Exercice 3 :

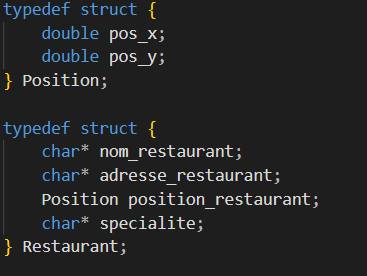
Deux points à préciser pour cet exercice :

- Nous avons adapté le format de certaines fonctions (arguments, valeurs de retour) pour que ce soit plus optimisé et plus propre (par exemple pour éviter une allocation des tableaux avec une taille aléatoire par exemple).   
En particulier, on redemande le chemin du fichier dans chaque fonction pour éviter que ça soit défini comme une variable globale ou que ça soit écrit en dur dans le code.

- A noter que la gestion d’erreur n’est pas entièrement prise en charge. Ce qui signifie que si on essaie d’entrer des valeurs qui n’ont pas été conçues pour fonctionner, le programme risque de réagir de manière imprédictible au lieu d’afficher une erreur.   
De même si on entre des entrés utilisateur trop grandes (car les tableaux ont des tailles fixes et la gestion infinie demande plus de complexité). Il ne faut donc pas s’attendre à ce que la gestion d’erreur soit totalement fonctionnelle.

## Question 1

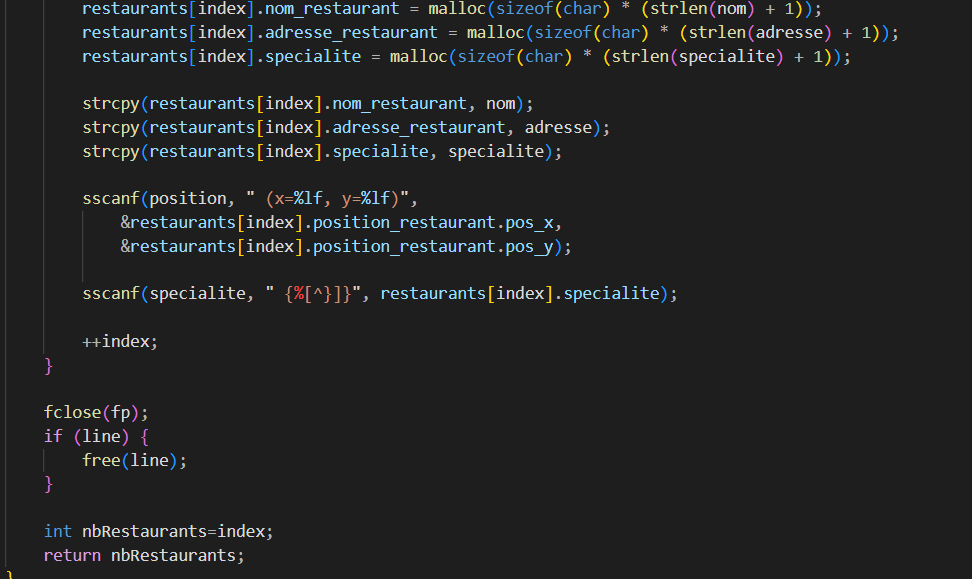
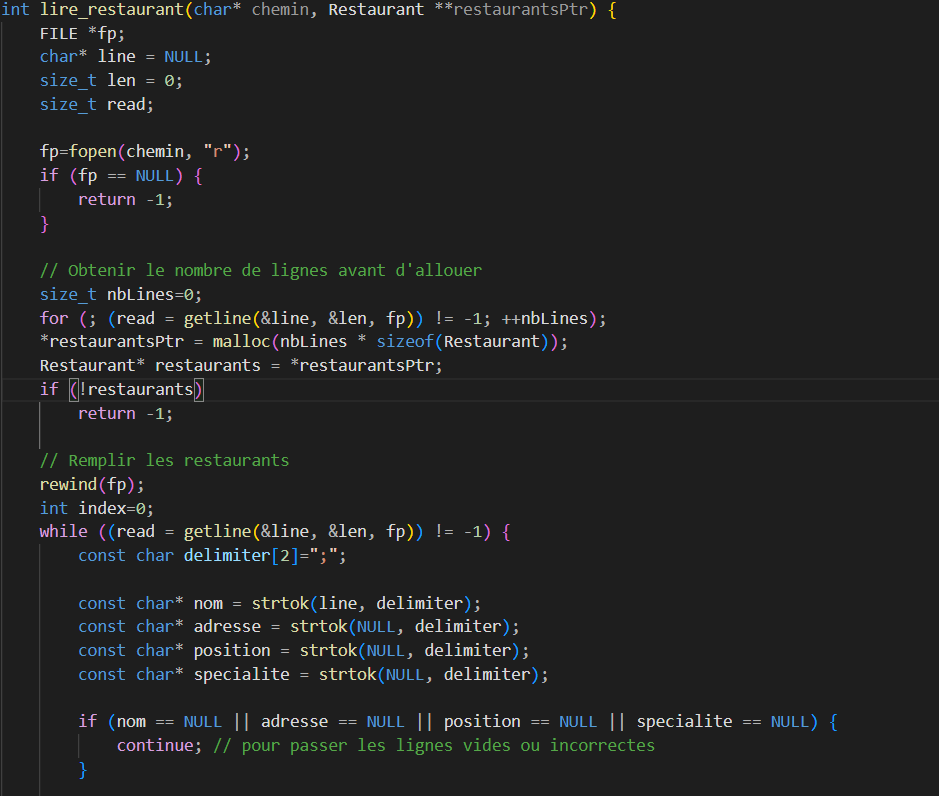
Nous utiliserons ces deux structures pour représenter les restaurants :



