Pâtisserie

Jonathan WITT
Christopher Marshall-Breton

Présentation

L'objectif de ce projet est de simuler les différentes étapes de création d'un gâteau, de sa commande à sa dégustation.
Évidemment, c'est surtout un prétexte pour manipuler les listes chaînées, les piles et les files.

La situation est la suivante :
Un pâtissier reçoit des commandes pour des gâteaux. Il a accès à une liste de goûts, et doit créer ses gâteaux en fonction des goûts demandés par la commande.
L'ordre des goûts a une importance ! On traite ici des pièces montées.
Une fois le gâteau effectué, il l'envoie au client qui le déguste.

Etapes

Les fonctionnalités à implémenter seront les suivantes :

1. Remplir les goûts à disposition du pâtissier
2. Ajout de commandes à la file de commandes.
3. Récupération d'une commande à partir de cette file.
4. Création du gâteau à partir de la commande et de la liste de goûts.
5. Livraison du gâteau.
6. Dégustation.

Consignes de rendu :

Vous avez jusqu'à dimanche 23h55 pour le rendre.
Vous pouvez travailler en équipes de 3 ou 4.
Ajoutez bien le nom de tous les participants.
Commentez votre code.
Le dépôt est à faire sur Moodle.

Structures

Pour ce projet, les structures sont **imposées**:

```c
typedef struct Element_str{
    char texte[50];
    struct Element_str* next;
}Element_str;
```

```c
typedef struct Pile_Gouts{
    //TODO

}Pile_Gouts;
```

```c
typedef struct Gateau{
    struct Element_str* commande;
    struct Pile_Gouts* p_gouts;
}Gateau;
```

```c
typedef struct Element_gtx{
    struct Gateau* gateau;
    struct Element_gtx* next;
}Element_gtx;
```

```c
typedef struct File_Commandes{
    //TODO
}File_Commandes;
```

```c
typedef struct File_Degustation{
    //TODO
}File_Degustation;
```

Mise en place

Avant de commencer, il va falloir créer les ressources pour le pâtissier, à savoir:

1. La liste de goûts à sa disposition,
   ```c
   Element_str* l_gouts;
   ```
2. La file de commandes,
   ```c
   File_Commandes* f_commandes;
   ```
3. La file de dégustation.
   ```c
   File_Degustation* f_degustation;
   ```

Ces variables seront les seules à survivre tout au long de votre programme. Pour les autres, il faudra bien faire attention à la gestion de la mémoire.

Les goûts

```
Element_str* initialiser_gouts();
```

ou `Element_str* initialiser_gouts(int nb, ...);` pour les chauds.

Le pâtissier possède une collection de goûts parmi lesquels il va piocher pour créer ses gâteaux. Les goûts de cette collection sont **uniques**, et ne sont qu'un modèle pour en créer d'autres.

Chaque goût sera stocké dans un Element_str.

La liste des goûts à disposition doit contenir :

- "Chocolat"
- "Vanille"
- "Fraise"
- "Abricot"
- "Pomme"
- "Banane"
- "Myrtille"

La variable `l_gouts` (en rouge) devra donc contenir ceci. **L'ordre n'a pas d'importance**:

Les commandes

Le client communique avec le pâtissier par l'intermédiaire de la file de commande.

Format des commandes

Chaque commande sera représentée par une chaîne de caractères contenant les premières lettres des goûts à ajouter.

Exemple : la commande "CMVP" demande un gâteau avec les goûts Chocolat, Myrtille, Vanille, Pomme.

Le goût le plus à gauche sera le goût le plus bas dans notre gâteau.

File de commandes

```
    void passer_commande(char commande[50], File_Commandes*
f_commandes );
```

Chaque commande sera ajoutée à une liste pour être traitée par le pâtissier.
L'ordre de traitement par le pâtissier doit être le même que celui d'arrivée des commandes.
On utilise donc une file.
help : Au sein de la file, chaque commande sera stockée dans un Element_str.
Attention : au-delà de 10 commandes, on ne doit plus pouvoir ajouter de commandes.

```
    Element_str* traiter_commande(File_Commandes* f_commandes
);
```

Le pâtissier doit maintenant pouvoir récupérer une commande de cette file.
Notez que cette fonction renvoie un Element_str* et pas une chaîne de caractères.
La raison à cela est que l'on va utiliser cette commande dans le gâteau, pour servir de modèle. Inutile donc de copier la chaîne 15 fois. La commande traitée doit bien sûr être la plus ancienne de la file. Il faut également la supprimer de la file une fois traitée.
help : Pensez au cas où on essaye de traiter une commande alors que la file est vide !

Les gâteaux

```
    Gateau* creer_gateau(Element_str* commande);
```

Dès une commande traitée, le pâtissier va créer un gâteau.
Dans un premier temps, on va créer un gâteau à partir d'une commande. Celui-ci aura donc une pile de goûts vide.

```
    void construire_gateau(Gateau* gateau, Element_str*
l_gouts);
```

Ce gâteau va ensuite être construit par étages. Pour chaque lettre de la commande, il faudra chercher le goût correspondant dans la liste de goûts, et ajouter à notre gâteau un goût similaire. Chaque gâteau va alors être composé d'une commande modèle et d'une pile de goûts.
L'ordre d'ajout a donc ici son importance. Le premier goût ajouté dans la pile correspond au goût le plus bas du gâteau. Il sera mangé en dernier par le client.
help : Attention à ne pas retirer de goûts de la liste de référence du pâtissier !

La livraison

```
    void livrer(Gateau* gateau, File_Degustation*
f_degustation);
```

La livraison consiste juste à ajouter le gâteau à la file de dégustation.
Attention : Pensez à ce qu'il advient de la commande pour ce gâteau !

La dégustation

```
    void degustation(File_Degustation* f_degustation, int
nb_parts);
```

Le client va ensuite déguster les gâteaux dans l'ordre. Il va à chaque dégustation manger le nombre de parts indiquées.
Attention : si le client ne finit pas un gâteau lors d'une dégustation, il devra continuer celui-ci avant de passer au suivant. L'ordre des gâteaux a évidemment aussi son importance, d'où la file.

Assemblage Bonus

Pour les plus avancés, vous pouvez assembler le tout en faisant en sorte que des commandes aléatoires soient ajoutées, et que le client déguste un nombre de parts aléatoires à chaque fois.
Vous pouvez également jouer sur la disponibilité des goûts, et faire en sorte que le pâtissier construise ses gâteaux en plusieurs temps.