0.

Next, remove (pop) one transaction from the stack — for example, remove the refund transaction. After popping, this transaction should be shown as reversed, with its amount sign changed and description updated. Finally, display the remaining transactions still in the stack.

This problem helps you practice how a stack using a linked list works with object manipulation during push and pop operations, how to apply conditional discounts, and how to convert and evaluate mathematical expressions using stacks (infix-to-postfix conversion and evaluation).

You can use the **rand()** function in C++ from the library **<cstdlib>** (and **<ctime>** to seed random numbers).

Example:

#include <cstdlib>
#include <ctime>
	// **Initialize random seed once in main()**
	srand(time(0));
	
	// **Generate a random value from 0 to 10**
	int randomIndex = rand() % 10;

Sample Input and outputs

Pushed Transactions: 1. Sale: Blue Jacket (1200) → Discount 15% → Final: 1020.00 2. Sale: Cotton Socks (450) → No discount 3. Sale: Leather Jacket (1700) → Discount 30% → Final: 1190.0(4. Refund: Defective Shirt (-300)	**Intermediate Expression Calculation:** Infix: (100 + 20) * 0.9 - 5 Postfix: 100 20 + 0.9 * 5 - Evaluated Result: 103.00
Pop (remove) one transaction: Popped Transaction: Refund: Defective Shirt [REVERSED] Amount changed from -300 to +300 Flag updated to 2	

Final Stack Output:

Top → [id=3, amt=1190.00, desc="Sale: Leather Jacket", flag=1]

[id=2, amt=450.00, desc="Sale: Cotton Socks", flag=1]
Bottom → [id=1, amt=1020.00, desc="Sale: Blue Jacket", flag=1]

Task 3: As a software developer, you have been assigned the task of developing an application for a busy restaurant that can handle a large volume of orders. To ensure that orders are processed in the order they were received, what type of data structure would you choose to store the orders? Please describe the necessary operations that would be required to add and remove orders from this data structure, including the quantity of each item. Additionally, how would you prevent the data structure from becoming full or empty to avoid overflow or underflow? Finally, explain your approach for printing out all the processed orders in the order they were received to ensure that the restaurant staff can efficiently fulfill the orders.