RAPPORT DEVOIR 1 SR01

OUEDRAOGO Taoufiq

BRENA--LABEL Yannis

EXERCICE1

Programme 1 : retourne 1. Car **&&** est prioritaire devant **||**.

Donc respectivement :

a&&b = 1

!0 && c++ = 1

(**!0 && c++) && !d++ = 0** , car la post-incrémentation fera incrémentée d et c, après l'opération en cours d'exécution. C'est pour cela que dans la ligne suivante on retrouve **c=-9** et **d=3**.

On aura donc en dernière opération **(a&&b) || (!0 && c++ && !d++)** = **1 ou 0** = **1**.

Programme 2 :

q pointe sur le 1er élément du tableau = 1

***++q**q++** = **4.** Car **++ prioritaire devant *.

***++q =-2**. Donc avec la pré-incrémentation **on pointera sur -2** (car la pré-incrémentation fait incrémenter l'élément avant de l'utiliser dans l'opération en cours)**.**

***q++=2** pour la post-incrémentation **q pointe toujours sur -2** et c'est seulement après l'opération que **q** sera incrémenté et **pointera sur 3.**

L'opération est donc **-2 * -2=4.**

Donc ***q=3.** Car q a subi une pré-incrémentation et une post-incrémentation à la ligne précédente.

Programme 3 :

c = !--a/++!b = impossible. Car **++!b = impossible**

!--a = 0. Car on prend le complémentaire de **--a=19.**

++ !b = impossible car **!b n'est pas une valeur pouvant recevoir une incrémentation.**

Programme 4 :

a>> = 2^b = a>>2^b = -4

Car **^ est prioritaire devant >>.**

2^b=1 car ou exclusif = 1 si les éléments sont différents.

a>> = 2^b sera donc un décalage à droite de 1 → une division par 2**1 d'où **a=-4**

Programme 5 :

q pointe sur le 1er élément du tableau= **1**.

d= *q&*q++|*q++ = 1|-2 = -1. Car **& prioritaire devant |.**

***q&*q++** le **et logique** -> = 1. **q pointera après cette opération sur -2.**

1|-2 ou logique **=-1**.

q= 3 car q a subi **2 post-incrémentation dans l'opération précédente** et l'incrémentation est effective après cette dernière.

EXERCICE2

Q1 :

- On déclare un entier n correspondant au nombre de notes entrées qui sera initialisé à 0.
- Puis à chaque saisi de note, n sera incrémenté. On considère qu'une note est comprise entre 0 et 60 et que le nombre de notes doit être égal au nombre d'élèves.
- Chaque note sera affectée à un élément du tableau POINTS.

Q2 :

- On déclare 2 entiers maxi et mini qui seront respectivement la note maximale et minimale.
- On initialise ces entiers avec la première note qui est le 1er élément de POINTS.
- Puis en parcourant chacun des éléments de POINTS on compare séparément la note avec chacun des 2 entiers déclarés et on fait les changements nécessaires.
- Si la note <mini alors mini prend la valeur de la note et on passe à la note suivante.
- Si la note >maxi alors maxi prend la valeur de la note et on passe à la note suivante.

- Pour la moyenne, on initialise d'abord à 0 un flottant somme=somme de toutes les notes, car on anticipe le fait qu'une moyenne n'est pas forcément entière.

- Puis à chaque parcourt de note on la somme avec la valeur de la somme précédente

- A la fin du parcourt, on a la somme totale et pour obtenir la moyenne des notes, il restera juste à diviser par le nombre de notes = n.

Q3 :

- On déclare un tableau NOTES de n entiers. Où chaque entier représente le nombre de notes correspondant au descriptif de l'énoncé.
- On initialise à 0 chaque élément du tableau car au départ il n'y a pas de notes correspondantes et par la suite on implémentera si besoin grâces aux différentes boucles.
- Puis on affiche les éléments de NOTES pour avoir une vision plus grande pour faciliter la validation des représentations en nuages et en bâtons.

