KEBLI Soulaymane
CHAPIRO Louka

SR01- Rapport : Devoir 1

Exercice 1

Question 1 :

```
int main(){
  int A=20,   B=5;
  int C=!--A/++!B;
  printf("A=%d B=%d c=%d \n", A,B,C);
}
```

Résultat : Ne fonctionne pas

Explication :

L'opérateur ! (not) renvoie un booléen 0 ou 1. Cependant, l'opération ++ tente d'incrémenter un booléen ce qui est impossible.
Voilà pourquoi il n'est pas possible de compiler ce programme.

Question 2 :

```
int main(){
  int A=20,   B=5,   C=-10,   D=2;
  printf("%d \n", A&&B||!0&&C++&&!D++);
  printf("c=%d d=%d \n", C, D);
}
```

Résultat : 1
c=-10 d=2

Explication :

A && B -> donne 1 car A et B sont différents de 0 (&& prioritaire). Ensuite, il y a un OU logique entre (A && B) et (!0 && C++ && !D++). Cependant, (A && B) = 1 donc il n'y a pas besoin de calculer la partie de droite du OU logique car le OU est forcément égal à 1 si l'une des deux parties est égale à 1.

C'est pour cela que les valeurs des variables C et D ne changent pas.

Question 3 :

```
int main(){
   int p[4]={1,-2,3,4};
   int *q=p;
   printf("c=%d\n",  *++q**q++);
   printf("c=%d \n",*q);
}
```

Résultat : c=4
c=3

Explication :
On définit un pointeur q = p ; Ainsi q pointe vers la première valeur du tableau p, c'est à dire 1

Ensuite pour (*++q * *q++) :

*++q : q est incrémenté, q étant une adresse, q désigne l'adresse de la deuxième valeur du tableau p.
L'étoile indique la valeur pointée par q c'est à dire -2

Pour *q++, *q pointe donc vers -2, puis à la fin de l'opération l'adresse q sera incrémentée de 1 et pointera vers 3.

On a donc une multiplication de -2 par -2 donnant 4

Puis comme expliqué, à la fin de l'opération q pointe vers 3 donc *p = 3

Question 4 :

```
int main(){
   int p[4]={1,-2,3,4};
   int *q=p;
   int d=*q&*q++|*q++;
   printf("d=%d\n",  d);
   printf("q=%d \n",*q);
}
```

Résultat : d=-1
q=3
Explication :
p et q sont définis de manière similaire à l'exercice précédent
d = *q & *q++ | *q++
Priorité sur le & par rapport au |
*q pointe vers 1 donc *q & *q++ = 1 et l'adresse q sera incrémenté pour l'opération suivante
Ensuite, on a 1 | *q++
q pointe désormais vers -2, q sera incrémenté après l'opération

on a donc 1 | -2

2 en binaire : 000....010 -> complément à 2 : 111....101 -> +1 : 111....110 = -2
000....001 | 111....110 = 111....111 -> complément à 2 : 000....000 -> +1 : 000....001 = 1
On a ainsi d = -1 et q pointe vers 3 donc q* = 3

Question 5 :

```c
int q5(){
    int a=-8 , b =3;
    int c = ++a && --b ? b-- : a++;
    printf ( format: " a=%d b=%d c=%d \n",a,b, c);
}
```

Résultat : a=-7 b=1 c=2

Explication :

On a : ++a && --b ? b-- : a++
Par priorité :
++a -> a = -7 préfixe donc se fait avant le &&
--b -> b = 2

-7 && 2 = 1 car -7 et 2 sont différents de 0

Comme ++a && --b = 1, b-- est réalisé.
Donc c = 2, puis b est décrémenté, b = 1
On a ainsi a = -7 ; b = 1 et c = 2

Question 6 :

```c
int main(){
   int   a=-8   ,   b   =3;
   a    >    >=2^b;
   printf("a=%d\n",a);
}
```

Résultat : a=-4
Explication :
On a : a>>= 2^b
L'opération en priorité est 2^b avec ^ représentant le XOR. On a donc 2 XOR 3.
010 ^ 011 = 1
On obtient donc a >>= 1 ce qui correspond à un décalage à droite de 1 bit.
8 en binaire : 000....1000 -> complément à 2 : 111....0111 -> +1 : 111....1000 = -8
Décalage de 1 bit à droite : 111....100 -> complément à 2 : 000....011 -> +1 : 000....100 = 4
donc a = -4

Exercice 2

Les commentaires ont été ajouté au code après le rapport, ils n'apparaissent donc pas sur les captures d'écran

Question 1 :

```c
void note_sr01(int POINTS[N]){
    printf( format: "Entrez la 1er note :");
    scanf( format: "%d",POINTS);
    for (int i = 1; i < N; ++i) {
        printf( format: "Entrez la %d-eme note :",i+1);
        scanf( format: "%d",POINTS+i);
    }
    return;
}
```

Cette fonction récupère un tableau d'entier en argument, et le remplit avec les données rentrées par l'utilisateur. Pour cela, on utilise la fonction scanf pour récupérer les entrées, et une boucle for sur la taille N du tableau POINTS. La fonction scanf nécessite l'adresse où la valeur doit être placée, c'est pour cela que nous utilisons (POINTS + i), correspondant à l'adresse du ième élément du tableau.

