Dossier SR01 DM1.

SR01 Devoirs N°1

Réalisé par
Alexandre Teixeira et Laurent Minatchy

Pour un devoir
Langage C/ M. Hicham Lakhlef

Dirigé par
M. Hicham Lakhlef

Réaliséà l'Université Technologique de Compiègne
UTC

Date butoire : 25 Octobre 2022

Table des matières

1. Introduction

1.1. Règle

Ce devoir peut se faire en binôme. Les deux étudiants formant un binôme doivent appartenir au même groupe de TD.

Les différents binômes deveront remettre un seul fichier *.zip contenant les codes sources ainsi qu'un rapport électronique (pdf) de quelques pages.

• Le dernier délai pour la remise est fixé pour le 25 octobre 2022.

2. Exercice 1

2.1. Premier Programme

2.1.1 Premier

Langage 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);
}
```

Listing 2.1 – Premier programme

2.1.2 Résultat

Ce code renvoie une erreur car l'incrémentation a la priorité sur les autres opérateurs. Or il n'y a aucune variable à incrémenter au bord du ++.

2.1.3 Second

Langage 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);
}
```

Listing 2.2 – Second programme

2.1.4 Résultat

Les icrémentations sont suffixes donc elles n'interviennent pas dans le calcul ligne 4. A && B = 1
!0 = 1
!0 && C ++= 1
!D++ = 0
! 0&& C ++&&! D ++ = 0
A&& B ||! 0&& C ++&&! D ++= 1 car A && B = 1

C et D ne sont pas incrémenter car (! 0&& C ++&&! D ++) est faux.
D'où ligne 4 affiche 1, c = -10 et d = 3.

2.1.5 Troisième

Langage C

```
1 # include < stdio.h>
2 int main (){
3 int p [4]={1 , -2 ,3 ,4};
4 int * q=p;
5 printf (" c=% d\n", *++ q** q ++) ;
6 printf (" c=% d \n" ,* q);
7 }
```

Listing 2.3 – Troisième programme

2.1.6 Résultat

Ligne 4 : *q = 1. Ainsi *++q renvoie -2 qui est le deuxième élément de la liste car q est incrémenter.
Ligne 5 : q est d'abord incrémenter par ++q et pointe sur -2. Donc *q++ renvoie -2 et ensuite q pointe sur 3.
Ainsi ligne 5 affiche 4 = (-2)*(-2) et ligne 5 affiche 3.

2.1.7 Quatrième

Langage C

```
1 #include <stdio.h>
2 int main (){
3 int p [4]={1 , -2 ,3 ,4};
4 int * q=p;
```

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

Listing 2.4 – Quatrième programme

2.1.8 Résultat

Ligne 4 : *q = 1
Ligne 5 : *q & *q++ => 1 et q pointe sur -2.
Ainsi * q &* q ++—* q ++ <=> 1—(-2) soit 0001 — 1110 ce qui donne -1.
De plus q est encore incrémenter en postfixé donc il pointe sur 3.
Ligne 6 : Affiche 3.

2.1.9 Cinquième

Langage C

```
#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);
}
```

Listing 2.5 – Cinquième programme

2.1.10 Résultat

Ligne 4 : ++a renvoie -7 et –b renvoie 2 d'où (++a && – b) renvoie 1.
Ainsi c = b – (suffixe) donc c = 2 et b = 1.
Ligne 5 : Donc a = -7, b = 1 et c = 2.

2.1.11 Sixième

Langage C

```
#include <stdio.h>
int main (){
```

```
3 int a=-8 , b =3;
4 a >>=2^b;
5 printf (" a=% d\n",a);
6 }
```

Listing 2.6 – Sixième programme

2.1.12 Résultat

Ligne 4 : 2 $\wedge\backslash$ b <=> 10 $\wedge\backslash$ 11 ce qui renvoie 1.
Ainsi on décale a = -8 d'un bit vers la droite soit 11000 devient 1100 = -4
Ligne 5 : Donc a = -4.

3. Exercice 2

3.1. Introduction

Le but de l'exercice est de créer un programme capable de faire des graphiques à partir des note des étudiants. Pour se faire, il faut alors implémenter des fonctions qui vont nous renvoyer la note maximale, et trier les notes dans des tableaux.

