Manon HINET
Ing1 GI Groupe 2

TD noté
Programmation procédurale

Arame Ndeye Diago

CY TECH - Cergy Paris Université
Année 2022 - 2023

SOMMAIRE

I. Sujet 1

a. Structure logement et remplissage tableau de logements

La structure d'un logement est définie dans le fichier *"main.c"* :

```c
//STRUCTURES
typedef struct Logement {

        float number;
        float accomodates;
        float bedrooms;
        float bathrooms;
        float beds;
        float price;
        float minimum_nights;
        float maximum_nights;
        float number_of_reviews;
        float distance[nb_colonnes];
} logement;
```

On demande à l'utilisateur dans la fonction "*initialisationLogementx"* de saisir les caractéristiques de son logement x :

```c
void intialisationLogementx(logement* x)
{
        printf("Nous allons commencer par initialiser le logement x. Veuillez saisir les caractéristiques suivantes :\n");

        (*x).number = 0;
        for(int i=0;i<nb_colonnes;i++)
        {
                (*x).distance[i] = 0;
        }
        (*x).price = 0;
        (*x).number_of_reviews = 0;

        printf("Entrez la valeur pour 'accomodates' :\n");
        scanf("%f",&(*x).accomodates);

        printf("Entrez la valeur pour 'bedrooms' :\n");
        scanf("%f",&(*x).bedrooms);

        printf("Entrez la valeur pour 'bathrooms' :\n");
        scanf("%f",&(*x).bathrooms);

        printf("Entrez la valeur pour 'beds' :\n");
        scanf("%f",&(*x).beds);

        printf("Entrez la valeur pour 'minimum_nights' :\n");
        scanf("%f",&(*x).minimum_nights);

        printf("Entrez la valeur pour 'maximum_nights' :\n");
        scanf("%f",&(*x).maximum_nights);
}
```

Ensuite, la fonction pour lire le fichier CSV et remplir un tableau avec les logements s'appelle *"remplirTabFichierCSV"*. Tout d'abord on ouvre le fichier avec la fonction *fopen*, ensuite on parcourt le fichier ligne par ligne avec la fonction *fgets* et on récupère les chaînes de caractères entre chaque virgule. On convertit ces chaînes de caractères en type *entier* ou *float* selon la caractéristique. Enfin, on ajoute les valeurs trouvées dans le tableau de logements.

b. Calcul distance avec une seule caractéristique

J'ai choisi pour la question *a* de calculer la distance en fonction de toutes les caractéristiques entrées par l'utilisateur pour le logement x. On applique donc la formule de la distance :

```
void calculDistance(logement tab[],logement x, int nb_lignes)
{
	for(int i=0;i<nb_lignes;i++)
	{
		tab[i].distance[1] = (x.accomodates - tab[i].accomodates)*(x.accomodates - tab[i].accomodates);
		tab[i].distance[1] = sqrt(tab[i].distance[1]);

		tab[i].distance[2] = (x.bedrooms - tab[i].bedrooms)*(x.bedrooms - tab[i].bedrooms);
		tab[i].distance[2] = sqrt(tab[i].distance[2]);

		tab[i].distance[3] = (x.bathrooms - tab[i].bathrooms)*(x.bathrooms - tab[i].bathrooms);
		tab[i].distance[3] = sqrt(tab[i].distance[3]);

		tab[i].distance[4] = (x.beds - tab[i].beds)*(x.beds - tab[i].beds);
		tab[i].distance[4] = sqrt(tab[i].distance[4]);

		tab[i].distance[6] = (x.minimum_nights - tab[i].minimum_nights)*(x.minimum_nights - tab[i].minimum_nights);
		tab[i].distance[6] = sqrt(tab[i].distance[6]);

		tab[i].distance[7] = (x.maximum_nights - tab[i].maximum_nights)*(x.maximum_nights - tab[i].maximum_nights);
		tab[i].distance[7] = sqrt(tab[i].distance[7]);
	}
}
```