Question 2 :

```c
int note_max(int tab[N]){
    int max =0;
    for (int i = 0; i < N; i++ ){
        if (tab[i] > max){
            max = tab[i];
        }
    }
    return max;
}
```

```c
int note_min(int tab[N]){
    int min =60;
    for (int i = 0; i < N; i++ ){
        if (tab[i] < min){
            min = tab[i];
        }
    }
    return min;
}
```

```c
float moyenne_note(int tab[N]){
    float moyenne, somme = 0;
    for (int i = 0; i < N; i++ ){
        somme += tab[i];
    }
    moyenne = somme/N;
    return moyenne;
}
```

La fonction note_max, retourne la plus grande valeur du tableau POINTS. Une variable max est initialisée à 0 (la note minimale). En parcourant le tableau avec une boucle for, s'il existe une valeur supérieure à max, elle devient le max. Ainsi, à la fin de la boucle, max contient la plus grande valeur du tableau.

La fonction note_min, retourne la plus petite valeur du tableau POINTS. Une variable min est initialisée à 60 (la note maximale). En parcourant le tableau avec une boucle for, s'il existe une valeur inférieure à min, elle devient le min. Ainsi, à la fin de la boucle, min contient la plus petite valeur du tableau.

La fonction moyenne_note retourne la moyenne des valeurs du tableau POINTS. On commence par additionner toutes les valeurs du tableau à l'aide d'une boucle for. Ensuite, on divise la somme par N pour obtenir la moyenne que l'on retourne. Il faut faire attention à utiliser des floats pour la moyenne.

Question 3 :

```
void init_notes(int notes[7],int points[N]){
    for (int k = 0; k < 7; k++){ // Initialiser chaq
        notes[k] = 0;
    }
    for (int i = 0; i < N; i++){ // Incrémentation d
        if(points[i] <= 9) notes[0]++;
        else if (points[i] <= 19) notes[1]++;
        else if (points[i] <= 29) notes[2]++;
        else if (points[i] <= 39) notes[3]++;
        else if (points[i] <= 49) notes[4]++;
        else if (points[i] <= 59) notes[5]++;
        else notes[6]++;
    }
}
```

Cette fonction permet d'initialiser le tableau NOTES passer en paramètre, en fonction du tableau POINTS. Pour cela, chaque valeur du tableau NOTES est passée à 0 afin qu'elles puissent être incrémentées plus tard.
Ensuite, une boucle for parcourt le tableau POINTS, en fonction de l'intervalle de valeurs, un rang de NOTES sera incrémenté.

Question 4 :

```
int max_notes(int tab[7]){
    int max =0;
    for (int i = 0; i < 7; i++ ){
        if (tab[i] > max){
            max = tab[i];
        }
    }
    return max;
}
```

La fonction max_notes permet de déterminer la valeur MAXN afin de réaliser les graphiques. C'est-à-dire la valeur maximale du tableau NOTES.

```c
void nuage(int notes[7]){
    int maxn = max_notes( tab: notes);
    for (int i = maxn; i > 0 ; i--){
        printf( format: "%d >",i);
        printf( format: "\t");
        for(int j = 0; j <7; j++){
            printf( format: "    ");
            if(notes[j] == i) printf( format: "o");
            else printf( format: " ");
            printf( format: "   ");
        }
        printf( format: "\n");
    }
    printf( format: "\t");
    for(int k = 0; k < 7; k++){
        printf( format: "+---");
        if(notes[k] == 0) printf( format: "o");
        else printf( format: "-");
        printf( format: "---");
    }
    printf( format: "+");
    printf( format: "\n");
    printf( format: "\t");
    printf( format: "  |0-9|\t|10-19|\t|20-29|\t|30-39|\t|40-49|\t|50-59|\t| 60 |\n");

}
```

La fonction nuage permet de réaliser un graphique en nuage de points. On utilise MAXN afin de déterminer la hauteur de graphique. Ensuite, pour chaque étage, si la NOTES contient un entier correspondant à la valeur de l'étage, on ajoute un 'o' au niveau de l'intervalle. Pour cela plusieurs boucle for sont utilisées.

