CHAZELAS Clément SR01
SAHNOUN Hassan A23

Compte rendu devoir 1

Exercice 1 :

1/ Lorsque on compile ce programme, cela nous affiche l'erreur suivante :
error: lvalue required as increment operand
Le problème vient de l'expression !-- A /++! B.
La pré-incrémentation et la pré-décrémentation nécessitent une lvalue (c'est-à-dire une valeur modifiable). Les opérateurs d'incrémentation et de décrémentation préfixés tels que ++ et -- ne peuvent pas être utilisés avec des valeurs non modifiables.
Pour corriger cette erreur on divise la tâche en étapes distinctes et on obtient le programme suivant :

```
int main() {
    int A = 20, B = 5;
    A--;
    A = !A; // Inverse la valeur de A
    B = !B; // Incrémente B
    B++;
    int C = A / B; // Effectue l'opération entre A et B
    printf("A=%d B=%d C=%d\n", A, B, C);
    return 0;
}
```

2/ Lorsque on compile ce programme, cela nous affiche :
1
c=-10 d= 2

Dans ce programme, l'expression A && B évalue à vrai (1), puisque A et B ont des valeurs différentes de zéro. Le reste de l'expression étant évalué de gauche à droite, le résultat final est 1. En raison de l'évaluation de court-circuit, C n'est pas incrémenté et D n'est pas décrémenté.

3/ Lorsque on compile ce programme, cela nous affiche :
c=4
c=3

Dans ce programme, d'abord, *++q a pour valeur -2, puis on le multiplie par *q++ qui a pour valeur -2 puisque le pointeur est post-incrémenté, on a donc bien -2*-2 =4. Ensuite, *q est 3 car q a été incrémenté lors de l'évaluation précédente.

4/ Lorsque on compile ce programme, cela nous affiche :
d=-1
q= 3

Dans ce programme, d'abord on *q est évalué comme la première valeur du tableau p, qui est 1. *q++ incrémente ensuite le pointeur q pour qu'il pointe vers le deuxième élément du tableau p (qui est -2), mais renvoie toujours la valeur précédente, c'est-à-dire 1 dans ce cas. Ensuite, *q++ incrémente à nouveau q pour pointer vers le troisième élément du tableau p (qui est 3), et renvoie la valeur précédente, qui est maintenant -2.
Maintenant, l'opération logique | est appliquée aux résultats précédents, c'est-à-dire 1 | -2, ce qui donne -1. Ainsi, la valeur de d devient -1. On affiche ensuite *q qui comme on l'a vu plus tôt pointe à présent sur 3.

5/ Lorsque on compile ce programme, cela nous affiche :
a=-7 b= 1 c= 2

Dans ce programme, on utilise l'opérateur ternaire pour construire un nouveau nombre. Dans l'opération int c = ++a && --b ? b-- : a++; ++a incrémente a à -7. Ensuite, --b décrémente b à 2. Le résultat de l'opération logique 'ET' est 1 (true) car les deux opérandes sont différentes de zéro.
Ainsi, c prend la valeur de --b, soit 2 puisque b est post-décrémenté, car l'opération 'ET' a retourné true. Par conséquent, c est égal à 2, a est égal à -7 et b = 1

6/ Lorsque on compile ce programme, cela nous affiche :
a=-4

Dans ce programme on va décaler a de 2^b bit vers la droite. ^ est l'opérateur XOR, 2^b est égal à 1. On décale donc -8 vers la droite de 1 bit ce qui correspond à -4 puisque le bit de signe est conservé.

Exercice 2 :

Structure POINTS :
On déclare notre structure POINTS, elle comporte un champ note ainsi qu'un pointeur vers la note suivante.

Fonction void note_max(POINTS* Points) :
Cette fonction va nous permettre de trouver la note maximale parmi celles entrées. On commence par initialiser un pointeur qui nous permettra de parcourir l'ensemble des notes et on initialise le maximum des notes à 0. On parcourt ensuite l'ensemble des notes en testant à chaque fois si la note actuelle est supérieure au maximum actuel. Si c'est le cas alors on remplace la valeur du maximum par la note actuelle. On affiche ensuite la note maximale.

Fonction void note_min(POINTS* Points) :
Cette fonction va nous permettre de trouver la note minimale parmi celles entrées. On commence par initialiser un pointeur qui nous permettra de parcourir l'ensemble des notes et on initialise le maximum des notes à la note actuelle. On parcourt ensuite l'ensemble des notes en testant à chaque fois si la note actuelle est inférieure au minimum actuel. Si c'est le cas alors on remplace la valeur du minimum par la note actuelle. On affiche ensuite la note minimale.

