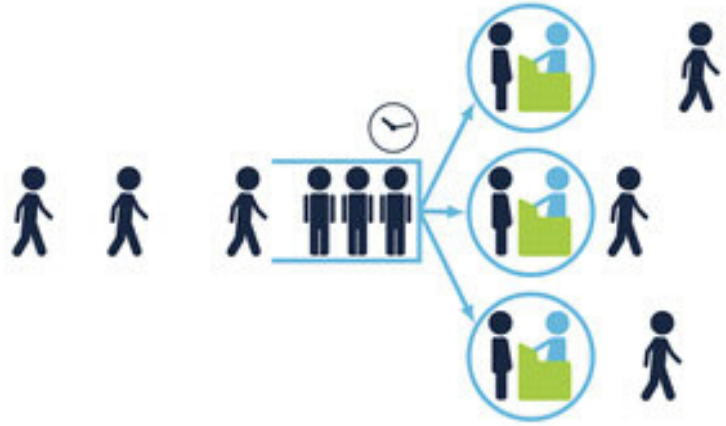
STRUCTURE DES DONNÉES

RAPPORT

IMPLÉMENTATION D'UNE FILE PAR UNE GESTION D'UNE CAISSE D'UN SUPERMARCHÉ



RÉALISÉ PAR:

- ASMA ELFAHIM
- LAILA HAMZA
- AMAL ADERDOUR

Encadré par:

- MR Yousef Es-saady

Ecole **S**upérieur de l'**E**ducation et de la **F**ormation - **A**gadir

Année scolaire: 2019 / 2020

SOMMAIRE

INTRODUCTION

Dans le cadre de notre deuxième année en **E**cole **S**upérieure de l'**E**ducation et de la **F**ormation , nous avons eu pour tâche la réalisation d'un programme en langage C.
Pour poursuivre la découverte des différentes structures de données, nous allons maintenant nous attarder sur les files.
Une file est un ensemble de valeurs qui a un début (**Début**) et une fin (**Queue**).
Le principe des files se base sur l'**enfilage** et le **défilage** en utilisant l'algorithme **FIFO**: First In First Out, qui sera mieux expliqué ci-dessous.
Pour ce faire, on a crée un exemple sur la gestion d'une caisse d'un supermarché pour implémenter la file.
La première chose à faire est de créer les structures suivantes:

- **File**
- **Maillon**
- **Client**

Ensuite, on a crée un ensemble de fonctions pour faciliter la gestion de la caisse.
Après des nombreuses analyses, nous avons abouti à un code, qui sera bien détaillé et expliqué dans les chapitres suivants.

Chapitre 1 : Créations des structures

1. Structure d'un maillon :

Pareil pour les listes chaînées, pour l'implémentation des files on aura besoin d'une structure d'un maillon. Cette dernière regroupe :

- Une **valeur** d'un élément qu'on utilisera dans l'implémentation la file
- Un **pointeur** qui permettra l'accès vers un maillon suivant

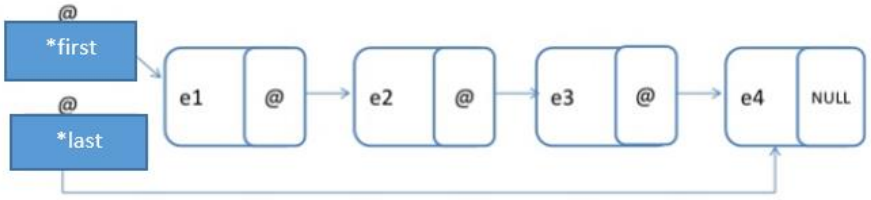
```
/* structure de maillon */
typedef struct maillon
{
    TClient value;
    struct maillon *next;
}Tmaillon;
```

2. Structure d'une file :

Afin de définir un élément de la file, le type struct sera utilisé.

Et pour avoir son contrôle, il est préférable de sauvegarder certains éléments :

- La tête : le premier élément (maillon),
- La queue : le dernier élément(maillon),
- Le nombre d'éléments



Pour réaliser cela, une structure sera utilisée. Voici sa composition :

```
/* structure de la File */
typedef struct File
{
    Tmaillon *first;
    Tmaillon *last;
    int nb_elements;
}TFile;
```

3. Structure d'un client :

Afin de pouvoir gérer une caisse d'un supermarché, on a créé un type TClient en utilisant une structure, pour l'identifier dans la file. Cette dernière rassemble des informations sur ce client, notamment :

- son nom,
- son prénom
- le montant à payer

```
/* structure de maillon */
typedef struct client
{
      char nom[25];
      char prenom[25];
      float montant_a_payer;
}TClient;
```

Les prototypes :

```
/* Prototypes */
TFile *Creer_file(void);
Bool is_empty_queue(TFile *file);
int queue_length(TFile *file);
TClient queue_first(TFile *file);
TClient queue_last(TFile *file);
void print_queue(TFile *file);
void push_queue(TFile *file,TClient x);
void pop_queue(TFile *file);
void clear_queue(TFile *file);
```

Le choix de ses fonctions sera expliqué dans le chapitre suivant.

