VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering (SCOPE)

Digital Assignment-1

Fall Semester 2024-25

COURSE CODE: BCSE202l

COURSE TITLE: Data Structures and Algorithms

FINANCE TRACKER

Team:

- **Thirushika V 23BCB0154**
- **Shree Lakshmi K 23BCE2120**
- **K Pranathi 23BCT0260**

INDEX

Introduction

It is quite simple for us students to lose track of our expenditures because we use digital currency for many transactions. To ensure we don't spend too much money, it is crucial for us to keep track of our expenditures. And this is the point at which our finance tracker is useful. Entering the transaction amount, date, and category is all that is required. You may also include a description for your transaction! The amount that has been credited and debited from your account is displayed, together with the current balance and the difference after all transactions. Lastly, the transactions are arranged according to their transaction sequence. We have achieved these functions by storing each of the transactions into a node and sorting them using bubble sort.

What is a Finance Tracker?

Financial tracking, also known as expense tracking, is the process of keeping tabs on your income and spending, ideally on a daily basis.

The finance tracker is a program designed to manage and track personal financial transactions efficiently using a doubly linked list data structure. It allows users to record, sort, and display their financial transactions while maintaining an updated bank balance.

Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

typedef struct Transaction {
    char date[11];
    double amount;
    char category[50];
    char description[100];
    int type; // takes 1 for credit and -1 for debit.
} Transaction;

typedef struct Node {
    Transaction transaction;
    struct Node* next;
    struct Node* prev;
} Node;

typedef struct DoublyLinkedList {
    Node* head;
    Node* tail;
} DoublyLinkedList;

```

```c
Node* createNode(Transaction transaction) {
    Node* newNode = malloc(sizeof(Node));
    if (!newNode) {
        fprintf(stderr, "Memory allocation failed for new node.\n");
        exit(EXIT_FAILURE);
    }
    newNode->transaction = transaction;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

bool isLeapYear(int year) {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

```

```c
bool isValidDate(const char* date) {
    int day, month, year;
    if (sscanf(date, "%4d-%2d-%2d", &year, &month, &day) != 3) {
        return false;
    }
    if (month < 1 || month > 12) {
        return false;
    }
    int daysInMonth;
    switch (month) {
        case 1: case 3: case 5: case 7: case 8: case 10: case 12:
            daysInMonth = 31;
            break;
        case 4: case 6: case 9: case 11:
            daysInMonth = 30;
            break;
        case 2:
            daysInMonth = isLeapYear(year) ? 29 : 28;
            break;
        default:
            return false;
    }

    return day >= 1 && day <= daysInMonth;
}
```

```c

DoublyLinkedList* createList() {
    DoublyLinkedList* list = malloc(sizeof(DoublyLinkedList));
    if (!list) {
        fprintf(stderr, "Memory allocation failed for list.\n");
        exit(EXIT_FAILURE);
    }
    list->head = NULL;
    list->tail = NULL;
    return list;
}

void insert_Transaction(DoublyLinkedList* list, Transaction transaction) {
    Node* newNode = createNode(transaction);
    if (!list->head) {
        list->head = newNode;
        list->tail = newNode;
    } else {
        list->tail->next = newNode;
        newNode->prev = list->tail;
        list->tail = newNode;
    }
}

```

```c
void sort_Transactions(DoublyLinkedList* list) {
    if (!list->head) return;

    Node* current;
    Node* index;
    Transaction temp;

    for (current = list->head; current != NULL; current = current->next) {
        for (index = current->next; index != NULL; index = index->next) {
            if (strcmp(current->transaction.date, index->transaction.date) > 0) {
                temp = current->transaction;
                current->transaction = index->transaction;
                index->transaction = temp;
            }
        }
    }
}

void display_Transactions(DoublyLinkedList* list) {
    Node* current = list->head;
    while (current) {
        printf("%s: %.2f [%s] - %s\n",
               current->transaction.date,
               current->transaction.amount,
               current->transaction.category,
               current->transaction.description);
        current = current->next;
    }
}

```