Première fonction

Écrire un programme qui lit les notes de N étudiants de l'UTC dans un devoir de l'UV SR01 et les mémorise dans un tableau POINTS de dimension N

```
int* recupere_note(int N)
{
    int i;
    int *POINT = (int*)malloc(N * sizeof(int));
    for (i = 0; i < N; i++)
    {
        printf("Entrez la note %d: ", i + 1);
        scanf("%d", &POINT[i]);
    }
    return POINT;
}
```

Listing 3.1 – Recupère notes

Deuxième fonction

On écrit ensuite trois fonctions :
— La note maximale du devoir SR01,
— La note minimale du devoir SR01,
— La moyenne des notes du devoir SR01.

```
int maxn(int *tab, int N)
{
    int i;
    int max = tab[0];
    for (i = 0; i < N; i++)
    {
        if (tab[i] > max)
        {
            max = tab[i];
        }
```

```
    }
    return max;
}
```

Listing 3.2 – Note maximale

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

Listing 3.3 – Note minimale

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

Listing 3.4 – Moyenne

3.1.1 Fonctions suplementaires utiles

La première fonction que j'ai implémenté en plus est max2 qui permet d'avoir le maximum d'un tableau mais aussi la position du maximum dans le tableau.

```
int* max2(int *tab, int N )
{
    int i;
    int *max;
    max = (int*)calloc(2, sizeof(int));
    max[0] = tab[0];
```

```
    max[1] = 0;
    for(i = 0; i < N; i++)
    {
        if (tab[i] > max[0])
        {
            max[0] = tab[i];
            max[1] = i;
        }
    }
    return max;
}
```

Listing 3.5 – max2

La deuxième fonction que j'ai implémenté est la fonction zero qui permet d'avoir le nombre de zéros dans un tableau et leur position. En effet cette fonction sera utile lorsqu'on va tracer un graphique car les zeros sont traités différemment des autres chiffres.

```
int* zero(int *tab, int N)
{
    int i;
    int *zero;
    int tp=0;
    zero = (int*)calloc(1, sizeof(int));
    zero[0] = 0;
    for (i = 0; i < N; i++)
    {
        if (tab[i] == 0)
        {
            tp++;
            zero[0]++;
            zero = (int*)realloc(zero, tp * sizeof(int));
            zero[tp] = i;
        }
    }
    return zero;
}
```

Listing 3.6 – zero

3.1.2 Troisième Fonction

Il faut maintenant implémenter une fonction notes qui sera la repartition des notes de 0 à 60. Cela formera l'axe des abscisses de notre graphique.

```
int* notes(int *tab, int N)
```

```
{
    int i;
    int *NOTES = (int*)malloc(7 * sizeof(int));
    for (i = 0; i < N; i++)
    {
        if (tab[i] >= 0 && tab[i] <= 9)
        {
            NOTES[0] += 1;
        }
        else if (tab[i] >= 10 && tab[i] <= 19)
        {
            NOTES[1] += 1;
        }
        else if (tab[i] >= 20 && tab[i] <= 29)
        {
            NOTES[2] += 1;
        }
        else if (tab[i] >= 30 && tab[i] <= 39)
        {
            NOTES[3] += 1;
        }
        else if (tab[i] >= 40 && tab[i] <= 49)
        {
            NOTES[4] += 1;
        }
        else if (tab[i] >= 50 && tab[i] <= 59)
        {
            NOTES[5] += 1;
        }
        else if (tab[i] >= 60 && tab[i] <= 69)
        {
            NOTES[6] += 1;
        }
    }
    return NOTES;
}
```

Listing 3.7 – notes

3.1.3 Fonction graphique

On peut alors implémenter notre fonction graphique en nuage de points à l'aide des fonctions implémentées précédemment.

```
void graph_nuage(int *note)
{
    int* cp = (int*)calloc(7, sizeof(int));
```

```
    int *max;
    int tp[2];
    int l;
    for(int i = 0; i < 7; i++)
        cp[i] = note[i];
    int* zeros = zero(note, 7);
    int nbr = max2(cp, 7)[0];
    tp[0]=0;
    for(int i=0;i<nbr-2;i++)
    {
        l = 1;
        max = max2(cp, 7);
        while(max[0] < tp[0]-l)
        {
            printf("\n%d >", tp[0]-l);
            l++;
        }
        if (tp[0] == max[0])
        {
            for(int i = tp[1]; i < max[1]-1; i++)
                printf("             ");
            printf("     o     ");
        }
        else
        {
            printf("\n%d >", max[0]);
            for(int j = 0; j < (max[1]); j++)
            {
                printf("          ");
            }
            printf("     o");
        }
        cp[max[1]] = -1;
        tp[0] = max[0];
        tp[1] = max[1];
    }
    printf("\n   ");
    int k = 0;
    for (int i = 0; i < zeros[0]; i++)
    {
        for(int j = k; j < zeros[i+1]; j ++)
            printf("+---------");
        printf("+----o----");
        k = zeros[i+1]+1;
    }
    for(int i = zeros[zeros[0]]+1; i< 7; i++)
        printf("+---------");
```

```
51      printf("+\n");
52      printf("    0          10          20          30          40
          50          59      60\n");
53  }
```

Listing 3.8 – notes

4. Exercice 3

4.1. Introduction

Le but de l'exercice est de faire un menu qui permet à l'utilisateur de trouver des restaurants qui sont proches de lui ou qui proposent des spécialités recherchées.