Question 5 :

```c
void baton(int notes[7]){
    int maxn = max_notes( tab: notes);
    for (int i = maxn; i > 0 ; i--){
        printf( format: "%d >",i);
        printf( format: "\t");
        printf( format: " ");
        for(int j = 0; j <7; j++){

            if(notes[j] >= i) printf( format: "#######");
            else printf( format: "       ");
            printf( format: " ");
        }
        printf( format: "\n");
    }
    printf( format: "\t");
    for(int k = 0; k < 7; k++){
        printf( format: "+-------");
    }
    printf( format: "+");
    printf( format: "\n");
    printf( format: "\t");
    printf( format: "  |0-9|\t|10-19|\t|20-29|\t|30-39|\t|40-49|\t|50-59|\t| 60 |\n");

}
```

Le fonctionnement de la fonction bâton est similaire à la fonction nuage. La seule différence est qu'il faut continuer à afficher #### lorsque la valeur dans NOTES est supérieure à l'étage. Comme cela, on a un bâton qui descend jusqu'à l'étage 0 du graphique.

```c
int main() {
    int points[N], notes[7];
    note_sr01( POINTS: points);
    printf( format: "La note maximale est : %d \n La note minimale est : %d \n La moyenne est : %f\n", note_max( tab: points), not
           moyenne_note( tab: points));
    init_notes(notes,points);

    printf( format: "\n");
    nuage(notes);
    printf( format: "\n");
    baton(notes);
    return 0;
}
```

Nous avons créé un main permettant d'utiliser toutes les fonctions de l'exercice. L'utilisateur doit entrer dans un premier temps les valeurs des notes. Le max, le min et la moyenne sont ensuite affichés. Le tableau NOTES est ensuite initialisé et les deux graphiques sont affichés. Il est donc facile de tester le bon fonctionnement des fonctions.

Exercice 3

L'objectif de cet exercice est de collecter un ensemble de données relatives à des restaurants et de les manipuler par la suite. Les informations concernant chaque restaurant sont stockées dans un fichier (restau.txt) selon le format donné dans la figure 2 du sujet.

```
Restaurant; adresse; coordonée; spécialité

Edern; 6, rue Arsène Houssaye - Paris 8ème;(x=1.5, y=44.8); {Cuisine gastronomique};
La Farnesina; 9, rue Boissy d'Anglas - Paris 8ème;(x=70.5, y=74.12578); {Cuisine italienne};
41 Penthièvre; 41 Penthièvre 41, rue de Penthièvre - Paris 8ème; (x=40.5, y=77.12578); {Cuisine traditionnelle française};
6 New York; 6, avenue de New York - Paris 16ème; (x=12.5, y=74.12578); {Cuisine gastronomique};
African Lounge; 20 bis, rue Jean Giraudoux - Paris 16ème; (x=18.5, y=7.578); {Cuisine africaine};
Le 122; 122, rue de Grenelle - Paris 7ème; (x=48.8571, y=2.319250000000011); {Cuisine gastronomique};
Le 404;69, rue des Gravilliers - Paris 3ème;(x=48.8645, y=2.3542); {Cuisine marocaine};
Le 41 Pasteur; 41, boulevard Pasteur - Paris 15ème; (x=48.8426, y=2.3132699999999886); {Bistrot};
Belle Armée; 3, avenue de la Grande Armée - Paris 16ème;(x=2.5, y=12.578); {Brasserie};
Benkay; 61, quai de Grenelle - Paris 15ème;(x=142.5, y=7.1); {Cuisine japonaise};
Agapé; 51, rue Jouffroy-d'Abbans - Paris 17ème;(x=128.5, y=200.178); {Haute cuisine française};
```

Figure 2: Exemple d'un fichier de données

Question 1 - Structure :

```c
//Structure pour définir la position de chaque restaurant
typedef struct Position{
    double x;
    double y;
}Position;

typedef Position *T_position;

//structure pour stocker dans un restaurant son nom, son adresse, sa position, et sa ou ses spécialités.
typedef struct Restaurant{
    char *nom_restaurant;
    char *adresse_restaurant;
    T_position position_restaurant;
    char *specialite;
}Restaurant;

// Structure pour stocker un restaurant et sa distance par rapport à la position actuelle.
typedef struct RestaurantDistance{
    Restaurant restaurant;
    double distance;
} RestaurantDistance;
```

Dans cette question l'objectif est de déclarer une structure Restaurant conforme pour la suite du devoir. Nous avons décidé de concevoir plusieurs structures différente afin d'optimiser nos algorithmes et notre travail mais aussi de faciliter la compréhension d'un quelconque lecteur.

Ces structures sont utilisées pour représenter des informations sur des restaurants, en particulier leur emplacement, leur nom, leur adresse, leurs spécialités, ainsi que leur distance par rapport à une position donnée. Nous avons donc déclarer trois structures : Position, Restaurant, RestaurantDistance :