Chapitre 2 : Les fonctions :

1. Création d'une file vide :

Comme on a défini un nouveau type TFile qui contient **une tête** et **une queue**, il suffit de créer un nouveau pointeur **pL** de ce type (TFile), lui allouer un espace dans la mémoire et préciser que le contenu des maillons (tête et queue) est nul.

Et on a initialisé le nombre d'éléments de la file par 0 (pL->nb_elements = 0).

```
TFile *Creer_file(void)
{
    TFile *pL;
    pL= malloc(sizeof(TFile));
    pL->first = NULL;
    pL->last = NULL;
    pL->nb_elements = 0;
  return pL;
}
```

2. Vérification si la file est vide :

La fonction **is_empty_queue** permet de vérifier si une file est vide ou pas. Elle doit prendre en paramètres d'entrée la file et retourner un booléen :

- **true (1)** : Si la file est vide,
- **false (0)** : sinon.
- Ce type booléen est déja défini dans une énumération « **Bool** ».

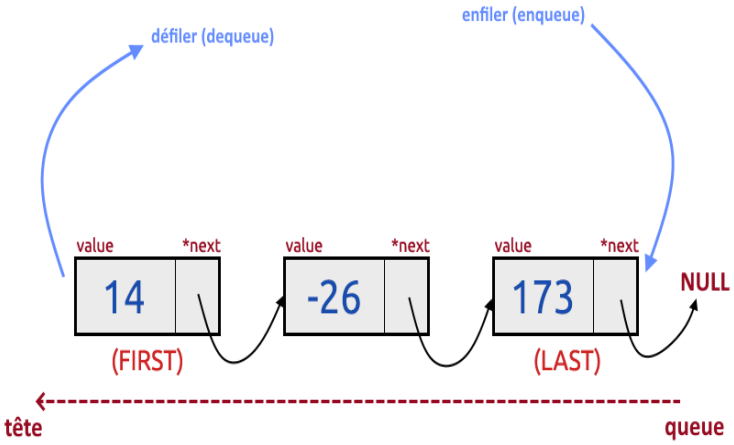
```
/* Définition du type Booléen */
typedef enum
{
    false,
    true
}Bool;
```

```
/**
* Vérifie qu'une File est vide
* retourne true si elle ne contient pas d'élements,
  false sinon
*/
Bool is_empty_queue(TFile *file)
{
    if(file->first == NULL && file->last == NULL)
        return true;
    return false;
}
```

3. Enfiler :

C'est la fonction qui ajoute un élément à la file.

La chose la plus importante qu'il faut prendre en considération est que l'insertion dans une file se fait toujours par la fin, celui-ci devient la tête de la file.



Pour ce faire :

On a créé un nouveau maillon et on lui a alloué un espace dans la mémoire, au cas où l'allocation n'est pas réussie, un message d'alerte s'affichera.

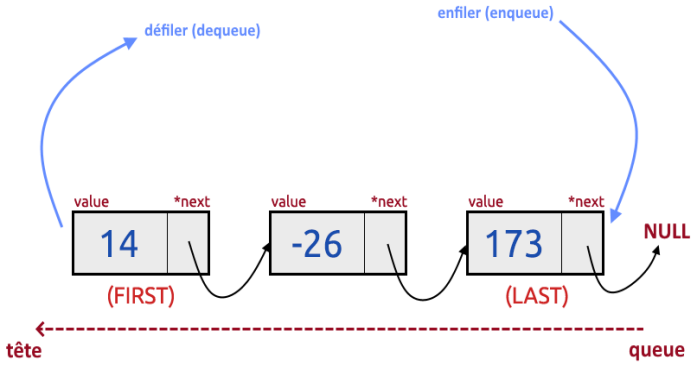
- Si la file est vide, dans ce cas on doit juste créer la file en faisant pointer **First** et **Last** vers le nouvel élément crée.
- Si la file n'est pas vide, dans ce cas il faut parcourir toute la file en partant du premier élément jusqu'à ce qu'on arrive au dernier. On rajoutera notre nouvel élément après le dernier. Celui-ci devient la nouvelle queue de la file.
- L'incrémentation **nb_element** par **1** pour dire voilà il y a un nouvel élément de plus dans la file.

```c
void push_queue(TFile *file,TClient x)
{
    Tmaillon *element;
    element = malloc(sizeof(*element));

    if(element == NULL)
    {
        fprintf(stderr, "\tErreur : probleme allocation dynamique.\n");
        exit(EXIT_FAILURE);
    }

    element->value = x;
    element->next = NULL;

    if(is_empty_queue(file))
    {
        file->first = element;
        file->last = element;
    }
    else
    {
        file->last->next = element;
        file->last = element;
    }
    file->nb_elements++;
}
```