## Question 2 :

Voici la fonction lire\_restaurant qui lit chaque ligne du fichier et utilise strtok pour split les valeurs avec le delimiter ‘ ;’. Nous avons fait en sorte que les lignes vides ou incorrectes ne soient pas pris en compte. Les positions seront retrouvées et placés dans la structure Position de position\_restaurant.

Comme précisé au début de l’exercice, nous avons modifié le format de la fonction pour qu’il y ai en argument un pointeur sur le tableau de restaurauts et pas juste le tableau. Cela permet de pouvoir allouer la mémoire du tableau à l’intérieur du lire\_restaurant et non pas en dehors (ce qui serait très peu pratique parce qu’on ne peut pas savoir le nombre de restaurants à l’avance).



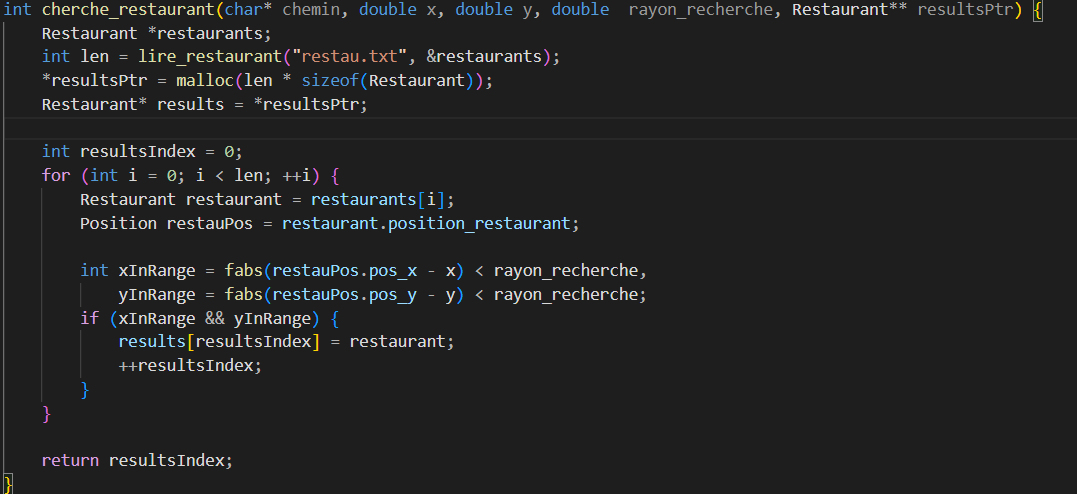
## Question 3 :

Voici la fonction d’insertion qui fonctionne de manière simple, en ajoutant un \n au début au lieu de l’ajouter à la fin, au cas où il ne serait pas déjà inséré (car il faut absolument que le restaurant soit placé sur une nouvelle ligne).