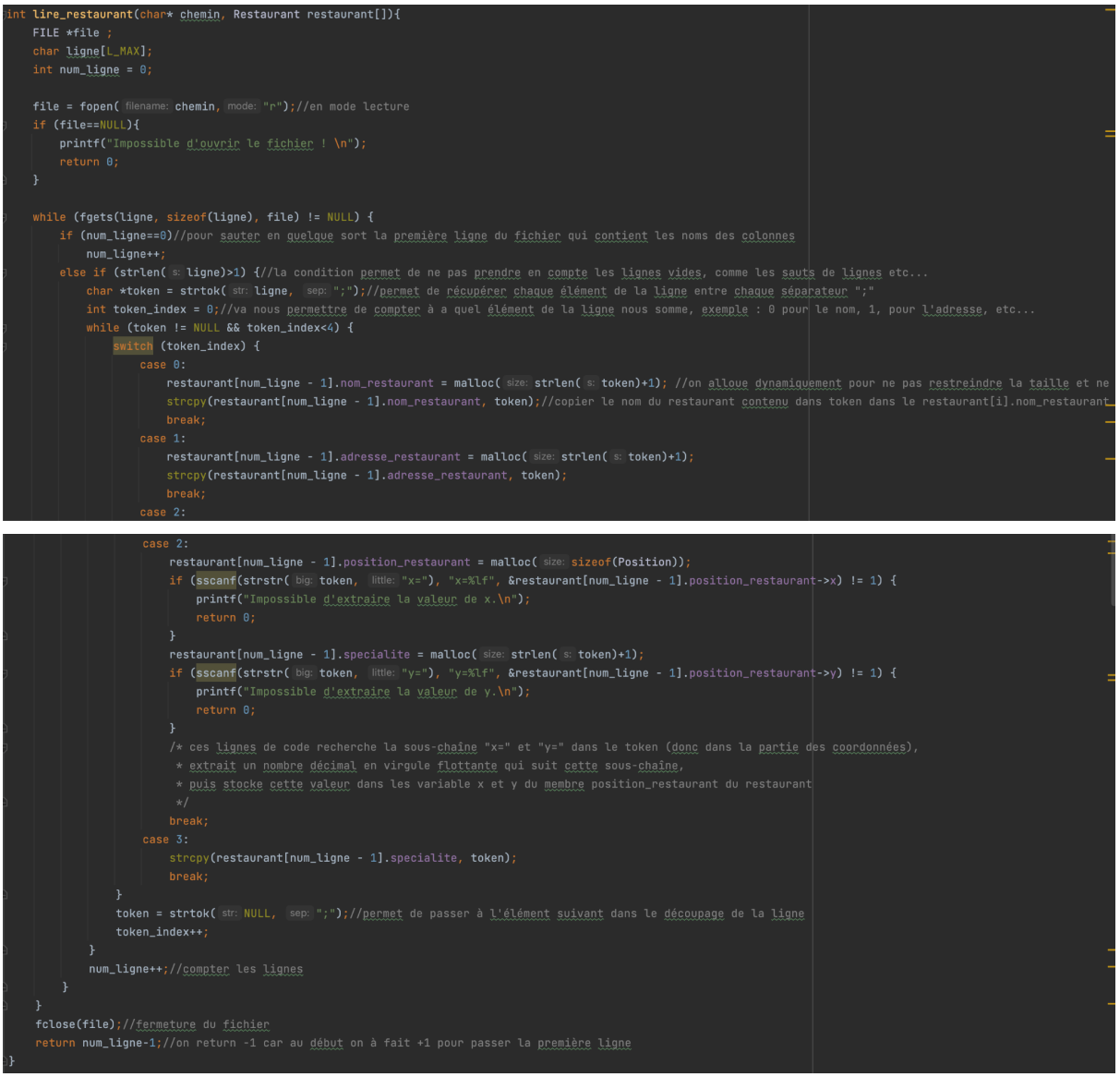
1. Position :
 - Cette structure représente la position d'un restaurant en termes de coordonnées x et y.
 - Elle est utilisée pour stocker les coordonnées géographiques de l'emplacement d'un restaurant.

2. Restaurant :
- Cette structure représente un restaurant avec plusieurs attributs :
    - *nom_restaurant* : Le nom du restaurant (stocké en tant que chaîne de caractères).
    - *adresse_restaurant* : L'adresse du restaurant (stockée en tant que chaîne de caractères).
    - *position_restaurant* : L'emplacement du restaurant, qui est un pointeur vers une structure de type Position (coordonnées géographiques).
    - *specialite* : Les spécialités du restaurant (stockées en tant que chaîne de caractères).
- Cette structure est utilisée pour stocker les informations détaillées sur un restaurant.

3. RestaurantDistance :
 - Cette structure est utilisée pour associer un restaurant à sa distance par rapport à une position actuelle.
 - Elle contient deux membres :
    - *restaurant* : Une structure de type Restaurant qui stocke les informations sur le restaurant lui-même.
    - *distance* : Un nombre à virgule flottante (double) représentant la distance entre le restaurant et une position donnée.
 - Cette structure est utile lorsque pour trier ou organiser une liste de restaurants en fonction de leur proximité par rapport à une position spécifique.

Ainsi, ces structures permettent de stocker des informations sur les restaurants et de les organiser en fonction de leur position géographique et de leur distance par rapport à un point de référence.