Fonction void note_moy(POINTS* Points) :
Cette fonction va nous permettre de calculer la moyenne des notes entrées. On commence par initialiser un pointeur qui nous permettra de parcourir l'ensemble des notes et on initialise le nombre de notes et la somme des notes à 0 et on déclare la moyenne. On parcourt ensuite l'ensemble des notes en additionnant la note actuelle à la somme des notes et en incrémentant le nombre de notes à chaque fois. On vérifie que n est supérieur à 0 puisqu'on ne peut pas diviser par 0 et on calcule ensuite la moyenne. On affiche enfin la moyenne.

Main :
On commence par initialiser un pointeur de type POINTS qui nous servira pour stocker les notes à NULL, un autre pointeur de type POINTS qui nous servira lors de l'ajout de notes à NULL, deux variable de type int qui nous serviront lors de l'affichage des graphiques à 0, ainsi que le tableau NOTES qui nous servira pour l'affichage des graphiques à {0,0,0,0,0,0,0}.
Nous avons décidé d'implémenter un menu afin que l'interface utilisateur soit compréhensible, facile d'utilisation, et afin de pouvoir réaliser plusieurs saisies de notes.
Si l'utilisateur souhaite entrer des notes on initialise un float qui recevra la note saisie à 0. On fait ensuite une boucle while qui s'arrêtera une fois que l'utilisateur aura fini d'entrer les notes, i.e l'utilisateur aura entré -1. Dans cette boucle après la saisie de l'utilisateur, on vérifie que la note est bien entre 0 et 60, si c'est le cas on crée un nouveau point ayant pour valeur la saisie et on vérifie si on est dans le cas du premier point ou non. Si oui le point courant devient le nouveau point, sinon c'est le point suivant qui devient le nouveau point. On met ensuite la valeur de init à et on réinitialise le tableau NOTES à {0,0,0,0,0,0,0} puisqu'on vient de réaliser une nouvelle saisie de notes.
Dans les cas 2, 3 et 4 on vérifie bien que des notes ont été entrées et si oui on effectue la fonction correspondante. Si non on en informe l'utilisateur.
Dans le cas 5 on vérifie que l'utilisateur a bien entré des notes puis si cela n'a pas été déjà fait avant, i.e init = 0, on remplit le tableau NOTES avec le nombre de chaque notes de chaque catégorie. Une fois cela fait, on parcourt le tableau NOTES pour savoir quel sera le maximum de l'axe des ordonnées et on met à 1 pour ne pas répéter cette opération avant la prochaine saisie de notes. On affiche ensuite le graphique en points. Pour ce faire on utilise deux boucles imbriquées entres elles afin d'afficher au début d'une ligne le nombre des ordonnées, puis on teste chaque case du tableau NOTES, si elle est égale au nombre des ordonnées alors on affiche un "o" sinon on affiche des espaces. On affiche ensuite l'axe des abscisses.
On utilise la même logique pour le cas 6 sauf que cette fois ci dans les boucles imbriquées si le nombre de note est supérieur ou égal au chiffre de l'axe des ordonnées on affiches "#####" sinon des espaces.
A la fin du main on libère l'espace utilisé pour la structure POINTS.

Exercice 3 :

Durant cette exercice, on définit une variable globale NB_RESTAURANT_MAX égale à 30 pour pouvoir initialiser statiquement tous les tableaux que l'on utilisera. Seulement il est tout à fait possible que ces tableaux ne comporte pas 30 restaurants (en l'occurrence dans "restau.txt" il y en a max 22) donc pour pouvoir parcourir ses tableaux on va utiliser le même principe que pour les chaînes de caractères, c'est à dire que l'on va utiliser un "Restaurant sentinelle" qu'on appelle fin et qui à pour position x,y = INT_MIN = -2147483648. On va le placer à la toute fin de chaque tableau. Ainsi on peut avoir le nombre de restaurant que l'on veut (tant que c'est inférieur à NB_RESTAURANT_MAX) et on peut parcourir chaque tableau tant que l'on ne tombe pas sur ce fameux restaurant sentinelle. On pourra modifier la valeur de NB_RESTAURANT_MAX si besoin.

Structure Restaurant:
On déclare notre structure Restaurant, elle comporte 3 tableaux de char nom,adresse et spécialité avec une taille maximale de 40 char ainsi qu'un tableaux contenant la position x,y du restaurant.

Fonction void seek_to_next_line(void) :
Cette fonction permet de vider le flux d'entrée pour éviter d'avoir des caractères non voulus lors de la saisie d'une chaîne de caractères (en particulier pour donner le nom d'un restaurant,l'adresse,les spécialités quand on veut l'ajouter à un fichier texte).

