Developer Manual

Wallet: Income & Expenses

2022.11.27

—

Abdelhamid Ahbane
N85AOM
BSc in Electrical Engineering
Budapest University of Technology and Economics, VIK

Outline :

1. Requirements
2. Structures
3. Functions & Key algorithms

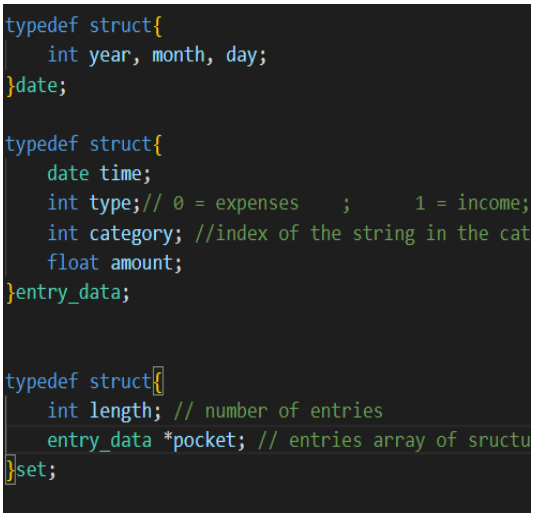
1. Requirements :

The program is developed and tested in Visual Studio Code.

For data structure, Structures are used to organize the data for easy access. 2d dynamic arrays are used for optimal performance and memory saving. File handling is also used to store all data in ".txt" files, and read data from it each time the user opens the program again.

Functions are used for every task in the code, with intuitive names, and detailed comments above every one of them.

2. Structures:



```
typedef struct{
    int year, month, day;
}date;

typedef struct{
    date time;
    int type;// 0 = expenses    ;      1 = income;
    int category; //index of the string in the cat
    float amount;
}entry_data;


typedef struct{
    int length; // number of entries
    entry_data *pocket; // entries array of sructu
}set;
```

Structure 1: Date

Stores the date of the entry. It contains 3 integer components for the year, month, and day.

Structure 2: Entry_data

Stores all the data related to the main input of the wallet. Its components are :
a Date structure (time), and an integer type (0 means expenses, 1 means income). An int-named category to store the index of the category strings. And finally a float for the amount of money in euros.

Structure 3: set

Stores the array of structures containing the entries, as well as their length.

2D Array: categories list

```
char **categories_list=(char**)malloc( (*p_cat_len)*sizeof(char*) );
char*** p_categories_list = &categories_list;
for(int i=0; i<(*p_cat_len); i=i+1)
    categories_list[i]=(char*)malloc(100*sizeof(char));
```

The 2d array stores the strings of the list of categories. Here the dynamic memory allocation is used to keep growing the array when the user enters a new category. We also keep track of its length in the "*p_cat_len" pointer.

3. Functions & key algorithms:

Functions 1,2,3:

```
date enter_date(void);
int enter_type(void);
float enter_amount(void);
```

1- **Enter date** function doesn't receive any arguments. It returns a date structure type. It is responsible to read the (yyyy.mm.dd) from the user and assign the values to "date struct", then returning it.

2-**Enter type** function doesn't require any arguments. It returns an int value (0 or 1) for the type of entry (0 = expenses) & (1 = income).

3-**Enter amount** function doesn't require any arguments. It returns a float value for the amount of money.

Function 4:

```
int choose_category(char ***strings, int total);
```

choose_category function receives the pointer to the 2d array of strings, and the total number of strings (length of categories list) as an argument.

The function will print all the categories to the user and ask them to choose one from the list as an integer option. Then it returns the value of this integer which corresponds to the index of the string in the "categories_liste" array.

Functions 5,6:

```
32    //add new entry
33    entry_data* add_entry(char ***strings, int total);
34    void add(set *a, entry_data *input);
35
```

5- **Add_entry** function receives the pointer to the 2d array of strings, and the total number of strings (length of categories list) as an argument.

It calls the previous functions (1,2,3,4), assigns their return values into a "entry_data" structure, and returns the pointer to this struct.

The struct is dynamically allocated here, so it must be freed by caller.

6-**Add** function receives the pointer to the structure called "set", and the pointer to the entry from the function (5). This function is responsible to add the structure of the entry (date, type, category, ammount) to the array of structs saved in the set. It will also increase the length of the set by one each time its used.

Functions 7:

```
//print menu
void print_menu( int* option, char*** p_categories_list, int* cat_len, set* wallet1, entry_data *input);
```

7- **Print_menu** function responsible for showing the menu to the user at the biginning of the programm, and after finishing any option as well.

The function receives all the arguments it need to pass to all the other functions inside of it. It doesn't return anything.

The function is inside a while loop in the main function, the only way to get out of the loop is by choosing option 4 from the menu. (Choosing an invalid option will show an error message and display the menu again).

Functions 8:

```
//add a new category to the list
void add_new_category(char ***strings, int *p_total);
```

5- **Add_new_category :** as the name indicates, this function asks the user to enter a string (name of the new category), then it will add this string to the categories_liste. Receives pointer to the list of strings, and pointer to the total number of categories. No return value.

Algorithm used :

First it creates a new 2d array of strings with length +1.

Copy old strings to the new array.

Add the new string in the last element.

Free the old dynamic array.

Assign address of the new array to the old one.

Functions 9:

```
//view statistics
void view_statistics(char*** p_categories_list, int* cat_len, set* wallet1 );
```

5- **view_statistics :** This function receives 3 arguments : the pointer to list of categories, the pointer to the number of categories, and the set containing all the data of entries. The function is only responsible for printing the values to the consol, therefore it has no return value.

Algorithms used :

1. A simple for loop that goes through all the "wallet1" elements and if conditions to make sure the values are inside the time interval. For the first category: summing up all the incomes and expenses till the end of the loop. Then printf after end of the loop and initialiying the total income and expenses again for use in the next category.
2. For finding most expensive category: creating a float array of size (number of categories), supposing value of first index is the max, then comparing all values with it. Changing this max to the new max if exists. Finally printing the name of the most expensive category using the index "max", and printing the value inside of it as well.