Rapport Devoir 1

Devoir 1 SR01

Table des matières

1 Introduction

Nous avons été amenés à travailler sur le premier DM de SR01 les deux premières semaines d'octobre 2022. Ce premier DM est une mise en pratique des notions vues en cours sur la première partie. Il concerne exclusivement le langage C, et porte sur les fonctions et la manipulation de fichiers. Le sujet de ce premier DM est disponible ici.

Ce rapport contient les explications des fonctions présentes dans les différents dossiers du rendu (EX1, EX2 et EX3). Des commentaires ont été laissés dans le code afin de mieux l'expliquer. De plus, nous allons également expliciter nos choix vis-à-vis du sujet et notre compréhension de l'énoncé.

Les différentes fonctions commentées sont présentes dans les dossiers suivants :

- *EX1* : pour l'exercice 1 ne contenant que les codes donnés dans le sujet
- *EX2* : pour l'exercice 2 contenant le fichier header (*EX2.h*), source (*EX2.c*) et le main permettant son utilisation (*main.c*)
- *EX3* : pour l'exercice 3 contenant le fichier header (*EX3.h*), source (*EX3.c*) et le main permettant son utilisation (*main.c*)

2 Exercice 1

Pour chacun de ces programmes, exécuter le programme et donner une explication du résultat obtenu. Pour comprendre ces programmes, quelques rappels sont nécessaires :

2.1 Rappels

2.1.1 Rappel 1 - booléens

Quand on traite des booléens, 0 (ou false) est évalué à faux, tandis que tout le reste sera évalué à vrai. L'opposé booléen (avec !) d'une valeur non fausse renverra donc toujours 0 (false), l'opposé booléen d'une valeur fausse quant à elle vaudra 1 (true). (ex : !20 vaut 0, et !0 ou !false valent 1).

2.1.2 Rappel 2 - lvalue et rvalue

Une lvalue est une donnée représentant un emplacement mémoire, une rvalue est une donnée représentant une valeur. Une lvalue peut être utilisée comme une rvalue, mais une rvalue ne peut pas être utilisée comme une lvalue. (ex : 20 est une rvalue, x est une lvalue). Ainsi, on ne peut pas modifier la valeur d'une rvalue, mais on peut modifier la valeur d'une lvalue. (ex : x = 20 est possible, 20 = x est impossible).

De manière symbolique, nous pourrions résumer ces notions au fait que les rvalues ne peuvent être placées à gauche d'un signe =, tandis qu'une lvalue peut être placer des deux côtés (à gauche pour l'affectation et à droite pour affecter sa valeur).

2.2 Programme 1

Il y a deux éléments majeurs à analyser dans ce programme : l'instruction !- -A et l'instruction ++!B.
Pour la première, on a - -A, qui décrémente la valeur de A et renvoie la valeur une fois A décrémenté (car pré-incrémentation). !- -A est donc équivalent à !19, ce qui est équivalent à 0.
Pour la seconde, on a !B, qui renvoie 0, car B est égal à 5. !B renvoie donc une rvalue, et on ne peut pas l'incrémenter, car on ne peut pas modifier la valeur d'une rvalue. On a donc une erreur.

```c
#include <stdio.h>

int main (){
    int A = 20, B = 5;
    int C=!- - A /++! B;

    printf ("A=%d B=%d C=%d \n", A, B, C);
}
```

FIGURE 1 – Programme 1

2.3 Programme 2

Il y a un élément majeur à analyser dans ce programme :
l'expression A&&B||!0&&C++&&!D++.
Pour la première partie, on a A&&B, qui renvoie 1, car A vaut 20 et B vaut 5. Les valeurs différentes de 0 et initialisées sont équivalentes à vrai, seul 0 équivaut à faux. A&&B est donc équivalent à 1.

On a ensuite l'opérateur logique ||, qui signifie "ou" et qui renvoie 1 si l'une des deux valeurs est égale à 1, et 0 sinon. Ici, le premier terme est égal à 1, donc l'expression est égale à 1 et le reste de l'expression n'est pas évalué.

La suite **non exécutée** est la suivante :
- !0 : aurait été évalué à 1.
- C++ : aurait été évalué à 1 et C aurait été incrémenté en post incrémentation.
- !D++ : aurait été évalué à 0 et D aurait été incrémenté en post incrémentation.

```c
#include <stdio.h>

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);
}
```

FIGURE 2 – Programme 2

2.4 Programme 3

Dans cette fonction, la partie la plus importante est l'expression *++q**q++.