c. Randomiser le tableau et le trier

Une fois la distance calculée, on doit randomiser le tableau de logement dans la question *b*. Pour cela, on utilise la fonction *"randomTab"*. Dans celle-ci, on génère deux nombres aléatoires qui seront les indices des 2 logements à échanger dans le tableau, ce que l'on va faire en utilisant la fonction "*echangeTab"*.

```
//On echange les cases indiceA et indiceB
void echangeTab(logement tab[],int indiceA, int indiceB)
{
	float tmp=tab[indiceA].number;
	tab[indiceA].number=tab[indiceB].number;
	tab[indiceB].number=tmp;

	tmp=tab[indiceA].accomodates;
	tab[indiceA].accomodates=tab[indiceB].accomodates;
	tab[indiceB].accomodates=tmp;

	tmp=tab[indiceA].bedrooms;
	tab[indiceA].bedrooms=tab[indiceB].bedrooms;
	tab[indiceB].bedrooms=tmp;

	tmp=tab[indiceA].bathrooms;
	tab[indiceA].bathrooms=tab[indiceB].bathrooms;
	tab[indiceB].bathrooms=tmp;

	tmp=tab[indiceA].beds;
	tab[indiceA].beds=tab[indiceB].beds;
	tab[indiceB].beds=tmp;

	tmp=tab[indiceA].price;
	tab[indiceA].price=tab[indiceB].price;
	tab[indiceB].price=tmp;

	tmp=tab[indiceA].minimum_nights;
	tab[indiceA].minimum_nights=tab[indiceB].minimum_nights;
	tab[indiceB].minimum_nights=tmp;

	tmp=tab[indiceA].maximum_nights;
	tab[indiceA].maximum_nights=tab[indiceB].maximum_nights;
	tab[indiceB].maximum_nights=tmp;

	tmp=tab[indiceA].number_of_reviews;
	tab[indiceA].number_of_reviews=tab[indiceB].number_of_reviews;
	tab[indiceB].number_of_reviews=tmp;

	for(int i=0;i<nb_colonnes;i++)
	{
		tmp=tab[indiceA].distance[i];
		tab[indiceA].distance[i]=tab[indiceB].distance[i];
		tab[indiceB].distance[i]=tmp;
	}
}
```

```c
//on choisi 2 indices du tableau aléatoirement pour ensuite echanger ces 2 cases
void randomTab(logement tab[], int n)
{
        int indiceA = 0;
        int indiceB = 0;

        for(int i=0;i<n;i++)
        {
                indiceA = (rand() % ((n-1) + 1 - 0)) + 0;
                indiceB = (rand() % ((n-1) + 1 - 0)) + 0;
                echangeTab(tab,indiceA,indiceB);
        }
}
```

Ensuite, pour la question *b*, on trie le tableau par ordre croissant en fonction de la distance de la caractéristique que l'on souhaite. On utilise un tri simple qui est le tri par sélection, la fonction est *"selection"*.

```c
//tri par selection par odre croissant de la distance correspondant a la caracteristique choisie
void selection(logement tab[], int nb_logements, int caracteristique)
{
        int min;
        int j;
        for(int i=0;i<nb_logements-1;i++)
        {
                min=i;

                for(j=i+1;j<nb_logements;j++)
                {
                        if(tab[j].distance[caracteristique] < tab[min].distance[caracteristique])
                        {
                                min=j;
                        }
                }

                if(i!=min)
                {
                        echangeTab(tab,i,min);
                }
        }
}
```

d. Calcul du prix recommandé pour le logement x

Pour calculer le prix recommandé du logement x dans la question *c*, on va choisir un nombre k de voisins que l'on souhaite à l'aide d'un *scanf* (je l'ai testé avec plusieurs valeurs de k). Le tableau étant trié par ordre croissant en fonction de la distance (similarité), on va faire la moyenne des prix des k premiers logements du tableau. Cette partie se trouve dans la fonction *"prixMoyen".*

e. Calcul distance avec d'autres caractéristiques

Pour la question *d,* on peut choisir de relancer le programme en choisissant une nouvelle caractéristique.