Question 2 - Lecture :

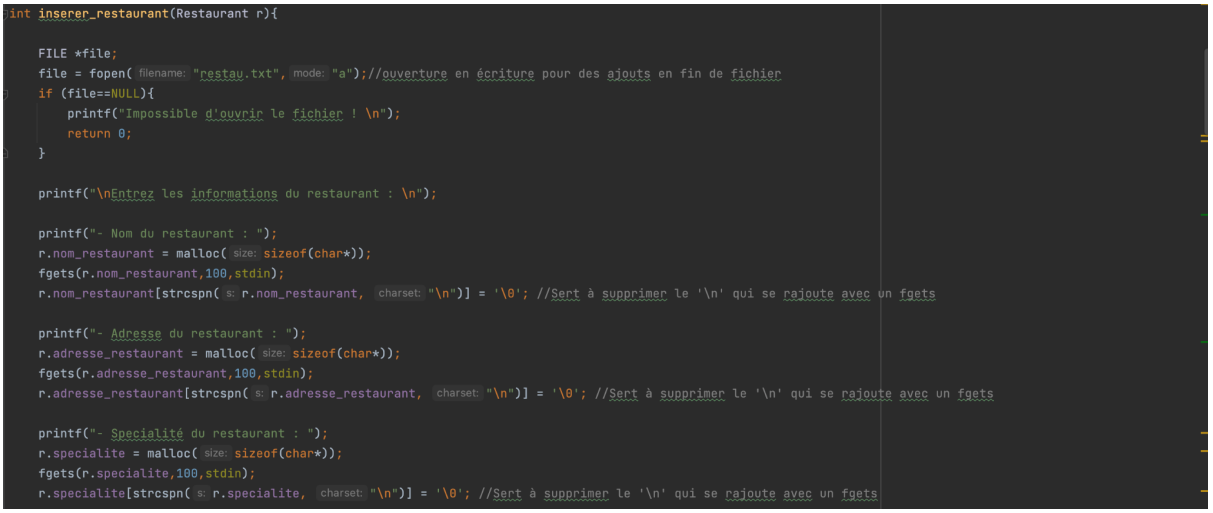


```c
int lire_restaurant(char* chemin, Restaurant restaurant[]){
    FILE *file ;
    char ligne[L_MAX];
    int num_ligne = 0;

    file = fopen( filename: chemin, mode: "r");//en mode lecture
    if (file==NULL){
        printf("Impossible d'ouvrir le fichier ! \n");
        return 0;
    }

    while (fgets(ligne, sizeof(ligne), file) != NULL) {
        if (num_ligne==0)//pour sauter en quelque sort la première ligne du fichier qui contient les noms des colonnes
            num_ligne++;
        else if (strlen( s: ligne)>1) {//la condition permet de ne pas prendre en compte les lignes vides, comme les sauts de lignes etc...
            char *token = strtok( str: ligne,  sep: ";");//permet de récupérer chaque élément de la ligne entre chaque séparateur ";"
            int token_index = 0;//va nous permettre de compter à a quel élément de la ligne nous somme, exemple : 0 pour le nom, 1, pour l'adresse, etc...
            while (token != NULL && token_index<4) {
                switch (token_index) {
                    case 0:
                        restaurant[num_ligne - 1].nom_restaurant = malloc( size: strlen( s: token)+1); //on alloue dynamiquement pour ne pas restreindre la taille et ne
                        strcpy(restaurant[num_ligne - 1].nom_restaurant, token);//copier le nom du restaurant contenu dans token dans le restaurant[i].nom_restaurant
                        break;
                    case 1:
                        restaurant[num_ligne - 1].adresse_restaurant = malloc( size: strlen( s: token)+1);
                        strcpy(restaurant[num_ligne - 1].adresse_restaurant, token);
                        break;
                    case 2:
                        restaurant[num_ligne - 1].position_restaurant = malloc( size: sizeof(Position));
                        if (sscanf(strstr( big: token,  little: "x="), "x=%lf", &restaurant[num_ligne - 1].position_restaurant->x) != 1) {
                            printf("Impossible d'extraire la valeur de x.\n");
                            return 0;
                        }
                        restaurant[num_ligne - 1].specialite = malloc( size: strlen( s: token)+1);
                        if (sscanf(strstr( big: token,  little: "y="), "y=%lf", &restaurant[num_ligne - 1].position_restaurant->y) != 1) {
                            printf("Impossible d'extraire la valeur de y.\n");
                            return 0;
                        }
                        /* ces lignes de code recherche la sous-chaîne "x=" et "y=" dans le token (donc dans la partie des coordonnées),
                         * extrait un nombre décimal en virgule flottante qui suit cette sous-chaîne,
                         * puis stocke cette valeur dans les variable x et y du membre position_restaurant du restaurant
                         */
                        break;
                    case 3:
                        strcpy(restaurant[num_ligne - 1].specialite, token);
                        break;
                }
                token = strtok( str: NULL,  sep: ";");//permet de passer à l'élément suivant dans le découpage de la ligne
                token_index++;
            }
            num_ligne++;//compter les lignes
        }
    }
    fclose(file);//fermeture du fichier
    return num_ligne-1;//on return -1 car au début on à fait +1 pour passer la première ligne
}
```

La fonction int *lire_restaurant*(char* chemin, Restaurant restaurant[]) effectue la lecture d'un fichier texte contenant des informations sur les restaurants. Elle commence par ignorer la première ligne, qui contient les en-têtes de colonnes. Ensuite, elle traite chaque ligne du fichier en extrayant le nom, l'adresse, les coordonnées géographiques (x et y), et la spécialité de chaque restaurant.
Plus en détail, pour chaque ligne non vide, la fonction découpe les informations en utilisant des points-virgules comme séparateurs. Elle alloue de la mémoire dynamiquement pour stocker les informations, en veillant à éviter les lignes vides.
Elle recherche les coordonnées géographiques (x et y) dans la ligne en cherchant les sous-chaînes "x=" et "y=" et les extrait pour les stocker dans la structure Restaurant. En cas d'erreur lors de l'extraction des coordonnées, la fonction affiche un message d'erreur.