```c
double Calculate_Total_Transaction(DoublyLinkedList* list) {
    double total = 0.0;
    Node* current = list->head;
    while (current) {
        total += current->transaction.amount;
        current = current->next;
    }
    return total;
}

void Update_BankBalance(double* balance, double amount) {
    *balance += amount; // Adds credit to bank balance and subtracts debits from bank balance.
}

void Calculate_Credits_Debits(DoublyLinkedList* list, double* totalCredits, double* totalDebits) {
    Node* current = list->head;
    while (current) {
        if (current->transaction.type == 1) {
            *totalCredits += current->transaction.amount; // Calculating total credits.
        } else if (current->transaction.type == -1) {
            *totalDebits += current->transaction.amount; // Calculating total debits.
        }
        current = current->next;
    }
}

```

```c
double Calculate_Balance_Change(double initialBalance, double finalBalance) {
    return initialBalance - finalBalance;
}

void Free_List(DoublyLinkedList* list) {
    Node* current = list->head;
    while (current) {
        Node* next = current->next;
        free(current);
        current = next;
    }
    free(list);
}

```

```c
int main() {
    DoublyLinkedList* financeList = createList();
    char continueInput;
    double bankBalance = 0.0;

    printf("Enter your bank balance from the prefered date of calculation: ");
    scanf("%lf", &bankBalance);

    do {
    Transaction transaction;

    printf("\nEnter transaction details:\n");
    char dateInput[11];
    while (1) {
        printf("Date (YYYY-MM-DD): ");
        scanf("%s", dateInput);
        if (isValidDate(dateInput)) {
            strcpy(transaction.date, dateInput);
            break; // Exit the loop if date is valid
        } else {
            printf("Invalid date format. Please enter a valid date (YYYY-MM-DD).\n");
        }
    }
```

```c
    printf("Amount: ");
    scanf("%lf", &transaction.amount);

    printf("Is this amount credited or debited? (c/d): ");
    char typeInput;
    scanf(" %c", &typeInput);
    transaction.type = (typeInput == 'c' || typeInput == 'C') ? 1 : -1;

    printf("Category: ");
    scanf("%s", transaction.category);

    printf("Description: ");
    getchar(); // Clear newline character
    fgets(transaction.description, sizeof(transaction.description), stdin);
    transaction.description[strcspn(transaction.description, "\n")] = 0;

    Update_BankBalance(&bankBalance, transaction.type * transaction.amount);
    insert_Transaction(financeList, transaction);
    printf("Current bank balance: %.2f\n", bankBalance);

    printf("Do you want to add another transaction? (y/n): ");
    scanf(" %c", &continueInput);

    } while (continueInput == 'y' || continueInput == 'Y');

```

```c
    sort_Transactions(financeList);
    printf("\nTransactions in order:\n");
    display_Transactions(financeList);//Displaying the Transaction History.

    double totalAmount = Calculate_Total_Transaction(financeList);///(Calling function to Calculate
                                                    // the Total Transactions  made by the user.)
    printf("\nTotal Transactions Made: %.2f\n", totalAmount);

    double totalCredits = 0.0;
    double totalDebits = 0.0;
    Calculate_Credits_Debits(financeList, &totalCredits, &totalDebits);;//((Calling the function to
                                                                // calculate the total credit
                                                                // amount and tota debit
                                                                // amount made.)

    printf("Current bank balance: %.2f\n", bankBalance);///(Function to Display the Bank
                                                        //Balance after all the transactions.)
    printf("Total credited: %.2f\n", totalCredits);
    printf("Total debited: %.2f\n", totalDebits);

    double CurrentBalance = bankBalance-totalDebits+totalCredits;
    double balanceChange = Calculate_Balance_Change(CurrentBalance, bankBalance);
    printf("Change in bank balance: %.2f\n", balanceChange);///(Displaying the
                                                            //difference from the inital
                                                            //bankbalnce and current BankBalance)

    Free_List(financeList);

    return 0;
}
```

Input:

```
PS E:\java new> cd "e:\java new\" ; if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunner
Enter your bank balance from the prefered date of calculation: 75000.80

