Expenses Tracker - Final Project Report

Htut Htet Naing, Htet Myat Phone Naing

Parami University

Data Structure and Algorithms

Prof. Aye Hnin Khine

25 May, 2025

Problem statement

The Problem Statement: The problem with many people's financial problems is that they cannot manage their expenses. Not only they cannot manage, but also, they cannot track or analyze how much they are spending on what area on both daily and monthly basis. It is also very time and energy consuming to write down each expense throughout the day and calculate their total spending amount timely based on their respective spending topics as day goes by, for days and months. Many people simply cannot follow this pattern as it needs their attention to detail levels. These emotions, energy and time cost more than their actual spent money. So, they simply skipped tracking their expenses and never tracked them anymore. And they can never know how much they spent throughout the day and month and can never know how much they can still spend or are spending in the whole day or month. They just cannot manage their expenses.

Our Solution: Thus, we are here to help with our expense tracking programme, called "the Expense Tracker." What we are going to solve is we note down your detailed expenses with specific areas of categories you have spent a specific amount on the specific day. We also show you the cumulative combined amount of how much you have spent throughout the day both in terms of categories and total amount for daily view and for monthly view, how much you have spent throughout the month with specific days you spent your money for. All you need to do is simply choose the category of choice and type the amount you have spent on the specific day. The rest is all upon us.

How to Use:

If you are going to add new expense input in a day, you first have to input the date which is (DD-MM-YYYY) format, and after that, you will see the ordering lists of expenses categories with number ordering from 1 to 10.

If you want to add the expense for transportation, you have to type 1.

Therefore, another box will pop up and ask you to type the amount.

After you have finished adding all of your expenses in a day, you can type 0 in the box, to stop adding and then it will show the total expense amount of the day.

You cannot choose the number out of range which is from 1 to 10, or you cannot type the text. Filling the invalid information will be an error and will ask you again to fill.

How to use: First you have 5 choices of Menu to choose from,

1. Add New Expense
2. View Daily Expenses
3. View Monthly Expenses
4. Delete Expense
5. Exit

For each option, you have to type a number and continue with exact instructions for each stage. If you input the wrong value, you will be redirected to the previous page which requests you to input the exact values as per instructions. You can simply exit the whole programme by typing 5 in the main menu pane.

Explanation of The Chosen Data Structures

The data structures we used in our programme are "Stack" and "LinkedList" data structures.

Stacked Data Structure

The Stacked data structure is used for Last In First Out (LIFO) operations for the daily view and monthly view purposes of the data. Showing back the most recent expense on the daily view with LIFO operation, the data structure "Stack" is very useful and efficient for our project. Pushing if we found expenses in daily_expenses_stack onto the stack and Pop daily_expenses_stack from the list to clear it for a new daily_expenses for a new day is very practical and efficient usage.

LinkedList Data Structure

The LinkedList data structure is used for memory management and efficient storing of data to save memory. As Linked Lists nodes store in whenever there's free spaces of memory and does not need to be orderly stored, it is very efficient for storing the complete record of all expenses (self.expenses in the Tracker class) in our programme. Its dynamic memory management system is an efficient O(n) solution with insertion and deletion in the middle of the LinkedList. Its dynamic sizing grow or shrink without any reallocation of large blocks of memory really gives our program memory efficiency by only holding the elements it currently holds.

Code Walkthrough

For the whole project, it was based on time, so we imported time and datetime libraries for the project.

For Step-1, we created a class Node, to hold data for the Expense object for the next pointer, for the LinkedList properties.

For Step-2, we created a class LinkedList, as a primary class to store for all expense records. Inside it, we made append, delete_node, and is_empty functions for this class later usage.

For Step-3, we created a class Stack. This Last In First Out (LIFO) property is mainly used for viewing daily expenses from the most recent one to pop out first. We made push, pop, is_empty, and peek functions as usual for Stack data structure.

For Step-4, we created a class Expense. This class will make objects for every time the user has spent one expense transaction. In the initiator function, date, category, amount, and description (optional) are included. It also has important input validation and error handling. In error handling, for date, it is date format validation, in category, its non-empty string, and for amount, it's positive integer and float data types.

For Step-5, we created a class Tracker. This is the central class for all functions of the whole project, to track the expenses. This works as a media for the user, data structures and objects. This makes expense objects from LinkedList from valid categories (multiple choice) and puts them in Stack data structure for viewing purposes. For valid categories, we made the helper function _get_category_input and error handled that. Then we made add_expense function to add date, amount, and description (optional) with error handling for category input. Then we have the view_daily_expenses function which requires a full input date string to show total spent amount for each category and for every category combined for that specific day. We also made an error handling to show that nothing was found if no expense was inserted on that day. Then we similarly made the view_monthly_expenses function which requires only month and year to show total spent amount for each category and for every category combined for that specific month. There's also an error handling if no expense was found. Then we finally have delete_expense function, for the situation where we wrongly inserted the data. It asks the user to input date, category, and amount of the expense, and then look for the expense object with that specific information, and if found, delete it. It also has error handling if no expense object with such provided information is found.

For Step-6, we have display_menu which is the main menu for the user once the programme starts to run. The user has to choose one of these Add New Expense, View Daily Expenses, View Monthly Expenses, Delete Expense and Exit function to proceed. We also have a helper function called run_tracker to match these functions with respective numbers to make multiple choices for the user to choose. Then based on the user's choice, the exact instructions to go on forward steps for each step will appear.

Time Complexity

Why these linked lists and stack data structure?

Linked List is a linear data structure and its elements are stored in nodes, or each node consists of a reference to the next node in the sequence. It can be grown easily or removed as we want to add new expenses and delete if wrong. Additionally, it doesn't need to pre-allocate a certain amount of memory, and it is good as we don't know how much is the expense in advance. We use stack as we want to track back the previous expense and yesterday expense to give an output back to the user.

```
Number of Expenses: 10
  Add Time = 0.000051s
  View Daily Time = 0.000004s
  View Monthly Time = 0.000004s
  Delete Time = 0.000003s

Number of Expenses: 100
  Add Time = 0.000902s
  View Daily Time = 0.000011s
  View Monthly Time = 0.000014s
  Delete Time = 0.000002s

Number of Expenses: 1000
  Add Time = 0.074312s
  View Daily Time = 0.000090s
  View Monthly Time = 0.000075s
  Delete Time = 0.000002s

Number of Expenses: 5000
  Add Time = 0.371277s
  View Daily Time = 0.000787s
  View Monthly Time = 0.000759s
  Delete Time = 0.000005s
```

These running time results show that as the number of expenses (amount) increases, the running time will increase as well. It is basically proportional to the number of increasing expenses, which is from 100 to 1000. The running time complexity is in linear time complexity (O(N)).