4. Défiler :

C'est la fonction qui supprime un élément de la file.

Toujours en respectant l'algorithme FIFO, le rôle de la fonction sera de retirer l'élément situé en tête de la file.



Pour ce faire, on va suivre les étapes suivantes :

- Vérifier si la file est vide.
- Le pointeur **temp** représente l'argument de la fonction et donc l'élément à supprimer, il contient maintenant l'adresse de la tête.
- Si la file contient un seul élément, on affecte la valeur nulle aux deux maillons (**tête et queue**) : **file->first=file->last=NULL**, sinon le deuxième élément de la file devient la nouvelle tête **file->first =file->first->next.**
- Libération de l'élément temp et décrémentation **nb_element** par 1.

```
void pop_queue(TFile *file)
{
    if(is_empty_queue(file))
    {
        printf("\tRien a retirer, la File est deja vide.\n");
        return;
    }

    Tmaillon *temp = file->first;

    if(file->first == file->last)
    {
        file->first = NULL;
        file->last = NULL;
    }
    else
        file->first = file->first->next;

    free(temp);
    file->nb_elements--;
}
```

5. Afficher la file :

Afin d'afficher une file entière, il faut se positionner au début de la file.

Pour ce faire, on a créé un nouveau maillon qui pointe vers la tête de la file.

Ensuite, en utilisant le pointeur **next** de chaque élément « **temp=temp->next** » ; la file sera parcourue de **sa tête** jusqu'à **sa queue** dont le « **next=NULL** », c'est la condition d'arrêt.

```
/**
* Affiche une File
*/
void print_queue(TFile *file)
{
    if(is_empty_queue(file))
    {
        printf("\tRien a afficher, la File est vide.\n");
        return;
    }

    Tmaillon *temp = file->first;

    while(temp != NULL)
    {
        printf("\t[%s %s - Montant a payer :%.2fDHs HT]\n",temp->value.nom,
        temp->value.prenom,temp->value.montant_a_payer);
        temp = temp->next;
    }
    printf("\n");
}
```

6. Afficher la tête de la file :

Pour récupérer le **premier élément de la file**, plus précisément **sa tête**, on a créé une fonction qui aura comme paramètres d'entrées une file et qui retourne un type **TClient**.

On vérifie tout d'abord si la file est vide en utilisant la fonction **is_empty_queue**. Cela étant, le programme s'arrête, sinon, elle retourne la valeur du premier élément.

Dans ce cas, la valeur affichée est les coordonnées du premier client arrivé à la caisse.

```
/**
* Retourne la tête de la File
* retourne La valeur en début de File
*/
TClient queue_first(TFile *file)
{
    if(is_empty_queue(file))
        exit(1);

    return file->first->value;
}
```

7. Afficher la queue de la file :

Pareil pour la fonction qui retourne **la tête** de la file.

Elle prend comme paramètres d'entrée une file **(TFile *file)** et elle retourne les informations du **dernier client** inséré dans la file d'attente (c'est-à-dire **la queue**).

```
/**
* Retourne la queue de la File
* retourne La valeur en fin de File
*/
TClient queue_last(TFile *file)
{
    if(is_empty_queue(file))
        exit(1);

    return file->last->value;
}
```

8. La longueur de la file :

Pour déterminer la longueur de la file on retourne le **nombre d'éléments** déjà défini dans la structure **file**.

Voici sa composition :

```c
/**
* Retourne la longueur d'une File
* retourne Le nombre d'élements de la File
*/
int queue_length(TFile *file)
{
    return file->nb_elements;
}
```

9. Vider la file :

C'est la fonction qui supprime tous les éléments de la file.