4.1.1 Première chose à faire

La première chose à faire est de déclarer une structure de Restaurant qui prend en compte le nom, l'adresse, la position et la spécialité proposée.

Code langage C

```c
typedef struct
{
    char nom_restaurant[100];
    char adresse_restaurant[100];
    char position_restaurant[100];
    char specialite[100];
}Restaurant;
```

Listing 4.1 – structure restaurant

4.1.2 Deuxième fonction

Ensuite, il faut pouvoir lire dans le fichier restau.txt et savoir le nombre de restaurants qu'il y a dans le fichier.

Code langage C

```c
int lire_restaurant(char* chemin, Restaurant restaurants[])
{
    FILE* fichier = NULL;
    fichier = fopen(chemin, "r");
    if (fichier != NULL)
    {
        int i = 0;
        while (fscanf(fichier, "%[^;];%[^;];%[^;];%[^\n]",
   restaurants[i].nom_restaurant, restaurants[i].
   adresse_restaurant, restaurants[i].position_restaurant,
   restaurants[i].specialite) != EOF)
        {
            i++;
```

```
11          }
12          fclose(fichier);
13          return i-1;
14      }
15      else
16      {
17          printf("Erreur d'ouverture du fichier");
18          return 0;
19      }
20  }
```

Listing 4.2 – lire restaurant

4.1.3 Troisième fonction

On implémente ensuite une fonction qui permet d'ajouter un restaurant au fichier restau.txt.

Code langage C

```
1  void inserer_restaurant(Restaurant restaurant)
2  {
3      FILE *f;
4      f = fopen("restau.txt", "a");
5      if (f == NULL)
6      {
7          printf("Erreur d'ouverture du fichier");
8      }
9      fprintf(f, "%[^;];%[^;];%[^;];%[^\n]", restaurant.
       nom_restaurant, restaurant.adresse_restaurant, restaurant.
       position_restaurant, restaurant.specialite);
10     fclose(f);
11 }
```

Listing 4.3 – inserer restaurant

4.1.4 Fonctions utiles

Avant de finir avec les deux grosses fonctions de l'exercice, j'ai implémenté des fonctions utiles à la suite de celle déjà crées. La première est copie restaurant qui permet de copier une liste de restaurants dans un nouveau tableau.

Code langage C

```
1  void copie_restaurant(Restaurant r1[], Restaurant r2[])
2  {
```

```
    for(int i = 0 ; i < 100 ; i++)
    {
        r2[i]= r1[i];
    }
}
```

Listing 4.4 – copie restaurant

La deuxième fonction utile est trier restaurant qui permet de trier un tableau de restaurants en fonction de la distance du retaurant par rapport à l'utilisateur.

Code langage C

```
void trier_restaurant(double x, double y, Restaurant results
   [])
{
    Restaurant temp;
    for(int j=1;j<=100;j++)
    {
        for(int i=0;i<100-1;i++)
        {
            char *pos = results[i].position_restaurant;
            char *po = strtok(pos, "=");
            char *pos1 = strtok(NULL, ",");
            char *po2 = strtok(NULL, "=");
            char *pos2 = strtok(NULL, ")");
            double x_restaurant = atof(pos1);
            double y_restaurant = atof(pos2);
            double distance = sqrt(pow(x_restaurant - x, 2) +
    pow(y_restaurant - y, 2));
            char *pos3 = results[i+1].position_restaurant;
            char *po3 = strtok(pos3, "=");
            char *pos4 = strtok(NULL, ",");
            char *po4 = strtok(NULL, "=");
            char *pos5 = strtok(NULL, ")");
            double x_restaurant2 = atof(pos4);
            double y_restaurant2 = atof(pos5);
            double distance2 = sqrt(pow(x_restaurant2 - x, 2)
    + pow(y_restaurant2 - y, 2));
            if (distance > distance2)
            {
                temp = results[i];
                results[i] = results[i+1];
                results[i+1] = temp;
            }
        }
    }
```

```
}
```

Listing 4.5 – trier restaurant

Enfin la dernière fonction implémentée est la fonction afficher restaurants qui permet d'afficher les informations des restaurants présents dans un tableau de restaurants.