## Question 4 :

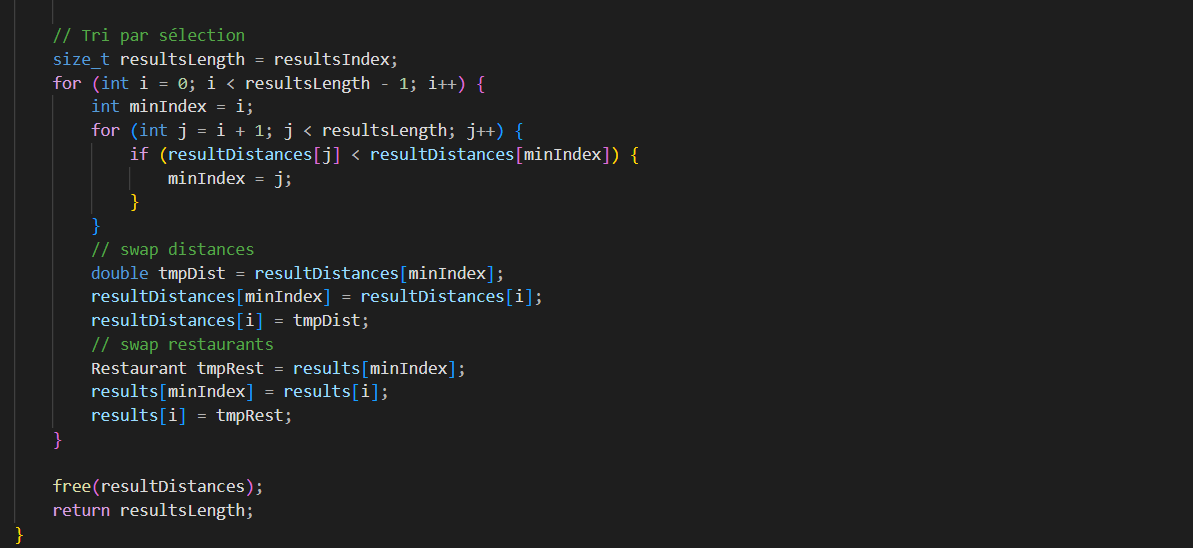
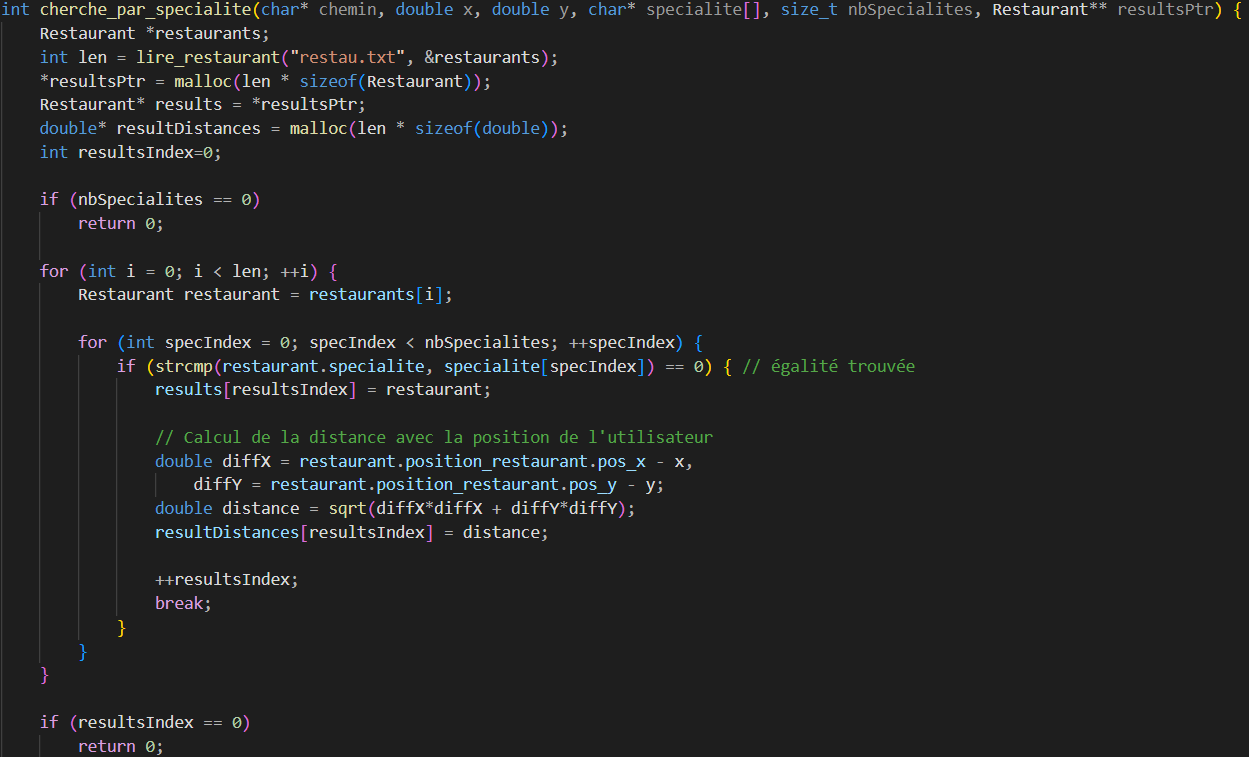
Pour cette fonction cherche\_restaurant, on parcourt chaque restaurant et à chaque fois que la position est dans la portée, alors on l’ajoute au tableau des résultats. Pour vérifier la portée, on doit vérifier sur X et Y et on a besoin de la méthode fabs de math.h qui calcule la valeur absolue pour les nombre flottants (double).



