Test 02 – Question 02

Overview:

You have just been hired as a Software Developer to work at one of Canada's largest banks. The bank has decided while a great deal of software exists for electronic funds they want to develop an application that keeps track of physical cash a person has in their wallet.

As a new hire you have been tasked with developing a proof of concept for this application using the **RAPTOR** programming language and **parallel arrays**.

Instructions:

Start by downloading the starting files, `Question02.py` and `wallet.txt`, under Week 15's content on D2L. After opening this file you will notice that you have been given a = completed `main` method and series of additional functions that you must implement. The instructions outlining what each function should do are provided below.

Step 01 – readFromWalletFile()

Reads the starting values for an individual's `wallet` from the file **wallet.txt**. You may assume this file contains 8 lines of integer data in the format:

- Line 1 – Number of **$20 Bills** in wallet
- Line 2 – Number of **$10 Bills** in wallet
- Line 3 – Number of **$5 Bills** in wallet
- Line 4 – Number of **Toonies** in wallet
- Line 5 – Number of **Loonies** in wallet
- Line 6 – Number of **Quarters** in wallet
- Line 7 – Number of **Dimes** in wallet
- Line 8 – Number of **Nickles** in wallet

The `readFromWalletFile` function must read the above data into a 1D list of the below format and then `return` this array from the function.

```
[num_20Bills, num_10Bills, num_5Bills, num_toonies, num_loonies,
num_quarters, num_dimes, num_nickels]
```

Step 02 – printMenu()

This function should display the menu:

```
*****MENU*****
1. Display Wallet Contents
2. Add to Wallet
```

```
3. Subtract from Wallet
4. Total Wallet Value
5. Purchase Item
6. Exit
```

After the function displays the above menu the `printMenu` function should prompt the user to make a valid section, from 1 to 6, and return the selection.

If the user attempts to make an invalid selection (i.e. a number outside of 1-6) continually re-prompt the user until a valid selection is made. The `printMenu` function should only return a number between 1-6.

Step 03 – displayWalletContents ()

The `walletContents` function is a non-value returning function that takes the parameter `wallet`. The function also makes use of the `names` 1D array which has been loaded in for you. Both `wallet` and `names` are parallel arrays of the length 8 (i.e. number of distinct coins/bills).

This function's task is to display the wallet contents to the console in the form:

```
*****Wallet*****
$20 Bills - 2
$10 Bills - 0
$5 Bills - 1
Toonies - 1
Loonies - 2
Quarters - 3
Dimes - 5
Nickels - 4
```

Each line above follows the format:
`<bill/coin name> – <number of that bill/coin in wallet>`

Step 04 – getInputs ()

The `getInputs` function returns two output parameters `coinsBillsName` and `numCoinsBill`.

`coinsBillsName` should be a value provided by the user indicating the name of the coin/bill they would like to add/subtract from their wallet. As such this value should be one of the values from the `names` array which has already been loaded for you.

`numCoinsBill` should be a value provided by the user indicating the number of coins/bills they would like to add/subtract from their wallet. As such this value should be a value of at least zero.

Step 05 – addToWallet ()

The `addToWallet` function is a non-value returning function that takes the three parameters `wallet`, `billCoinNameToAdd`, and `numBillsCoinsToAdd`. The function also makes use of the `names` 1D array which has been loaded in for you.

The purpose of this function is to update the `wallet` reference so that the `billCoinNameToAdd` is **increased** by `numBillsCoinsToAdd`; for example, the function call `addToWallet(wallet, "$20 Bill", 1)` would **increase** the number of `"$20 Bills"` in `wallet` by `1`.

Hint: Start by determining what index `billCoinNameToAdd` is at in `names`.

Step 06 – subtractFromWallet ()

The `subtractFromWallet` function is a non-value returning function that takes the three parameters `wallet`, `billCoinNameToSubtract`, and `numBillsCoinsToSubtract`. The function also makes use of the `names` 1D array which has been loaded in for you.