Enfin, la fonction ferme le fichier après avoir terminé la lecture puis renvoie le nombre total de restaurants lus, en excluant la première ligne contenant les noms de colonnes.

Question 3 - Insertion :



```c
int inserer_restaurant(Restaurant r){

    FILE *file;
    file = fopen( filename: "restau.txt", mode: "a");//ouverture en écriture pour des ajouts en fin de fichier
    if (file==NULL){
        printf("Impossible d'ouvrir le fichier ! \n");
        return 0;
    }

    printf("\nEntrez les informations du restaurant : \n");

    printf("- Nom du restaurant : ");
    r.nom_restaurant = malloc( size: sizeof(char*));
    fgets(r.nom_restaurant,100,stdin);
    r.nom_restaurant[strcspn( s: r.nom_restaurant,  charset: "\n")] = '\0'; //Sert à supprimer le '\n' qui se rajoute avec un fgets

    printf("- Adresse du restaurant : ");
    r.adresse_restaurant = malloc( size: sizeof(char*));
    fgets(r.adresse_restaurant,100,stdin);
    r.adresse_restaurant[strcspn( s: r.adresse_restaurant,  charset: "\n")] = '\0'; //Sert à supprimer le '\n' qui se rajoute avec un fgets

    printf("- Specialité du restaurant : ");
    r.specialite = malloc( size: sizeof(char*));
    fgets(r.specialite,100,stdin);
    r.specialite[strcspn( s: r.specialite,  charset: "\n")] = '\0'; //Sert à supprimer le '\n' qui se rajoute avec un fgets
```

```c
    r.position_restaurant = malloc( size: sizeof(T_position));
    printf("- Position du restaurant : \n");
    printf("x = ");
    scanf("%lf",&r.position_restaurant->x);
    printf("y = ");
    scanf("%lf",&r.position_restaurant->y);

    fprintf(file,"\n%s; %s;(x=%lf,y=%lf); {%s};\n",r.nom_restaurant,r.adresse_restaurant,r.position_restaurant->x,r.position_restaurant->y,r.specialite);
    //on ecrit dans le fichier au même format que le reste des restaurants
    fclose(file);

    //on libère la mémoire précédemment allouée
    free(r.nom_restaurant);
    free(r.adresse_restaurant);
    free(r.specialite);
    free(r.position_restaurant);

    printf("\nLe restaurant a bien été inséré dans le fichier.\n");

    return 1;
}
```

La fonction int *inserer_restaurant*(Restaurant r) permet d'ajouter les informations d'un nouveau restaurant dans un fichier texte "restau.txt". Elle ouvre le fichier en mode ajout ("a"), demande à l'utilisateur d'entrer les détails du restaurant, y compris le nom, l'adresse, la spécialité et les coordonnées géographiques (x et y), puis écrit ces informations dans le fichier au même format que les autres restaurants. Ensuite, elle libère la mémoire allouée pour les informations du restaurant et affiche un message de confirmation. La fonction renvoie 1 pour indiquer le succès de l'insertion.

Question 4 - Recherche selon une position :

```c
//calculer la distance entre une position et 1 point donné
double calc_distance(Position *pos1, double x, double y) {
    double dx = pos1->x - x;
    double dy = pos1->y - y;
    return sqrt(pow(dx,2) + pow(dy,2));
}
```