## Question 5 :

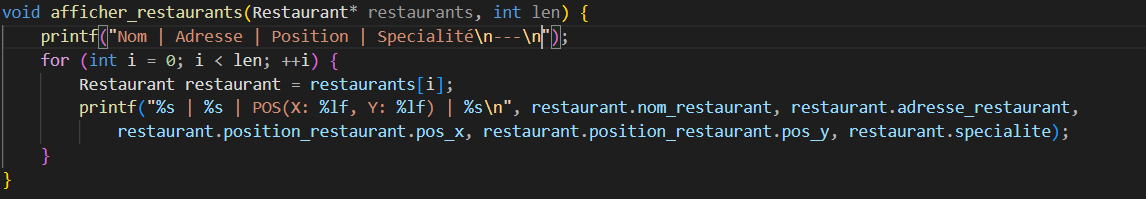
Nous avons ajouté l’argument chemin mais aussi l’argument « nbSpecialites » qui est nécessaire de spécifier parce que sinon on ne pourra pas correctement itérer sur le tableau des spécialités (on ne peut pas déterminer la taille tableau passé en argument en tant que pointeur).

La fonction se divise en deux parties, d’abord la recherche des restaurants, puis un tri est effectué pour ranger les restaurants en fonction de la distance (on utilise ici le principe distance = sqrt(x²+y²)).



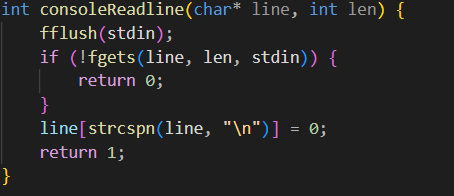
## Question 6 :

Nous avons ajouté une fonction afficher\_restaurants () pour empêcher de devoir réecrire la même chose à chaque fois (il faut afficher les restaurants pour beaucoup de choix de fonctions).



Nous avons également ajouté certaines fonctions utiles, tel que prompt\_position() pour demander la position à l’utilisateur, prompt\_specialites pour demander les spécialités (nécessaire pour le choix 4).



Ainsi que la fonction consoleReadline qui permet de demander un input de chaine de caractère à l’utilisateur (parce qu’en C c’est un peu plus compliqué que dans la plupart des langages plus haut niveau) :  


Enfin, voici le code de la fonction main (voir le fichier .c pour plus de visibilité).