f. Architecture du code et exemple d'exécution

Le programme principal est dans le fichier *"main.c"*. Ensuite, toutes les fonctions qui concernent le tableau de logements se trouvent dans *"tab.c",* celles concernant les logements dans *"logement.c"* et celles concernant le fichier csv dans *"fichier.c".* On trouve dans le fichier *"main.h"* les inclusions de bibliothèques, les variables définies, la structure *"logement"* et les prototypes des fonctions.
Voici un exemple d'exécution du programme :

```
cytech@manon-hinet-student-laptop:~/Desktop/Ing1/Prog/Projet 1/06-12-2/Exercice1$ ./prog
Nous allons commencer par initialiser le logement x. Veuillez saisir les caractéristiques suivantes :
Entrez la valeur pour 'accomodates' :
10
Entrez la valeur pour 'bedrooms' :
5
Entrez la valeur pour 'bathrooms' :
6
Entrez la valeur pour 'beds' :
2
Entrez la valeur pour 'minimum_nights' :
4
Entrez la valeur pour 'maximum_nights' :
50

--------------------------------------------------------------------------------------------------------------------------------

Veuillez choisir une caractéristique : 1. Accomodates | 2. Bedrooms | 3. Bathrooms | 4. Beds | 5. Minimum nights | 6. Maximum nights
1

Avec combien de logements voulez-vous comparer le logement x ?
5

Les 5 logements les plus similaires au logement x sont les logements avec les numéros :
1811  5757  5029  2225  985
Le prix recommandé pour le logement x d'après la moyenne est de 423.60 €

--------------------------------------------------------------------------------------------------------------------------------

Voulez-vous tester le programme avec une autre caractéristiques ? 1. oui 2 non
1

--------------------------------------------------------------------------------------------------------------------------------

Veuillez choisir une caractéristique : 1. Accomodates | 2. Bedrooms | 3. Bathrooms | 4. Beds | 5. Minimum nights | 6. Maximum nights
2

Avec combien de logements voulez-vous comparer le logement x ?
10

Les 10 logements les plus similaires au logement x sont les logements avec les numéros :
2225  2017  1186  2518  5232  5713  1062  2394  2725  2272
Le prix recommandé pour le logement x d'après la moyenne est de 369.70 €

--------------------------------------------------------------------------------------------------------------------------------

Voulez-vous tester le programme avec une autre caractéristiques ? 1. oui 2 non
2
```

II. Sujet 2

a. Structure processus et file

On suppose pour toutes les techniques que les processus arrivent tous au même moment. La structure d'un processus comprend un caractere (le nom du processus), une durée d'exécution et un numéro de priorité :

```c
typedef struct Processus
{
        char nom;
        float duree_exec;
        int priorite;
} processus;
```

Ensuite, on utilise la structure d'une file que l'on a vu en cycle pré-ingénieur, en mettant comme élément de la file un processus.

```
typedef struct ordonnanceur
{
        processus val;
        struct ordonnanceur * suiv;
} File;


typedef File * PtrFile;
```

b. FCFS

Pour cette technique d'ordonnancement non-préemptif, on commence par demander à l'utilisateur le nombre de processus à ajouter à l'ordonnanceur. Ensuite, on génère aléatoirement le temps d'exécution de ces processus (entre 1 et 10 secondes) et l'on attribue le numéro de priorité le plus petit au premier qui arrive (on ne s'intéressera pas aux numéros de priorité pour cette technique).
Une fois tous les processus ajoutés au fur et à mesure qu'ils sont générés, on attend le nombre de secondes du premier processus arrivé puis on le supprime de la file, ainsi de suite tant que la file n'est pas vide.
Voir les fonctions dans le fichier *FCFS.c.*