Enter transaction details:
Date (YYYY-MM-DD): 2024-10-30
Amount: 300
Is this amount credited or debited? (c/d): D
Category: Food
Description: Lunch
Current bank balance: 74700.80
Do you want to add another transaction? (y/n): y

Enter transaction details:
Date (YYYY-MM-DD): 2024-10-31
Amount: 500
Is this amount credited or debited? (c/d): d
Category: Bills
Description: Groceries
Current bank balance: 74200.80
Do you want to add another transaction? (y/n): y

Enter transaction details:
Date (YYYY-MM-DD): 2024-11-1
Amount: 20000
Is this amount credited or debited? (c/d): c
Category: Salary
Description: Google
Current bank balance: 94200.80
Do you want to add another transaction? (y/n): y

Enter transaction details:
Date (YYYY-MM-DD): 2024-11-2
Amount: 2000
Is this amount credited or debited? (c/d): c
Category: Gift
Description: Diwali
Current bank balance: 96200.80
Do you want to add another transaction? (y/n): y
```

```
Enter transaction details:
Date (YYYY-MM-DD): 20000
Invalid date format. Please enter a valid date (YYYY-MM-DD).
Date (YYYY-MM-DD): 2024-11-2
Amount: 20000
Is this amount credited or debited? (c/d): d
Category: Accessory
Description: Earphones
Current bank balance: 76200.80
Do you want to add another transaction? (y/n): y

Enter transaction details:
Date (YYYY-MM-DD): 2024-11-3
Amount: 200
Is this amount credited or debited? (c/d): d
Category: Food
Description: Evening Snacks
Current bank balance: 76000.80
Do you want to add another transaction? (y/n): y

Enter transaction details:
Date (YYYY-MM-DD): 2024-11-3
Amount: 578.90
Is this amount credited or debited? (c/d): d
Category: Bills
Description: Recharge
Current bank balance: 75421.90
Do you want to add another transaction? (y/n): n
```

Output:

```
Transactions in order:
2024-10-30: 300.00 [Food] - Lunch
2024-10-31: 500.00 [Bills] - Groceries
2024-11-1: 20000.00 [Salary] - Google
2024-11-2: 2000.00 [Gift] - Diwali
2024-11-2: 20000.00 [Accessory] - Earphones
2024-11-3: 200.00 [Food] - Evening Snacks
2024-11-3: 578.90 [Bills] - Recharge