Pour la première partie, on a *++q, cette instruction incrémente d'abord q, puis renvoie la valeur de la case mémoire pointée par la nouvelle valeur de q. Ici, q valait l'adresse de la première case du tableau p, donc q vaut maintenant l'adresse de la deuxième case du tableau p. *++q est donc équivalent à p[1], soit -2.

Pour la seconde partie, on a *q++, cette instruction renvoie la valeur de la case mémoire pointée par q, puis incrémente q. Ici, q vaut l'adresse de la deuxième case du tableau p, (q a été incrémenté dans la première partie), donc *q++ est équivalent à p[1]. *q++ renvoie donc -2, et q vaut maintenant l'adresse de la troisième case du tableau p, soit 3.
On a donc *++q**q++ qui est équivalent à p[1]*p[1], soit -2*-2, donc 4, et *q vaut 3.

```c
#include <stdio.h>

int main (){
    int p[4]={1,-2,3,4};
    int *q=p;

    printf("c=%d\n", *++q**q++);
    printf("c=%d\n", *q);
}
```

FIGURE 3 – Programme 3

2.5 Programme 4

Pour ce programme, il faut se focaliser sur l'expression *q & *q++ | *q++.

Pour la première partie, on a *q & *q++. *q renvoie la valeur de la case mémoire pointée par q, soit 1. *q++ renvoie la valeur de la case mémoire pointée par q, soit 1, puis incrémente q. *q & *q++ est donc équivalent à 1 & 1, soit 0001 & 0001, soit 0001, soit 1 ("et" binaire).

Pour la seconde partie, nous utilisons le résultat précédent et nous allons appliquer le ou binaire |. Nous avons *q++ qui renvoie la valeur de la case mémoire pointée par q, soit -2, car q a été incrémenté dans la première partie et pointe maintenant sur la deuxième case du tableau p. *q++ renvoie donc -2, puis incrémente q.

*q & *q++ | *q++ est donc équivalent à 1 | -2, soit 0001 | 1110, soit 1111, soit -1 ("ou" binaire puis complémentation à 2). À la fin de l'expression, q vaut l'adresse de la troisième case du tableau p, soit 3 et d vaut -1.

```c
#include <stdio.h>

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);
}
```

FIGURE 4 – Programme 4

2.6 Programme 5

Pour ce programme, il faut se focaliser sur l'expression ++a&&- -b ? b- - : a++.

Pour la première partie, on a ++a&&- -b, qui incrémente a, décrémente b, puis effectue l'opérateur "et" sur leurs nouvelles valeurs, ++a&&- -b est donc équivalent

à -7&&2, soit 1.

On a ensuite l'opérateur ternaire ?, qui renvoie la valeur de la deuxième expression si la première est évaluée à vrai, et la valeur de la troisième si la première est évaluée à faux. Ici, la première expression est évaluée à vrai, donc la valeur de la deuxième expression est renvoyée, soit b- -, qui renvoie la valeur de b, soit 2, puis décrémente.

++a&&- -b ? b- - : a++ ; est donc équivalent à 2. On a donc a vaut -7, b vaut 1 et c vaut 2.

```
#include <stdio.h>

int main (){
    int a = -8, b=3;
    int c =++a&&- -b ? b- -:a++;

    printf("a=%d b=%d c=%d\n", a, b, c);
}
```

FIGURE 5 – Programme 5

2.7 Programme 6

Dans ce programme, le plus important est la ligne a »= 2 ^ b.

Pour cette ligne, on a d'abord l'opérateur "ou exclusif" avec 2 ^ b, qui est équivalent à 2 ^ 3, soit en binaire 10 ^ 11, donc 1.

On a ensuite l'opérateur "décalage à droite" avec a »= 2 ^ b, donc a »=1, qui est équivalent à a/2 donc -8/2 = -4 (on décale de 1 bit vers la droite, donc on divise par 2). Cela se vérifie : 1111 1000 »=1 donne 1111 1100, soit -4.
On a donc a vaut -4 et b vaut 3.

```
#include <stdio.h>

int main (){
    int a =-8, b=3;
    a >>=2^b;

    printf("a=%d\n", a);
}
```

FIGURE 6 – Programme 6

3 Exercice 2

3.1 Analyse du sujet

L'objectif de cet exercice est de nous faire réfléchir sur l'utilisation des boucles et des tableaux. Afin de rendre le code simple, ce dernier a été rédigé sous forme de fonctions.