c. SJF

Pour cette seconde technique d'ordonnancement non-préemptif, on commence par demander à l'utilisateur le nombre de processus à ajouter à l'ordonnanceur. Ensuite, on génère aléatoirement le temps d'exécution de ces processus (entre 1 et 10 secondes) et l'on attribue le numéro de priorité le plus petit au premier qui arrive (on ne s'intéressera pas aux numéros de priorité pour cette technique), comme pour la première technique.
Une fois un processus créé, on l'ajoute dans la file par ordre croissant du temps d'exécution. Comme précédemment, lorsque tous les processus ont été ajoutés, on attend le nombre de secondes du premier processus dans la file (celui avec le plus petit temps d'exécution) puis on le supprime de la file, ainsi de suite tant que la file n'est pas vide.
Voir les fonctions dans le fichier *SJFc.*

d. Priorité

Pour cette troisième technique d'ordonnancement non-préemptif, on commence par demander à l'utilisateur le nombre de processus à ajouter à l'ordonnanceur. Ensuite, on génère aléatoirement le temps d'exécution de ces processus (entre 1 et 10 secondes) et contrairement aux deux premières techniques, on attribue le numéro de priorité aléatoirement, en vérifiant que le numéro de priorité n'est pas déjà attribué à un autre processus dans la file.
Une fois un processus créé, on l'ajoute dans la file par ordre croissant du numéro de priorité. Enfin, lorsque tous les processus ont été ajoutés, on attend le nombre de secondes du premier processus dans la file (celui avec le plus petit temps numéro de priorité) puis on le supprime de la file, ainsi de suite tant que la file n'est pas vide.
Voir les fonctions dans le fichier *priorite.c.*

e. Fonction run et step

La fonction *step* défile et attend le nombre la durée d'exécution du processus en tête de file (qui est en train d'être exécuté).
La fonction *run* exécuter la fonction step tant que la file n'est pas vide.
Elles se trouvent dans le fichier *ordonnanceur.c.*

```c
PtrFile step(PtrFile file)
{
	PtrFile p = file;
	if(file != NULL)
	{
		printf("\nVoici la file de processus :\n");
		afficher_file(file);
		printf("\n");

		//tant que la duree d'execution du processus en train d'etre execute est positive on continue l'execution
		while(file->val.duree_exec > 0)
		{
			//on afficher un decompte du temps d'execution du processus en cours
			printf("[%c] : %.0f\n",file->val.nom,file->val.duree_exec);
			file->val.duree_exec--;

			//on attend 1 seconde
			sleep(1);
		}

		//on supprime le processus en tete de la file
		file = file->suiv;
		free(p);
	}

	return file;
}

void run(PtrFile file)
{
	//tant que la file n'est pas vide on effectue un tour d'ordonnancement
	while(file != NULL)
	{
		file = step(file);
	}
}
```

f. Exemple d'exécution

```
cytech@manon-hinet-student-laptop:~/Desktop/Ing1/Prog/Projet 1/06-12-2/Exercice2$ ./prog
Veuillez choisir une technique d'ordonnancement non-préemptif : 1. FCFS 2. SJF 3. Priorité
3
Combien de processus voulez-vous ajouter ?
3

Voici la file de processus :
[b : 5s | 1] -> [c : 6s | 2] -> [a : 9s | 3] ->

[b] : 5
[b] : 4
[b] : 3
[b] : 2
[b] : 1

Voici la file de processus :
[c : 6s | 2] -> [a : 9s | 3] ->

[c] : 6
[c] : 5
[c] : 4
[c] : 3
[c] : 2
[c] : 1

Voici la file de processus :
[a : 9s | 3] ->

[a] : 9
[a] : 8
[a] : 7
[a] : 6
[a] : 5
[a] : 4
[a] : 3
[a] : 2
[a] : 1
```