Total Transactions Made: 43578.90
Current bank balance: 75421.90
Total credited: 22000.00
Total debited: 21578.90
Change in bank balance: 421.10
PS E:\java new>
```

Data Structures and Algorithms used

Data Structure used: Doubly linked list.

Sorting Algorithm used: Bubble Sort.

Doubly Linked List:

❖Structure of a Doubly Linked List:

1.Node: Each item in the list.

- A node holds:
  a. Data: The actual content, like a transaction
  b. Next Pointer: Points to the next node
  c. Previous Pointer: Points to the previous node.

2. Head: Points to the first node in the list.

3. Tail: Points to the last node in the list.

❖Doubly Linked List is Considered advanced for the following reasons:

➢ Two-way Navigation:

Each node in a doubly linked list contains 2 pointers (i.e. previous and next). With the help of the 2 pointers, we can traverse through the list in both forward and backward directions.

- Enhanced Insertion and Deletion:

  As each node has pointers pointing to the next and the previous nodes, we don't have to traverse the entire list to find the predecessor of a node during insertion and deleting process and saves time.

- Complexity in Implementation:

  The implementation of a doubly linked list is generally more complex than that of a singly linked list. Developers must manage additional pointers and ensure proper linkage in both directions, increasing the potential for errors if not handled carefully.

- Memory Overhead:

  Each node requires additional memory for the extra pointer, making doubly linked lists more memory-intensive than singly linked lists.

❖ Why Doubly Linked List for Finance Tracker??

Doubly Linked List is a good data structure for finance tracker because each node stores transaction details like date of transaction, amount, credit/debit, category and description of the transaction. The two-way navigation (forward and backward) makes it flexible and efficient for keeping record of all transactions.

- Adding Transactions is Easy: Each new transaction just gets added to the end of the list, which is quick and efficient.
- Sorting Transactions by Date: Since each node links to the next and previous nodes, sorting by date is easier to do.
- Viewing and Calculating Total: You can go through the list easily to display transactions or calculate totals, which makes it simple to see how much you've spent or earned.
-Using Memory Efficiently: The list only uses memory for the transactions entered, so it's efficient and doesn't waste space.

- Parts of the Doubly Linked List in the Program:

1. Transaction Structure:

This holds details for each transaction, such as:

-The date of the transaction in YYYY-MM-DD format.

- The amount of money involved.

- The category (e.g., "Food" or "Rent").

- A description to explain what the transaction was for.

- The type of transaction, where 1 means "credit" (money added) and -1 means "debit" (money spent).

2. *Node Structure:*

Each node holds:

- A Transaction, which is one entry of the data mentioned above.

- A pointer to the next node in the list.

- A pointer to the previous node in the list.

❖Working of the Program:

1. Creating a Node: When a new transaction is entered, the program makes a new node for it using the createNode() function. This function allocates memory and fills in the transaction details for that node.
2. Adding Transactions to the List: The insert_Transaction() function adds each new node to the end of the list. If this is the first transaction, it becomes both the head and the tail of the list. For each new transaction, it links to the current tail, making it easy to add new transactions in order.
3. Sorting Transactions: The sort_Transactions() function arranges the transactions by date, so they appear in order. It goes through the list and compares dates to arrange them correctly.
4. Displaying Transactions: The display_Transactions() function goes through each node from the head to the tail, printing out each transaction's details. This way, you can see all your transactions in order.
5. Calculating Totals: The program includes functions like Calculate_Total_Transaction() and Calculate_Credits_Debits() to add up the amounts for all transactions. It goes through each transaction and checks if it's a credit or debit, then keeps a running total for each type.

6. Freeing Memory: After the program is done, the Free_List() function frees up the memory for each node in the list so that the program doesn't use extra memory.

❖Algorithm Used:

The Bubble sort algorithm is used in this program to achieve chronological order of the transaction.

Algorithm:

```
Void BubbleSort(node* head)
{
    if(!head)
            return;
     bool swapped;
     node* current;
     node* last=NULL;
     do
     {
       swapped=false;
       current=head;
       while(current->next!=last)
      {
           if(current->data>current->next->data)
            {
               int temp=current->data;
               current->data=current->next->data;
               swapped=true;
            }
   current=current->next;
     }
     last=current;
   }
}
```

Features

Key Features:

❖ Transaction Recording:

Users can input transactions with details such as date, amount, category, description, and type (credit or debit).

Transactions are stored in nodes of a doubly linked list, facilitating efficient insertion, deletion, and traversal.

❖ Date Validation:

The program includes a function to validate the date format and ensure that the entered date is correct (considering leap years and month-specific days).

❖ Transaction Sorting:

Transactions are sorted by date to provide an organized view of the transaction history.

❖ Display and Summary:

The program displays all recorded transactions in order.

It calculates and displays the total amount of transactions, total credits, total debits, and the change in bank balance.

It provides an updated bank balance after each transaction, helping users keep track of their financial status.

❖ Memory Management:

The program ensures proper memory allocation for the nodes in the doubly linked list and includes a function to free the allocated memory to avoid memory leaks.

Video Link

https://youtu.be/Onm4dwi9Zvs?feature=shared

Presentation PPT Link:

https://drive.google.com/drive/folders/1c9IBa7g8xuw8nO4x14u2qwyrGLySxSgN?usp=sharing

Conclusion

The Finance Tracking app built using a doubly linked list is a great tool for anyone who wants to know where all their money goes. It provides easy addition and organization of the transactions made by the user. It gives them their transactional summary in a second. The efficient use of memory and the ability to navigate in both the directions makes it an user friendly application.

THANK YOU

Team:

- **Thirushika V 23BCB0154**
- **Shree Lakshmi K 23BCE2120**
- **K Pranathi 23BCT0260**