3.2 Question 1

Dans cette question, nous allons définir une fonction qui lit les notes de N étudiants et les mémorise dans un tableau points de dimension N. La question semble indiquer que la taille est fixe, nous allons donc utiliser define pour définir globalement la taille du tableau. Pour lire les notes, nous effectuons une boucle, nous nous servons ici à nouveau de N que nous avons défini.

```
#define N 15
```

La fonction retourne un tableau d'entiers, son retour est donc de type int*. On commence par allouer dynamiquement un tableau de la taille de N entiers, puis on demande N fois à l'utilisateur d'entrer une note (grâce à une boucle). Par au vu de la suite du sujet, il semble qu'il y ait un maximum pour la valeur d'une note, qui est 60. On s'assure donc que l'utilisateur a bien entré une note entre 0 et 60 à chaque fois. Voici la fonction :

```
int* lectureNotes() {
    int* points = malloc(N * sizeof(int));
    int i;
    for (i = 0; i < N; i++) {
        do {
            printf("Entrez la note de l'etudiant %d: ", i + 1);
            scanf("%d", &points[i]);
            if (points[i] < 0 || points[i] > 60) {
                printf("La note doit etre entre 0 et 60.\n");
            }
        } while (points[i] < 0 || points[i] > 60);
    }
    return points;
}
```

FIGURE 7 – Fonction lectureNotes

3.3 Question 2

Dans cette question, nous allons définir une fonction qui calcule le minimum, le maximum et la moyenne d'un tableau de notes. Il est seulement demandé d'afficher le minimum, le maximum et la moyenne, le type de retour de la fonction est donc void. Pour cela, nous allons utiliser une boucle pour parcourir le tableau et calculer

les valeurs. Pour le minimum et le maximum, nous initialisons les variables min et max à la première valeur du tableau. Pour la moyenne, nous initialisons la variable somme à 0. Pour chaque valeur du tableau, nous vérifions si elle est plus petite que min, si c'est le cas, nous mettons à jour la valeur de min. De même, nous vérifions si la valeur est plus grande que max, si c'est le cas, nous mettons à jour la valeur de max. Pour la moyenne, nous additionnons simplement toutes les valeurs du tableau et nous les divisons ensuite par le nombre d'éléments du tableau (sa taille). Voici la fonction :

```
void statsNotes(int points[N]) {
    int i;
    int max = points[0];
    int min = points[0];
    int somme = 0;
    for (i = 0; i < N; i++) {
        if (points[i] > max) {
            max = points[i];
        }
        if (points[i] < min) {
            min = points[i];
        }
        somme += points[i];
    }
    printf("La note maximale est %d, la note minimale est  %d, la
moyenne est %f.\n", max, min, (float) somme / N);
}
```

FIGURE 8 – Fonction statsNotes

3.4 Question 3

Dans cette question, nous allons définir une fonction qui établit un tableau notes de dimension 7 contenant le nombre d'étudiants ayant obtenu un certain intervalle de notes par tranche de 10 (notes[0] étudiants ayant obtenu entre 0 et 9, notes[1] étudiants ayant obtenu entre 10 et 19, etc.). notes[6] étudiants ayant obtenu 60, on en déduit donc que 60 est la note maximale. Le type de retour de la fonction est donc int*, car on retourne un tableau d'entiers. Pour le fonctionnement de la fonction, nous allouons dynamiquement un tableau de 7 entiers. Ensuite nous utilisons une boucle pour parcourir le tableau contenant les notes, à chaque valeur, nous ajoutons 1 à la case correspondante dans le tableau notes. Voici la fonction :

```c
int* notes_tab(int points[N]){
    int* notes = malloc(7 * sizeof(int));
    int i;
    for (i = 0; i < 7; i++) {
        notes[i] = 0;
    }
    for (i = 0; i < N; i++) {
        if (points[i] == 60) {
            notes[6]++;
        } else if (points[i] >= 50) {
            notes[5]++;
        } else if (points[i] >= 40) {
            notes[4]++;
        } else if (points[i] >= 30) {
            notes[3]++;
        } else if (points[i] >= 20) {
            notes[2]++;
        } else if (points[i] >= 10) {
            notes[1]++;
        } else {
            notes[0]++;
        }
    }
    return notes;
}
```