The purpose of this function is to update the `wallet` reference so that the `billCoinNameToSubtract` is **decreased** by `numBillsCoinsToSubtract`; for example, the function call `subtractFromWallet(wallet, "Quarters", 3)` would **decrease** the number of `"Quarters"` in `wallet` by `3`.

IMPORTANT: If you try to subtract more of a specific type of coin/bill than you actually have in `wallet` **do not** update `wallet` at all.

Step 07 – sumWalletContents ()

The `sumWalletContents` function is a value returning function that takes the parameter `wallet`. The function also makes use of the `values` 1D array which has been loaded in for you.

This function should `return` the total value of the coins and bills in `wallet`; for example, if `wallet` contains 2 `$20 Bills`, 0 `$10 Bills`, 1 `$5 Bills`, 1 `Toonies`, 2 `Loonies`, 3 `Quarters`, 5 `Dimes`, and 4 `Nickels` the sum of `wallet` should be $50.45.

Remember: `wallet` and `values` are parallel arrays.

Step 08 – getCostOfItem ()

The `getCostOfItem` function returns one output parameters `itemCost`.

`itemCost` should be a value provided by the user indicating the dollar value of the item they would like to purchase. Please ensure a proper input value is provided before returning back to the `main`.

Step 09 – calculateChange()

The `calculateChange` function is a <u>value returning function</u> that takes the two parameters `amountPaid` and `amountDue`. Both parameters represent dollar values of a sale; for example, if I bought an item that costs $12.33 and payed $15 cash, then `amountPaid` would be $15 and `amountDue` would be $12.33.

The function already contains starter code that should not be modified. This code provides a `values` list containing the values of the coins/bills and a list called `changeDue` which is the list you will update to return the bills/coins owed to the customer. `amountOwed` is the dollar amount of change owed to the customer.

When returning change to a customer the `calculateChange` function should return the <u>least number of coins/bills</u>. For example, in the case of a purchase of $12.33 after rounding to the nearest nickel value of $12.35 the change owed to the customer paying $15 cash is $15 - $12.35 = $2.65. The change due to customer would consist of 1 `Toonies`, 2 `Quarters`, 1 `Dimes`, and 1 `Nickels`. Therefore, the `calculateChange` function should return the following list for this example: `changeDue = [0, 0, 0, 1, 0, 2, 1, 1]`

Step 10 – purchaseItem()

IMPORTANT: Must have Step 09 completed in order to implement this function

The `purchaseItem` function is a <u>non-value returning function</u> that takes the two parameters `wallet` and `cost`. The function also makes use of the `values` 1D array which has been loaded in for you. The purpose of this function is to update `wallet` to the coins/bills remaining after a purchase of a specified `cost`.

For example, assuming `wallet = [2, 0, 1, 1, 2, 3, 5, 4]` and `cost = $12.33` then the final value of `wallet` should be `[1, 0, 2, 2, 2, 5, 6, 5]`.

Explanation:
When you `purchaseItem` you **<u>always</u>** pay for it using the largest coin/bill in excess of the amount to be paid available in your `wallet`; in our example, when we want to pay for an item costing $12.33 (rounded to nearest nickel = $12.35) and because we have 2 `$20 Bills` we will pay with one of them and therefore change owed is $7.65.

Our `calculateChange` function can be called as `calculateChange(20, 12.32)` because we are paying $20 on a purchase of $12.32. This call should return the list

`[0, 0, 1, 1, 0, 2, 1, 1]` as 1 `$5 Bills`, 1 `Toonies`, 2 `Quarters`, 1 `Dimes`, and 1 `Nickels` sums to $7.65 which is the amount of change due to the customer!

The last step is add the returned change, `[0, 0, 1, 1, 0, 2, 1, 1]`, to your `wallet`.

IMPORTANT: In the `purchaseItem` function if you do not have enough money in your `wallet` to complete the purchase display the message "`Insufficient funds in wallet!`" and exit the function. **Hint:** Use `sumWalletContents()`

Step 11 – writeToWalletFile()

Update **wallet.txt** to contain the current contents of `wallet`.
Review Step 01 for expected file format of **wallet.txt**