Project CS103 S2022
Coffee Machine

FERLIN Jules

INTERNATIONAL UNIVERSITY OF SARAJEVO
2004
IUS

Table of Contents

Subject

The subject is very simple. We have to simulate an entire coffee machine. The coffee machine could provide about 5 types of coffee with a limited amount and it will ask for money, add this money to stock, give change et reduce money in stock.

For example:

"If we ask for espresso, and it costs 1 KM, we give 2 KM, if you don't have coins to give back, (either 2X 0.5, or 1) you should send back the money, and say we don't have change.

If we make 50 espressos, and want to make the 51st. The machine should say, sorry we don't have this coffee any more, we have other types, would like to try (or something similar)

If we put 1 KM, the number coins in the machine will increase for 1 KM, if we put 5 KM, and machine give back 4X1 of 1 KM, then, the total of 1 KM coins should be reduced."

Answer

Structures

I made 4 structures in order to have the simplest program. First one is for drinks, it gathers name, price and the number of drinks who can be made. Second structures is for coins, it regroups the value of the coin and how many coins which remains. Third one is to store choice of user. It gathers a drink, money given, the number of the drink chosen and a structure to store coins given.

```
typedef struct {
    char name[20];
    float price;
    int number;
} struct_drinks;
```

```
typedef struct {
    struct_drinks d;
    int type;
    float money;
    struct_given given;
} struct_user;
```

```
typedef struct {
    int c05;
    int c1;
    int c2;
    int c5;
} struct_given;
```

```
typedef struct {
    float value;
    unsigned int number;
} struct_coins;
```

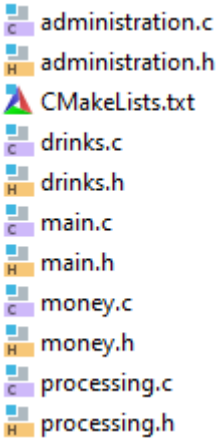
Modularity

To divide the project in multiple part, I split it in 4 different parts.

First part of the program is the processing's files. These files compute all variables needed during the whole program and save data in file. They also contain function who compute some result needed during the program.

Second and third parts is for the management of drinks and money.

Last part is for administration mode where we can find functions to add or remove drink from the list.

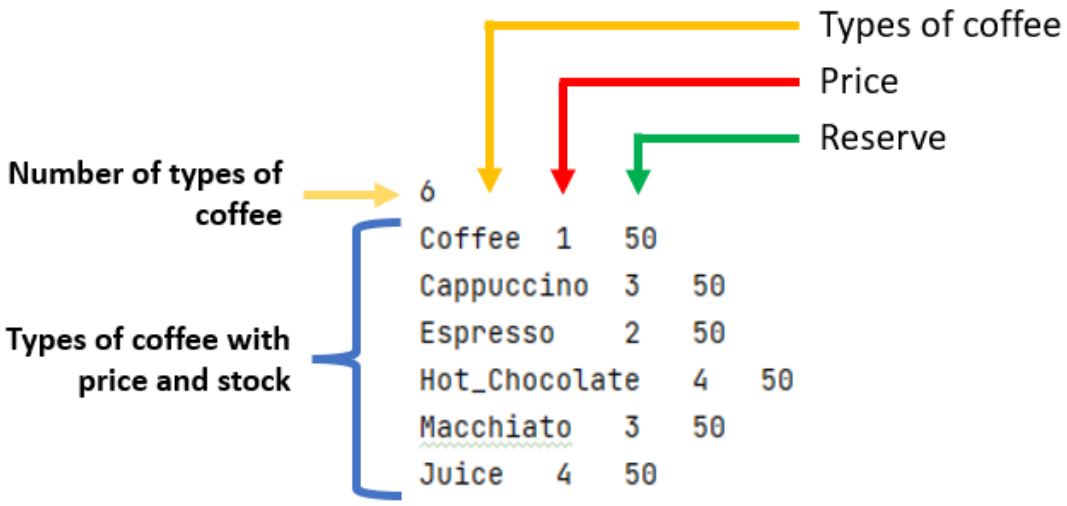


Save

In order to save data in file, I use two separate file, one for drinks and the other for money