FIGURE 9 – Fonction notes_tab

3.5 Question 4

Dans cette question, nous allons définir une fonction qui établit un graphique en nuage de points représentant le tableau notes de la question précédente. Cette fonction récupère le tableau notes avec la fonction notes_tab et le tableau de notes en paramètre. Cette fonction effectue juste un affichage, son type de retour est donc void. Le reste de la fonction est juste une mise en pratique de ce que nous avons appris pour formater du texte et faire des boucles. Il y a cependant une différence avec le sujet : plutôt que d'afficher un nombre de lignes fixe, nous calculons le maximum du tableau notes et nous affichons autant de lignes que la valeur maximale. Voici la fonction :

```c
void affichagePoints(int points[N]) {
    int* tab_notes = notes_tab(points);
    int max = tab_notes[0];
    for (int i = 0; i < 7; i++) {
        if (tab_notes[i] > max) {
            max = tab_notes[i];
        }
    }
    for (int i = max; i > 0; i--) {
        printf("%2.d >  ", i);
        for (int j = 0; j < 7; j++) {
            if (tab_notes[j] == i) {
                printf("    o   ");
            } else {
                printf("        ");
            }
        }
        printf("\n");
    }
    printf("      ");
    for (int j = 0; j < 7; j++) {
        if (tab_notes[j] == 0) {
            printf("+--o--");
        } else {
            printf("+------");
        }
    }
    printf("+\n      ");
    for (int j = 0; j < 6; j++) {
        printf("| %-2.1d-%2.d ", j * 10, (j + 1) * 10 - 1);
    }
    printf("|  %d+  |\n\n", 60);
}
```

FIGURE 10 – Fonction affichagePoints

3.6 Question 5

Dans cette question, nous allons faire comme pour la question précédente, mais en affichant cette fois-ci un histogramme. Le principe de la fonction est le même, mais avec des caractères d'affichage différent. Voici la fonction :

```c
void affichageBarres(int points[N]) {
    int* tab_notes = notes_tab(points);
    int max = tab_notes[0];
    for (int i = 0; i < 7; i++) {
        if (tab_notes[i] > max) {
            max = tab_notes[i];
        }
    }
    for (int i = max; i > 0; i--) {
        printf("%2.d >  ", i);
        for (int j = 0; j < 7; j++) {
            if (tab_notes[j] >= i) {
                printf(" ########");
            } else {
                printf("         ");
            }
        }
        printf("\n");
    }
    printf("       ");
    for (int j = 0; j < 7; j++) {
        printf("+--------");
    }
    printf("+\n       ");
    for (int j = 0; j < 6; j++) {
        printf("| %-2.1d-%2.d ", j * 10, (j + 1) * 10 - 1);
    }
    printf("|  %d+  |\n\n", 60);
}
```

FIGURE 11 – Fonction affichageBarres

3.7 Programme principal

Nous définissons un programme principal qui va appeler les fonctions précédentes. On lit les notes avec la fonction lectureNotes, on affiche ses statistiques avec la fonction statsNotes, puis on affiche graphiquement les résultats avec les fonctions affichagePoints et affichageBarres. La fonction notes_tab n'est pas appelée directement, mais par les fonctions affichagePoints et affichageBarres. À la fin du programme, nous libérons la mémoire allouée pour le tableau de notes avec la fonction free. Voici le programme principal :

```c
int main() {
    int* points = lectureNotes();
    statsNotes(points);
    affichagePoints(points);
    affichageBarres(points);
    free(points);
    return 0;
}
```

FIGURE 12 – Programme principal

4 Exercice 3

L'objectif de cet exercice est de nous faire réfléchir sur l'utilisation des structures. Nous avons choisi de modifier légèrement quelques fonctions afin d'en faciliter l'implémentation et l'utilisation. Nous avons explicité nos choix dans les parties de chaque question. Dans cet exercice, nous utilisons le fichier texte fournit avec le sujet. Nous déduisons déjà que nous ne devons pas traiter la première ligne et que nous allons devoir traiter le cas de ligne vide.