Fonction Restaurant nouveau_restaurant(char *ligne) :
Cette fonction permet de récupérer toutes les informations sur un restaurant quand il est sous le format "nom;adresse;position;spécialité" dans une chaîne de caractère à l'aide d'un while qui s'arrête à chaque fois qu'on arrive sur le caractère ";".

Fonction int lire_restaurant(const char *chemin, Restaurant (*restaurant)[ ]) :
Cette fonction permet de lire un fichier texte et de récupérer tous les restaurants à l'intérieur et de les placer dans le pointeur restaurant en utilisant la fonction fgets() sur le fichier indiqué par chemin, on utilise ensuite la fonction nouveau_restaurant sur chaque ligne pour récupérer chaque restaurant à chaque ligne..

Fonction void tri_tableau(Restaurant (*restaurant)[ ],double x,double y,int n) :
Cette fonction permet de trier les restaurant dans le pointeur restaurant en fonction de s' ils sont proche ou non des coordonnées x,y, l'int n correspond au nombre de restaurant dans le tableau. L'algorithme est un simple tri par insertion.

Fonction void afficher_restaurant(Restaurant restaurant[]) :
Cette fonction permet d'afficher une liste de restaurant dans le terminal à l'aide de printf et d'une boucle while.

Fonction void inserer_restaurant(Restaurant restaurant):
Cette fonction permet d'ajouter une restaurant au fichier "restau.txt" en utilisant fprintf().

Fonction void cherche_restaurant(double x,double y, double rayon_recherche,Restaurant (*restaurant)[ ]) :
Cette fonction permet de mettre dans le tableau restaurant tous les restaurants présents dans “restau.txt” qui sont à une distance inférieur à rayon_recherche. Pour ce faire on parcourt liste_restaurant (un tableau comportant tous les restaurants de “restau.txt” formé à l’aide de lire_restaurant) à l’aide d’un while et on vérifie à chaque etape la distance.

Fonction void cherche_par_specialite(double x, double y, char* specialite[],Restaurant (*result)[ ]):
Cette fonction permet de mettre dans le tableau restaurant tout les restaurant présent dans “restau.txt” qui ont au moins une spécialité en commun avec le tableau de chaine de caractère specialite et le trie selon leur distance au coordonnées x,y. pour ce faire on parcourt liste_restaurant (un tableau comportant tout les restaurant de “restau.txt” formé à l’aide de lire_restaurant) à l’aide d’un while et on vérifie grâce a la fonction specialite_dans_restaurant qui retourne un booléen si il ya une spécialités recherchées et qui l’ajoute au tableau pointé par result. À la fin de la boucle while on utilise tri_tableau sur result pour trier les restaurants dans l’ordre.

Fonction int specialite_dans_restaurant(Restaurant restaurant,char* specialite[ ]):
Cette fonction permet de savoir si une des spécialités présentes dans le tableau specialite est présente dans le restaurant. Pour ce faire, on vérifie si chaque spécialité du tableau spécialité est une sous-chaîne des spécialités du restaurant. Pour ce faire à chaque caractère, on compare un-à-un les caractères suivants pour vérifier s' ils sont identiques. Dès lors qu’il n’y a pas égalité on modifie la valeur d’une variable nommée Present à 0 (initialisé a 0 également) et on quitte la boucle.Sinon ils sont tous identiques alors on peut modifier Present à 1 et quitter la boucle et renvoyer true.

Main :
On commence le programme par demander à l’utilisateur ses coordonnées à l’aide de scanf et en initialisant une variable jeu à 1 et une variable réponse.
On entre ensuite dans une boucle while qui dure tant que jeu!=0.
à chaque boucle on demande à l’utilisateur de choisir parmi 4 options et de stocker le résultat dans la variable reponse.

Dans la première option, on initialise une chaine specialite et specialite_ptr et on demande au maximum 10 fois à l’utilisateur d'entrer une spécialité (Il peut écrire NULL pour quitter la boucle et se contenter de ce qu’il a écrit) ces résultat sont stockés dans specialite puis transmis à specialite_ptr pour finalement utiliser la fonction cherche_par_specialite et quitter la boucle while en mettant jeu à 0.

Dans la seconde option,on demande un rayon à l’utilisateur et on utilise la fonction cherche_restaurant.

Dans la troisième option, on demande à l’aide de fgets à l’utilisateur le nom,l’adresse et les spécialitées qu’il souhaite insérer à “restau.txt”, puis on utilise scanf pour la position x,y.
On utilise ensuite inserer_restaurant.

Dans la quatrième option, on met une sentinelle à la fin du tableau restaurant et on modifie le jeu à 0.

Après avoir quitté la boucle principale, on utilise la fonction afficher_restaurant pour afficher le résultat de la ou les recherches.