Cette fonction a pour but de vider l'ensemble de la file. Pour ce faire, on va tester :

- ➔ Si la file est vide, on affiche un message que la file est déjà vide.
- ➔ Sinon : On supprime le premier élément de la file par la fonction **pop_queue**, jusqu'à ce que la file soit complétement vide.

```c
/**
* Nettoie la File de tous ses éléments
*/
void clear_queue(TFile *file)
{
    if(is_empty_queue(file))
    {
        printf("\tRien a nettoyer, la File est deja vide.\n");
        return;
    }

    while(!is_empty_queue(file))
        pop_queue(file);
}
```

Code principal

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Mes_Fonctions.h"


int main(void)
{

    printf ("********************GESTION D'UNE FILE D'ATTENTE D'UNE CAISSE D'UN SUPERMARCHE****************\n");
    TClient* client =malloc (sizeof (TClient));
    int choix;
    char nom[10];
    TFile *file =malloc(sizeof(TFile));
    file = Creer_file();//cree une file

    printf("Nom du caissier'e':\n");
    scanf("%s", &nom);
    printf("Bienvenue '%s' tu peux maintenant recevoir tes clients, le premier client arriv%c sur place est le premier servi!\n" , nom, 130);
    printf("\n------Effectuez un choix:------\n");

    printf("\t1- Tapez 1 pour recevoir le's' client's' !! \n");
    printf("\t2- Tapez 2 au cas ou le client a pay%c ou a quitt%c [ REMARQUE!!: Premier client arriv%c est le premier servi, il sera donc supprim%c de votre file d'attente!! ] \n",130,130,130,130);
    printf("\t3- le nombre des clients dans la file d'attente \n");
    printf("\t4- Supprimer tout les clients (NB: en cas de panne, vous pouvez annuler tout les clients!!) \n");
    printf("\t5- Afficher l'ensemble de la file d'attente de votre caisse \n");
    printf("\t6- Afficher le premier client arriv%c \n", 130 );
    printf("\t7- Afficher le dernier client arriv%c \n", 130);

    printf("\t!!ATTENTION!!: Si vous souhaitez quitter, TAPEZ 0\n");
    do{
    printf("Saisissez votre choix:\t");
    scanf("%d",&choix);
    switch(choix)
    {
        case 1:{// ajouter un client
                printf("\tNom du client:    \t");
                scanf("%s",client->nom);
```

```
                printf("\tPr%cnom du client:\t",130);
                scanf("%s",client->prenom);
                printf("\tmontant a payer :\t");
                scanf("%f",&client->montant_a_payer);
                push_queue(file,*client);
              };break;
        case 2:{//Supprimer la premier element
                *client = queue_first(file);
                printf("\n\tle premier client est: %s %s -
montant_a_payer: %.2fDH\n",client->nom ,client->prenom,client-
>montant_a_payer);
                pop_queue(file);
                printf("\tBRAVO!! Le premier client a et%c bien
servi. \n",130);
              };break;
        case 3:{// l'affichage de la longueur de la file
                 printf("\tVotre file d'attent contient %d
client's'.\n",queue_length(file));
              };break;
        case 4:{// Supprimer tout les elements de la file
                 clear_queue(file);
              };break;
        case 5:{// l'affichage de la file
                 printf(" - l'ensemble des clients en attente
dans votre caisse est: \n");
                 print_queue(file);
              };break;
        case 6:{// l'affichage de la file
               *client = queue_first(file);
                printf("\nle prochain client est: [%s %s -
montant_a_payer: %.2fDH]\n",client->nom ,client->prenom,client-
>montant_a_payer);
              };break;
        case 7:{// l'affichage de la file
                *client = queue_last(file) ;
                printf ("\nle dernier client est: [%s %s -
montant_a_payer: %.2fDH]\n",client->nom ,client->prenom,client-
>montant_a_payer);
              }; break;
     }
     printf("\n") ;
     } while (choix !=0);
    return 0 ;
}
```

Conclusion :

Le programme a été organisé en trois fichiers : « main.c », « MesFonctions.h » et « MesFonctions.c ». Ainsi qu'une petite vidéo descriptive du code. Vous trouverez, ci-joint, le code principal dans un fichier zip « Files ».

A l'aide de ce projet nous avons pu comprendre et expérimenter les différentes étapes de l'implémentation d'une file en commençant par l'analyse des différentes fonctions dont on aura besoin pour enrichir le code.

De plus la programmation nous a permis d'améliorer nos connaissances du langage C.

Vu les circonstances qu'on est entrain de vivre ces derniers temps, nous avons fait de notre mieux pour pouvoir concevoir ce programme.

Cependant, malgré les nombreuses fonctions que nous avons créées, nous ne pensons pas avoir épuisé la question, et nous espérons sincèrement susciter d'autres fonctions pour mieux maîtriser les files.