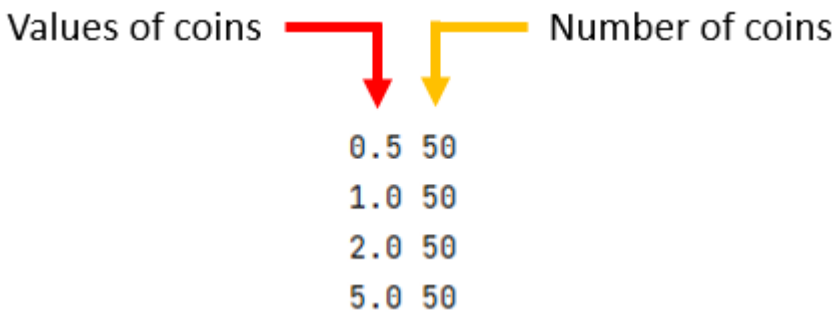
Drinks file

In the file for drinks, I store the number of types of coffee and details about them. We can see on the picture how is it stored. I save the number of types of coffee because in this program, this number can change when the user add or remove a drink.



Money file

In the file for money, I store only the value of coins and the amount for each of them.



Files in details

All parts are divided in 2 files, a source file for the definition of each functions and an header for the prototype.

Processing

In processing file there are 6 functions. Those functions are the most common part, these could be used everywhere in the program :

- Initialisation
- Print_info
- Saving
- Reset
- Parse_float
- Clear_buffer

```c
/**...*/
struct_drinks *initialisation(struct_drinks *drinks, struct_coins *coins, int *number_of_coffee);

/**...*/
void print_info();

/**...*/
void saving(struct_drinks *drinks, struct_coins coins[number_coin], int number_of_coffee);

/**...*/
void reset(struct_drinks *drinks, struct_coins *coins, int number_of_coffee);

/**...*/
float parse_float(char str[5]);

/**...*/
void clear_buffer();
```

Initialisation:

This function sets all variables needed during the program like drinks and coins. It will read files and save data into structure named drinks and coins. Both are pointers, because I need to modify it. Moreover, when the function is called, I don't know the size of drinks so I make a malloc to set the size with the number who is read in the file. This function returns NULL if a problem appears or like one of the file is missing.

Print_info:

Just print some information when the program start.

```
****************************************************************
****************************************************************
**                                                            **
**                Coffee machine emulation                    **
**                                                            **
**                        By Jules                            **
**                                                            **
****************************************************************
****************************************************************
```

Saving:

This function saves information about drinks and coins into files. It check also if the name of drinks contains a space and replace it with an underscore (_).

Reset:

This functions replace the quantity of each drinks and each coins by 50

Parse_float:

This function is pretty similar to strtof but work better then strtof in my case because it will return -1 if the function founds a letter or other characters different than a number.

```c
float parse_float(char str[5]) {
    int length = (int)strlen( Str: str);
    int i = 0;
    int error = 0;
    while (error == 0 && i < length) {
        if (str[i] == ',' || str[i] == '.') {
            str[i] = 46;
        } else if (str[i] < 48 || str[i] > 57) {
            error = 1;
        }
        i++;
    }
    if (error == 1) {
        printf( format: "Error, you can use only number\n");
        return -1;
    } else {
        return strtof( Str: str,  EndPtr: NULL);
    }
}
```

Clear_buffer:

Function that clear the input file stdin because sometimes they still have a character and most of the time it is \n

Drinks

In the drink file, I made 2 functions: drink_menu and check_drink

```
int drink_menu(struct_drinks drink[], const int nb_drink, struct_user *user) {...}

bool check_drink(struct_user user) {...}
```

Drink_menu:

This function print a menu with all possible drinks and 3 more options: administration mode (code is 1234), reset and exit. It will return 0 if nothing wrong happen and the user wants a drink, 1 if the user chooses to enter in administration mode, 2 if the user wants to reset the machine, 3 if he wants to leave and 36 (crazy number) if a problem is found during the execution. The last one should never be returned.

```
=== Choice of drinks: ===
        1 - Coffee - 1.0km
        2 - Cappuccino - 3.0km
        3 - Espresso - 2.0km
        4 - Hot Chocolate - 4.0km
        5 - Macchiato - 3.0km
        6 - Juice - 4.0km
        7 - Administration
        8 - Reset
        9 - Exit
Choice :
```

Check_drink:

This function check if the number of the drink chosen is higher than 0. It return true if all correct and false if the number equals to 0.

Money

In the file for coins there are also 2 functions: coin_add and check_change.

```
int coin_add(struct_user *user) {...}

bool check_change(struct_user user, struct_coins coins[]) {...}
```

Coin_add:

This function asks the user to enter the right amount of money for the drink chosen. It will stores each coins given and the total. It will return 0 if the user wants to cancel, 1 if user gave the right amount of money and 2 if the user gave more money than needed.

Check_change:

This function checks if the program have enough change to give back to user. If enough change is available it will return true and modify quantities of coins : add those given by the user and remove those gives back to the user. If there is no enough change it will return false.

Administration

That was not ask by the professor but in a real coffee machine you can add or remove a drink from the list, so I wanted to code this part as well as other. In this file I implement 6 functions:

- Code_check
- Administration_menu
- Enter_code
- Add_drink
- Remove_drink
- Administration

```
bool code_check(int a, int b) {...}

int administration_menu() {...}

bool enter_code() {...}

struct_drinks *add_drink(struct_drinks *drink, int *nb_type) {...}

struct_drinks *remove_drink(struct_drinks *drink, int *nb_type) {...}

struct_drinks *administration(struct_drinks *drink, int *nb_type) {...}
```

Code_check:

This function checks if a equal to b. If a equals to b it will return true, if not it will return false.

Administration_menu:

This function prints a menu for administration mode and return the choice of the user.

```
=== Administration Menu ===
        1 - Add drink
        2 - Remove drink
        3 - Leave
Choice:
```

Enter_code:
This function asks to enter the code needed to enter in the administration mode and checks if it is right or wrong by calling the function code_check. It will return true if the code is correct and false if not.

Add_drink:
This function asks the administrator to enter a new beverage and verify what he enters. Then it will use realloc to extend the size of the variable "drinks". It will return the structure of drinks with the new drink.

Remove_drink:
This function asks the administrator which drink he wants to remove and removes it from the list. It will return the structure of drinks without the drink removed.

Administration:
This function is the main part of the administration mode. It calls others functions related to administration and executes all of them. It will return the list of drinks modified.

Main.h
This file is an header file that helps. I only include one time all files in this header file and I have to include this file in each .c files. Otherwise it contains all definitions of structure.

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

#define number_coin 4

/**...*/
typedef struct {
    char name[20];
    float price;
    int number;
} struct_drinks;

/**...*/
typedef struct {
    float value;
    unsigned int number;
} struct_coins;

/**...*/
typedef struct {
    int c05;
    int c1;
    int c2;
    int c5;
} struct_given;

/**...*/
typedef struct {
    struct_drinks d;
    int type;
    float money;
    struct_given given;
} struct_user;

#include "drinks.h"
#include "money.h"
#include "processing.h"
#include "administration.h"
```

Result and upgrade

Conclusion
At the end, this is a great project to do in team, even if I did it alone, and when we are beginner in programming. My program contain about 745 lines of code.

Upgrade
The main upgrade can be to add more printf with text and extra details. We could add a function to modify the secret code to enter in administration mode

Project link

Project files:

https://replit.com/join/rpwdsdmxcw-seluj78

https://github.com/Seluj/CS103-Project

Documentation:

http://chris.ferlin.fr:83/