Le format de chaque ligne est le suivant :

```
nom_restau; num, rue - ville; (x=val_x, y=val_y); {specialite};
```

4.1 Déclaration de la structure

Nous avons choisi de stocker dans une liste chaînée simple la liste des spécialités. Dans une structure représentant le restaurant, nous avons donc un pointeur vers la liste des spécialités. La structure Specialite contient un pointeur vers la spécialité suivante et un tableau de caractères contenant le nom de la spécialité. La structure Restaurant contient un pointeur vers la liste des spécialités, un tableau de caractères contenant le nom du restaurant, un tableau de caractères contenant l'adresse du restaurant et un tableau de double contenant les coordonnées du restaurant. Cette structure nous permet de stocker les informations d'un restaurant et de les manipuler facilement.

Les structures sont choisies sont les suivantes :

```c
typedef struct Specialite {
    char nom_specialite[MAX_MOT];
    struct Specialite *suivant;
} Specialite;
```

FIGURE 13 – Structure Specialite

```c
typedef struct Restaurant {
    char nom_restaurant[MAX_MOT];
    char adresse_restaurant[MAX_ADRESSE];
    Specialite * specialite;
    double position_restaurant [2];
} Restaurant;
```

FIGURE 14 – Structure Restaurant

NB : MAX_MOT est défini à 40 et MAX_ADRESSE à 100

Nous utilisons donc les fonctions d'usages des listes :

— creer_specialite : permet d'allouer en mémoire une spécialité et de la retourner
— est_present : permet de parcourir une liste et rechercher si une spécialité est présente
— liberer_specialite : permet de libérer la mémoire allouée pour une liste de spécialité
— affiche_specialite : permet d'afficher toutes les spécialités d'un restaurant
— ajout_specialite : permet d'ajouter une spécialité à la liste chainée en fin de liste

Le code des différentes fonctions est présent dans le fichier *EX3.c* et leur prototype dans *EX3.h*.

Nous avons de modifier les prototypes des fonctions de recherche afin qu'elles puissent retourner le nombre de restaurants du tableau de résultat et de pouvoir utiliser cette valeur dans le main. Nous avons également ajouté une fonction d'affichage permettant d'afficher les informations du restaurant et la distance entre l'utilisateur et celui-ci.

4.2 Fonction : lire_restaurant

Cette fonction permet de lire un fichier de restaurant au format txt et de stocker les restaurants dans un tableau. Pour cela, elle utilise la fonction fgets qui permet de lire une ligne du fichier et de la stocker dans une chaine de caractère. Ensuite, elle utilise la fonction strtok qui permet de découper une chaine de caractère en plusieurs sous-chaines de caractère. Ici le séparateur est le caractère ';'. Enfin, elle utilise la fonction atof qui permet de convertir une chaine de caractère en un nombre à virgule flottante.

Le seul problème est pour les coordonnées qui sont au format (x=valeur ;y=valeur) et non pas (valeur ;valeur). Donc, il faut découper la chaine de caractère en deux sous-chaines de caractère et convertir chacune d'entre elles en nombre à virgule flottante. Pour cela, nous utilisons la fonction strtok qui permet de découper une chaine de caractère en plusieurs sous-chaines de caractère. Ici le séparateur est le caractère ','. Nous enlevons le caractère '(', 'x' et '=' de la première sous-chaine de caractère. Pour la deuxième sous-chaine de caractère, nous enlevons le caractère 'y', '=', ';' et ')'.

Enfin, nous convertissons les deux sous-chaines de caractère en nombre à virgule flottante, à l'aide de la fonction atof. Les strtok ne pouvant être utilisés qu'une seule fois sur une même chaine de caractère. C'est pourquoi nous utilisons une variable temporaire pour stocker la chaine de caractère contenant l'ensemble des spécialités.

Nous faisons appel aux fonctions creer_specialite et ajout_specialite pour créer la liste des spécialités et y ajouter les spécialités du restaurant.

La fonction étant assez longue, nous avons choisi de ne pas la mettre dans son entièreté dans le rapport. Elle est présente et commentée dans le fichier *EX3.c*

4.3 Fonction : inserer_restaurant

La fonction prend en paramètres les informations sur le restaurant et va les ajouter au fichier en utilisant son format spécifié plus haut. On ouvre le fichier en mode "a" pour ajouter à la fin du fichier.