Nous avons commencé par créer une fonction qui permet de calculer la distance entre une position et un point donné. Le fait d'avoir créer cette fonction va nous permettre de simplifier et d'alléger en terme de lecture et d'écriture nos algorithmes.

```c
int cherche_restaurant(double x, double y, double rayon_recherche, Restaurant results[]){

    Restaurant *allRestaurants;
    allRestaurants = malloc( size: L_MAX * sizeof(Restaurant));

    int numRestaurants = lire_restaurant( chemin: "restau.txt",  restaurant: allRestaurants);//on relève le nombre de restaurant présent dans le fichier

    int nombre_result = 0;

    /*création d'un tableau pour stocker tous les restaurants et leur distance
     *chaque "case" du tableau comprendra un élément de type RestaurantDistance,
     *on pourra comme ça stocker dans une seule case du tableau un restaurant et sa distance avec l'usager*/
    RestaurantDistance *restaurantDistances = malloc( size: numRestaurants * sizeof(RestaurantDistance));

    // Calculez la distance pour chaque restaurant par rapport à la position actuelle de l'utilisateu
    // et stockez ces informations dans le tableau de RestaurantDistance.
    for (int i = 0; i < numRestaurants; i++) {
        double distance = calc_distance( pos1: allRestaurants[i].position_restaurant, x, y);
        restaurantDistances[i].restaurant = allRestaurants[i];
        restaurantDistances[i].distance = distance;
    }
```

```c
    // Triez le tableau de RestaurantDistance en fonction de la distance (on effectue ici un tri par insertion).
    for (int i = 1; i < numRestaurants; i++) {
        RestaurantDistance temp = restaurantDistances[i];
        int j = i - 1;
        while (j >= 0 && restaurantDistances[j].distance > temp.distance) {
            restaurantDistances[j + 1] = restaurantDistances[j];
            j--;
        }
        restaurantDistances[j + 1] = temp;
    }

    //ici on affecte enfin les restaurantsé étant dans le rayon dans le tableau result passé en paramètre
    for (int i = 0; i < numRestaurants; ++i) {
        if (restaurantDistances[i].distance<=rayon_recherche){
            results[nombre_result] = restaurantDistances[i].restaurant;
            nombre_result++;//pour connaitre le nombre de restaurant dans le rayon
        }
    }

    if (nombre_result>0) {
        for (int i = 0; i < nombre_result; ++i) {
            printf("\n%s - Distance : %lf", results[i].nom_restaurant,calc_distance( pos1: results[i].position_restaurant,x,y));       }
    }//on afficher tous les restaux trouvés avec leur distance par rappor tà l'utilisateur
    printf("\n");
    return nombre_result;
}
```

La fonction int *cherche_restaurant*(double x, double y, double *rayon_recherche*, Restaurant results[]) réalise une recherche de restaurants en fonction d'une position (x, y) spécifiée et d'un rayon de recherche donné. Elle commence par allouer de la

mémoire pour un tableau dynamique *allRestaurants* de type Restaurant. Ce tableau servira à stocker les informations de tous les restaurants.

En utilisant la fonction *lire_restaurant*, elle extrait les informations des restaurants à partir du fichier "restau.txt" et les stocke dans le tableau *allRestaurants* et récupère le nombre total de restaurant. Ensuite, elle alloue de la mémoire pour un tableau dynamique de type *RestaurantDistance* appelé *restaurantDistances*. Ce tableau a pour but de stocker chaque restaurant en association avec sa distance par rapport à la position de recherche.

La fonction calcule la distance entre la position actuelle de l'utilisateur (x, y) et chaque restaurant en parcourant le tableau *allRestaurants*. Elle associe chaque restaurant à sa distance respective, puis stocke ces informations dans le tableau *restaurantDistances*.
Une fois que le tableau *restaurantDistances* est complet, la fonction le trie en fonction de la distance, en effectuant un tri par insertion. Cette étape permet de classer les restaurants du plus proche au plus éloigné par rapport à la position de recherche.

La fonction filtre les restaurants qui se trouvent à l'intérieur du rayon de recherche en comparant la distance de chaque restaurant avec le rayon spécifié. Les restaurants qui satisfont cette condition sont stockés dans un tableau *results* passé en paramètre. Elle compte également le nombre de restaurants trouvés. Si des restaurants sont trouvés dans le rayon de recherche, la fonction affiche les noms de ces restaurants ainsi que leur distance par rapport à la position de recherche.

Enfin, la fonction renvoie le nombre de restaurants trouvés dans le rayon de recherche, ce qui permet à l'utilisateur de savoir combien de restaurants répondent à ses critères de recherche.