Code langage C

```
void afficher_restaurants(Restaurant restaurants[], int size)
{
    int i = 0;
    for (i = 0; i < size; i++)
    {
        printf("Nom du restaurant : %s", restaurants[i].
    nom_restaurant);
        printf("Adresse du restaurant : %s", restaurants[i].
    adresse_restaurant);
        printf("Position du restaurant : %s", restaurants[i].
    position_restaurant);
        printf("Sp cialit  du restaurant : %s", restaurants
    [i].specialite);
    }
}
```

Listing 4.6 – afficher restaurants

4.1.5 Recherche par distance

On implémente maintenant une fonction qui permet d'avoir la liste des restaurants qui sont proches de nous en fonction d'un certain rayon.

Code langage C

```
void cherche_restaurant(double x, double y, double
    rayon_recherche, Restaurant results[])
{
    Restaurant restaurants[100];
    int nb_restaurant = lire_restaurant("restau.txt",
    restaurants);
    Restaurant tp[100];
    copie_restaurant(restaurants, tp);
    int i = 0;
    int j = 0;
    for (i = 1; i < nb_restaurant; i++)
    {
        char *pos = restaurants[i].position_restaurant;
```

```
12          char *po = strtok(pos, "=");
13          char *pos1 = strtok(NULL, ",");
14          char *po2 = strtok(NULL, "=");
15          char *pos2 = strtok(NULL, ")");
16          double x_restaurant = atof(pos1);
17          double y_restaurant = atof(pos2);
18          double distance = sqrt(pow(x_restaurant - x, 2) + pow
    (y_restaurant - y, 2));
19          if (distance <= rayon_recherche)
20          {
21              results[j] = tp[i];
22              j++;
23          }
24      }
25  }
```

Listing 4.7 – cherche restaurant

4.1.6 Recherche par spécialité

La dernière fonction à implémenter est la fonction de recherche par spécialité. La fonction nous donne la liste des restaurants faisant la spécialité recherchées ordonnées du plus proche au plus éloigné.

Code langage C

```
1  void cherche_par_specialite(double x, double y, char *
    specialite, Restaurant results[])
2  {
3      Restaurant restaurants[100];
4      int nb_restaurant = lire_restaurant("restau.txt",
    restaurants);
5      Restaurant tp[100];
6      copie_restaurant(restaurants, tp);
7      int i = 0;
8      int j = 0;
9      for (i = 1; i < nb_restaurant; i++)
10     {
11         if (strcmp(restaurants[i].specialite,specialite) ==
    0)
12         {
13             results[j] = tp[i];
14             j++;
15         }
16     }
17     trier_restaurant(x, y, results);
18
```

```
}
```

Listing 4.8 – cherche spécialité

4.1.7 Menu des fonctionnalités

Enfin on implémente dans la focntion main un menu qui présente l'ensemble des fonctionnalités présentes pour l'utilisateur.

Code langage C

```c
int main()
{
    Restaurant restaurants[100];
    Restaurant results[100];
    int nb_restaurant = lire_restaurant("restau.txt",
   restaurants);
    Restaurant tp[100];
    copie_restaurant(restaurants, tp);
    int choix;
    printf("1- Inserer un restaurant\n");
    printf("2- Afficher les restaurants\n");
    printf("3- Afficher les restaurants dans un rayon\n");
    printf("4- Afficher les restaurants par specialite\n");
    printf("5- Quitter\n");
    printf("Votre choix : ");
    scanf("%d", &choix);
    switch (choix)
    {
    case 1:
        Restaurant r;
        printf("Nom du restaurant : ");
        scanf("%s", r.nom_restaurant);
        printf("Adresse du restaurant : ");
        scanf("%s", r.adresse_restaurant);
        printf("Position du restaurant : ");
        scanf("%s", r.position_restaurant);
        printf("Sp cialit  du restaurant : ");
        scanf("%s", r.specialite);
        inserer_restaurant(r);
        break;
    case 2:
        afficher_restaurants(restaurants, nb_restaurant);
        break;
    case 3:
        double x, y;
        double rayon_recherche;
```

```
36          printf("Entrer x : ");
37          scanf("%lf", &x);
38          printf("Entrer y : ");
39          scanf("%lf", &y);
40          printf("Entrer rayon : ");
41          scanf("%lf", &r);
42          cherche_restaurant(x, y, rayon_recherche, results);
43          afficher_restaurants(results, 100);
44          break;
45      case 4:
46          double x2, y2;
47          char specialite[100];
48          printf("Entrer x : ");
49          scanf("%lf", &x2);
50          printf("Entrer y : ");
51          scanf("%lf", &y2);
52          printf("Entrer specialite : ");
53          scanf("%s", specialite);
54          cherche_par_specialite(x2, y2, specialite, results);
55          afficher_restaurants(results, 100);
56          break;
57      case 5:
58          printf("Au revoir");
59          break;
60      default:
61          printf("Erreur");
62          break;
63      }
64      return 0;
65  }
```

Listing 4.9 – cherche spécialité