Afin d'écrire les informations du restaurant, nous utilisons la fonction fprintf. Elle prend en paramètre le fichier dans lequel écrire, le format de la chaine de caractère à écrire, et les valeurs à insérer dans la chaine de caractère. Ainsi, pour écrire le nom du restaurant, nous utilisons *%s*, et nous passons le nom du restaurant en paramètre.

Nous écrivons ainsi les informations suivantes : Nom du restaurant ; Adresse du restaurant ; Position du restaurant ; Spécialités du restaurant en utilisant le caractère { pour indiquer le début de la liste, puis nous écrivons les spécialités séparées par des virgules, et nous terminons par le caractère } pour indiquer la fin de la liste.

Pour écrire les spécialités, nous utilisons une boucle while pour parcourir la liste des spécialités et ce tout en vérifiant si nous sommes au dernier élément de la liste. Si nous sommes à la fin de la liste, nous n'écrivons pas de virgule après le nom de la spécialité. Cette vérification est faite en vérifiant si le champ suivant est NULL. A la fin de la boucle, nous écrivons le caractère } pour fermer la liste des spécialités.

Enfin, nous fermons le fichier.

```
void inserer_restaurant(Restaurant restaurant) {
    FILE* output_file = fopen(CheminFichier, "a");
    if (output_file == NULL) {
        printf("Erreur: le fichier restaurants.txt n'a pas pu etre
ouvert\n");
        return;
    }
    fprintf(output_file, "\n\n%s; %s;(x=%lf, y=%lf); {", restaurant
.nom_restaurant, restaurant.adresse_restaurant, restaurant.
position_restaurant[0], restaurant.position_restaurant[1]);
    Specialite *specialite = restaurant.specialite;
    while(specialite != NULL) {
        if (specialite->suivant == NULL)
        {
            fprintf(output_file, "%s", specialite->nom_specialite);
        }
        else {
            fprintf(output_file, "%s, ", specialite->nom_specialite
);
        }
        specialite = specialite->suivant;
    }
    fprintf(output_file, "};\n\n");
    fclose(output_file);
}
```

FIGURE 15 – Fonction inserer_restaurant

4.4 Fonction : cherche_restaurant

Cette fonction prend en paramètre la position de l'utilisateur par rapport à x et à y, son rayon de recherche, la liste des résultats et des restaurants initiaux et enfin, le nombre de restaurants.

Nous avons ici ajouté en paramètre la liste des restaurants et leur nombre afin de pouvoir parcourir simplement cette liste. Afin, de calculer la distance entre le restaurant et l'utilisateur, nous utilisons la fonction calcul_distance qui retourne la distance entre deux points. Si cette distance est inférieure ou égale au rayon de recherche, on ajoute le restaurant à la liste des résultats.

La fonction tri ensuite les restaurants par ordre de proximité croissante avec l'utilisateur, ce n'était pas non plus demandé, mais cela nous semblait plus logique.

```
int cherche_restaurant(double x, double y, double rayon_recherche,
Restaurant results[], Restaurant restaurants[], int nb_restaurants)
{
    if (restaurants == NULL || nb_restaurants <= 0 || results ==
NULL || rayon_recherche <= 0) {
        return 0;
    }
    int nb_results = 0;
    for (int i = 0; i < nb_restaurants; i++) {
        if (calcul_distance(x, y, restaurants[i].
position_restaurant[0], restaurants[i].position_restaurant[1]) <=
rayon_recherche)
        {
            results[nb_results] = restaurants[i];
            nb_results++;
        }
    }
    if (nb_results > 1) {
        tri_restaurant(results, nb_results);
    }
    return nb_results;
}
```

FIGURE 16 – Fonction cherche_restaurant

La fonction de calcul de distance utilise les fonctions pow (puissance) et sqrt (racine carrée) de la librairie math.h. Et repose sur la formule suivante :

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2} = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$$

```
double calcul_distance(double x1, double y1, double x2, double y2)
{
    return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
}
```

FIGURE 17 – Fonction calcul_distance

La fonction de tri se base sur le tri par insertion.C'est-à-dire, on itère sur chaque élément du tableau et on l'insère à la bonne place parmi les éléments déjà triés. Ainsi, à un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés.

```
void tri_restaurant(Restaurant *results, int nb_restaurants) {
    if (results == NULL || nb_restaurants <= 0) {
        return;
    }
    Restaurant temp;
    for (int i = 0; i < nb_restaurants; i++) {
        for (int j = i + 1; j < nb_restaurants; j++) {
            if (calcul_distance(0, 0, results[i].
position_restaurant[0], results[i].position_restaurant[1]) >
calcul_distance(0, 0, results[j].position_restaurant[0], results[j
].position_restaurant[1])) {
                temp = results[i];
                results[i] = results[j];
                results[j] = temp;
            }
        }
    }
}
```

FIGURE 18 – Fonction tri_restaurant

4.5 Fonction : cherche_par_specialite

Nous avons également modifié le prototype de cette fonction en ajoutant la liste des restaurants et le nombre de restaurants. Nous avons modifié le type des spécialités puisque nous avons une structure adaptée.