Question 4 - Recherche par spécialité :

```c
void cherche_par_specialite(double x, double y, char *specialite[], Restaurant results[]) {

    Restaurant *allRestaurants;
    allRestaurants = malloc( size: L_MAX * sizeof(Restaurant));
    int numRestaurants = lire_restaurant( chemin: "restau.txt",  restaurant: allRestaurants);//on relève le nombre de restaurant présent dans le fichier

    int nombre_result = 0;

    /*création d'un tableau pour stocker tous les restaurants et leur distance
    *chaque "case" du tableau comprendra un élément de type RestaurantDistance,
    *on pourra comme ça stocker dans une seule case du tableau un restaurant et sa distance avec l'usager*/
    RestaurantDistance *restaurantDistances = malloc( size: numRestaurants * sizeof(RestaurantDistance));

    // Calculez la distance pour chaque restaurant par rapport à la position actuelle de l'utilisateur
    // et stockez ces informations dans le tableau de RestaurantDistance.
    for (int i = 0; i < numRestaurants; i++) {
        double distance = calc_distance( pos1: allRestaurants[i].position_restaurant, x, y);
        restaurantDistances[i].restaurant = allRestaurants[i];
        restaurantDistances[i].distance = distance;
    }

    // Triez le tableau de RestaurantDistance en fonction de la distance (ici on effectue un tri par insertion).
    for (int i = 1; i < numRestaurants; i++) {
        RestaurantDistance temp = restaurantDistances[i];
        int j = i - 1;
        while (j >= 0 && restaurantDistances[j].distance > temp.distance) {
            restaurantDistances[j + 1] = restaurantDistances[j];
            j--;
        }
        restaurantDistances[j + 1] = temp;
    }
```

```c
    // Parcourez le tableau trié et ajoutez les restaurants correspondant aux spécialités passées en paramètre
    for (int i = 0; i < numRestaurants; i++) {
        char *token = strtok( str: restaurantDistances[i].restaurant.specialite,  sep: "{ , }");
        //va permettre de séparer les diférentes spécialités du restaurant des séparateurs  "{ , }" pour bien isolé chaque sépcialitée

        while (token != NULL) {//boucle while parcourir toutes les spécialités du restaurant
            // Convertir la spécialité en minuscules pour la comparaison insensible à la casse
            for (int k = 0; token[k] != '\0'; k++) {
                token[k] = tolower( c: token[k]);//tolower va permettre de convertir en minuscule
            }
            for (int j = 0; specialite[j] != NULL; j++) {//boucle pour parcourir toutes spécialités entrées par l'utilisateur
                // Convertir la spécialité recherchée en minuscules pour la comparaison insensible à la casse
                char specialiteLower[N_MAX];
                strcpy(specialiteLower, specialite[j]);
                for (int k = 0; specialiteLower[k] != '\0'; k++) {
                    specialiteLower[k] = tolower( c: specialiteLower[k]);
                }
                //comparer les 2 spécialités en minuscule
                if (strcmp(token, specialiteLower) == 0) {
                    results[nombre_result] = restaurantDistances[i].restaurant;
                    nombre_result++;
                    break;
                }
            }

            token = strtok( str: NULL,  sep: ", ");//passer à la spécialité suivante du restaurant
        }
    }
```

```c
    if(nombre_result>0){
        for (int i = 0; i < nombre_result; ++i) {
            printf("\n%s - Distance : %lf", results[i].nom_restaurant,results[i].position_restaurant->x,results[i].position_restaurant->y,
                   calc_distance( pos1: results[i].position_restaurant,x,y));
        }
        printf("\n\n%d restaurant(s) trouvé selon vos/votre spécialite(s)\n",nombre_result);
        //on affiche le nombre de restaurant trouvé et les différents restaurant du plus proche au plus loin
    } else
        printf("Aucun restaurant ne fait cette/ces spécialités !\n");


    // Libérer la mémoire allouée pour chaque spécialité
    for (int i = 0; i < numRestaurants; i++) {
        free(restaurantDistances[i].restaurant.specialite);
    }
    free(allRestaurants);
    free(restaurantDistances);
}
```

La fonction void cherche_par_specialite(double x, double y, char *specialite[], Restaurant results[]) effectue une recherche de restaurants en fonction de leur spécialité, de la proximité par rapport à une position spécifique (x, y) et à l'intérieur d'un rayon donné.
La fonction commence par allouer de la mémoire pour un tableau dynamique *allRestaurants* de type Restaurant, destiné à stocker les informations de tous les restaurants.
En utilisant la fonction *lire_restaurant*, elle extrait les informations des restaurants à partir du fichier "restau.txt", les stocke dans le tableau *allRestaurants* et récupère le nombre total de restaurant dans le fichier.
La fonction alloue ensuite de la mémoire pour un tableau dynamique de type *RestaurantDistance* appelé *restaurantDistances*. Ce tableau servira à stocker chaque restaurant en association avec sa distance par rapport à la position de recherche.
La distance entre la position actuelle de l'utilisateur (x, y) et chaque restaurant est calculée en parcourant le tableau *allRestaurants*. La fonction associe chaque restaurant à sa distance respective, puis stocke ces informations dans le tableau *restaurantDistances*.
Une fois que le tableau *restaurantDistances* est complet, la fonction le trie en fonction de la distance, en effectuant un tri par insertion. Cette étape permet de classer les restaurants du plus proche au plus éloigné par rapport à la position de recherche.
La fonction filtre les restaurants en parcourant le tableau trié et ajoute ceux qui correspondent aux spécialités passées en paramètre. Elle prend en compte la sensibilité à la casse (minuscules/majuscules) en convertissant les spécialités en minuscules pour la comparaison.
Les restaurants correspondant aux spécialités recherchées sont stockés dans un tableau *results*, avec un compteur pour connaître leur nombre.
Si des restaurants sont trouvés, la fonction affiche les noms de ces restaurants ainsi que la distance par rapport à la position de recherche. Elle informe également l'utilisateur du nombre de restaurants trouvés.
Si aucun restaurant ne correspond aux spécialités recherchées, la fonction affiche un message indiquant qu'aucun restaurant ne propose ces spécialités.
Enfin, la mémoire allouée pour les spécialités de chaque restaurant est libérée, et la mémoire allouée pour les tableaux est également libérée.

La fonction permet donc de rechercher des restaurants en fonction de leur spécialité, de leur proximité géographique, et de les afficher triés par distance.

Enfin, nous avons créé un main permettant d'afficher un menu et de laisser le choix à l'utilisateur de gérer toutes ces fonctions. (voir fichier EX3_main.c)