Nous utilisons la fonction est_present lorsque nous itérons sur les spécialités d'un restaurant. Si la spécialité est présente dans notre liste de recherche, nous ajoutons le restaurant et nous utilisons la commande break afin de ne pas plus itérer sur les spécialités de ce restaurant et de passer au suivant.

Enfin, si nous avons au moins deux restaurants, nous trions le tableau par rapport à la distance.

```
int cherche_par_specialite(double x, double y, Specialite *spec,
Restaurant *results, Restaurant restaurants[], int nb_restaurants)
{
    if (restaurants == NULL || nb_restaurants <= 0 || results ==
NULL || spec == NULL) {
        return 0;
    }
    int pos =0;
    for (int i =0; i < nb_restaurants; i++) {
        Specialite *temp = restaurants[i].specialite;

        while (temp != NULL) {
            if (est_present(spec, temp->nom_specialite)) {
                results[pos] = restaurants[i];
                pos++;
                break;
            }
            temp = temp->suivant;
        }
    }
    if (pos > 1) {
        tri_restaurant(results, pos);
    }
    return pos;
}
```

FIGURE 19 – Fonction inserer_restaurant

4.6 Explication - main

Nous allons dans cette partie expliquer le main et afficher quelques images de l'interface de notre programme.

Nous allons utiliser le main afin de lier les différentes fonctions présentées plus haut et notre interface. L'utilisateur n'aura ainsi qu'à se laisser guider par notre interface dans ses choix et ses saisies.

Pour cela notre main fonctionne de la façon suivante. Nous allons d'abord afficher un listage des différents choix s'offrant à l'utilisateur (ce menu s'affichera après chacune des utilisations). Nous récupérons ensuite le choix de l'utilisateur et appelons la fonction associée (et récupérons les informations nécessaires pour son appel).Ce menu est le suivant :

```
********** Menu **********
1. Lecture restaurant fichier
2. Affichage restaurant
3. Ajout restaurant au fichier
4. Recherche restaurants autour de moi
5. Recherche restaurants par specialite
6. Liberation de la memoire et quitter
Saisir votre choix :
```

FIGURE 20 – Affichage - menu

Une fois le choix de l'utilisateur saisie, un affichage adapté est proposé. Nous faisons également une gestion de l'ouverture du fichier afin de ne pas ouvrir deux fois un fichier. De plus, cette gestion de l'ouverture permet de ne pas exécuter les fonctions suivantes si le fichier n'a pas été ouvert. C'est par exemple le cas des choix : affichage, recherche et quitter (si le fichier n'a pas été ouvert, on ne libère pas la mémoire puisque non utilisé). Nous avons ainsi les affichages suivants pour les différentes fonctions :

```
********** Ouverture du fichier **********

Nombre de restaurants dans le fichier : 21
```

FIGURE 21 – Affichage - ouverture fichier

Ouverture d'un fichier déjà ouvert :

```
********** Ouverture du fichier **********

Le fichier est deja ouvert
```

FIGURE 22 – Affichage - fichier déjà ouvert

Si l'utilisateur choisi une option non proposée :

```
********** Erreur **********

Choix invalide
```

FIGURE 23 – Affichage - Erreur saisie

Une fois que le fichier s'ouvre correctement, nous pouvons utiliser les différentes fonctions du menu.

À commencer par afficher notre liste de restaurant :

```
********** Affichage des restaurants **********

Restaurant : Edern
        Adresse : 6, rue Arsene Houssaye - Paris 8eme
        Position : +1.500000, +44.800000
        Specialite :
                Cuisine gastronomique
Restaurant : La Farnesina
        Adresse : 9, rue Boissy d'Anglas - Paris 8eme
        Position : +70.500000, +74.125780
        Specialite :
                Cuisine italienne
Restaurant : 41 Penthievre
        Adresse : 41 Penthievre 41, rue de Penthievre - Paris 8eme
        Position : +40.500000, +77.125780
        Specialite :
                Cuisine traditionnelle fran├°aise
Restaurant : 6 New York
        Adresse : 6, avenue de New York - Paris 16eme
        Position : +12.500000, +74.125780
        Specialite :
                Cuisine gastronomique
Restaurant : African Lounge
        Adresse : 20 bis, rue Jean Giraudoux - Paris 16eme
        Position : +18.500000, +7.578000
        Specialite :
                Cuisine africaine
```

FIGURE 24 – Affichage des restaurants

Nous pouvons ensuite recherche un restaurant par rapport à notre position et un rayon de recherche :

```
********** Recherche d'un restaurant **********

Entrez votre position X :
0
Entrez votre position Y :
0
Entrez le rayon de recherche :
20
Restaurant : Gotti
Distance : 8.559910
        Adresse : 48, rue de Prony - Paris 17eme
        Position : +7.500000, +4.125780
        Specialite :
                Cuisine italienne
Restaurant : Belle Armee
Distance : 12.824043
        Adresse : 3, avenue de la Grande Armee - Paris 16eme
        Position : +2.500000, +12.578000
        Specialite :
                Brasserie
Restaurant : Le Bistro Mavrommatis Passy
Distance : 15.062868
        Adresse : 70, avenue Paul Doumer - Paris 16eme
        Position : +10.500000, +10.800000
        Specialite :
                Cuisine grecque
```

FIGURE 25 – Recherche restaurants par rapport à ma position

Puis rechercher ce restaurant par rapport à une ou plusieurs spécialités. Nous avons choisi de chercher par rapport à la spécialité viande :

```
********** Recherche d'un restaurant par specialite **********

Entrez la specialite a rechercher :
Viande
Voulez-vous ajouter une autre specialite ? (1 : Oui, 0 : Non)
0

Entrez votre position X :
0
Entrez votre position Y :
0
Restaurant : Bey Steakhouse
Distance : 17.732910
        Adresse : 7-9, rue Waldeck-Rousseau - Paris 17eme
        Position : +12.500000, +12.578000
        Specialite :
                Viande
                rotisserie
Restaurant : Unico
Distance : 62.757250
        Adresse : 15, rue Paul Bert - Paris 11eme
        Position : +17.800000, +60.180000
        Specialite :
                Viande
                rotisserie
```

FIGURE 26 – Recherche restaurants par rapport à des spécialités

Enfin, nous souhaitons fermer notre programme et libérer la mémoire :

```
********** Fermeture du programme **********

Suppression du restaurant Edern
Suppression du restaurant La Farnesina
Suppression du restaurant 41 Penthievre
Suppression du restaurant 6 New York
Suppression du restaurant African Lounge
Suppression du restaurant Le 122
Suppression du restaurant Le 404
Suppression du restaurant Le 41 Pasteur
Suppression du restaurant Belle Armee
Suppression du restaurant Benkay
Suppression du restaurant Agape
Suppression du restaurant Un Air de Famille
Suppression du restaurant Le Bistro Mavrommatis Passy
Suppression du restaurant Le Caroubier
Suppression du restaurant Al Dente
Suppression du restaurant Bey Steakhouse
Suppression du restaurant Casa Luca
Suppression du restaurant Gotti
Suppression du restaurant Le Timgad
Suppression du restaurant La Famiglia
Suppression du restaurant Unico
Au revoir !
```

FIGURE 27 – Fermeture programme

5 Conclusion

Le premier DM nous a permis de mettre en application les notions vues en cours et en TD sur une nouvelle problématique plus large. Nous avons utiliser de nombreuses fonctions vues en cours et en TD afin de répondre aux exigences de l'énoncé. Nous avons également fait quelques choix afin de simplifier le bon déroulement de nos programmes. Ce premier devoir était très enrichissant.

En effet, nous avons grâce à celui-ci pu perfectionner les différents éléments suivants :

- compréhension des post-incrémentation et pré-incrémentation
- utilisation des opérateurs binaires
- utilisation de tableaux d'entiers et de caractères
- lecture et écriture dans un fichier texte
- utilisation de liste chaînée simple
- allocation et libération mémoire

Table des figures