Q4 :

- On déclare un entier MAXN initialisé avec NOTES[0]. MAXN =plus grande valeur de NOTES.
- Pour la recherche même procédure que note maximale ref q2.
- Pour la représentation en nuage de points, pour chaque valeur prise par MAXN on va placer les points correspondant aux catégories dont le nombre de notes vaut MAXN.
- Commencer par MAXN facilite la construction du nuage de points du haut vers le bas. On aurait pu utiliser une boucle for de NMAX->0 inclus. 2 cas se présentent.
- 1er cas : MAXN !=0 alors on place simplement les ordonnés (à l'aide d'une boucle for) ainsi que les points correspondants en repérant les catégories dont le nombre de notes = MAXN. Si c'est le cas (cad si NOTES[i]==MAXN) alors on place ° ; sinon on laisse des espaces pour ne pas fausser la représentation.
- 2e cas : MAXN==0. Dans ce cas, on place les abscisses ainsi que les catégories de notes vides. Pour chaque catégorie (cad chaque élément de NOTES), si la catégorie !=0 alors on complète les abscisses avec ……| sinon on insère un point ° dans la représentation des abscisses.
- A la fin de ce 2e cas, on délimite les catégories … |0-9|10-19|…

Q5 :

- La représentation en bâtons est assez similaire à celle des nuages de points.
- Les 2 seules différences sont les suivantes : on ne représentera pas les catégories vides(cad NOTES[i]=0) et pour chaque catégorie non vide, on va représenter les bâtons non seulement pour la valeur MAXN correspondante (cad le nombre de notes de la catégorie) mais aussi pour les valeurs inférieures non nulles. Par exemple, si NOTES[2]=3 on va placer des bâtons pour les valeurs 3, 2 et 1.

EXERCICE3

Q1 :

- Représentation sous forme de liste chainée. Chaque voiture aura les caractéristiques données dans l'énoncé et on aura pour chaque voiture un pointeur suivant, pointant sur la voiture suivante.
- Les éléments de la structure ont leur 1ère lettre en majuscule pour éviter les confusions avec les différentes valeurs des voitures entrées dans la suite de l'exercice.

Q2 :

- On crée une fonction Voiture créer_voiture() permettant d'allouer de la mémoire pour une structure Voiture et qui renvoie un type Voiture dont la mémoire a été dynamiquement allouée. Ainsi on l'utilisera dans la fonction init n fois.
- Un parc sera donc représenté sous forme de liste chaînée.
- On crée un pointeur qui au départ sera la 1ère voiture.
- Pour chaque voiture on va demander les caractéristiques puis passer à la voiture suivante.
- Lorsqu'on aura atteint le nombre voulu = n, on ferme la liste chaînée.

Q3 :

- On crée une fonction void *existe(char *matricule, Voiture *voitures, int n) qui renvoie 0 si la voiture (dont l'immatriculation a été entré en argument) n'existe pas (cad le matricule ne correspond à aucun matricule de voiture dans le parc). Sinon, elle renvoie le pointeur sur cette voiture. Pour cela, on va déclarer un pointeur pointant d'abord sur la tête du parc.
- Pour louer une voiture on va d'abord vérifier si elle existe(cad la fonction existe renvoie le pointeur) . Dans ce cas on pointera directement sur la voiture concernée.
- Ensuite on vérifie son état (0 si déjà en location et 1 si disponible). Donc si l'état de la voiture =0 on ne pourra pas louer et si l'état =1 on pourra louer la voiture et donc l'état sera passé à 0.

Q4 :

- Pour retourner une voiture on va d'abord vérifier si elle existe(cad la fonction existe renvoie le pointeur) . Dans ce cas on pointera directement sur la voiture concernée.
- Ensuite on vérifie son état (0 si déjà en location et 1 si disponible). Donc si l'état de la voiture !=0 on affiche une erreur parce que la voiture est censée être en location. Sinon on demande la distance parcourue et on l'ajoute au kilométrage total de la voiture et la voiture devient disponible (son état passe de 0 à 1).

Q5 :

- Pour renvoyer l'état d'une voiture on vérifie que la voiture existe déjà.

- Si c'est le cas, on récupère le pointeur sur la voiture et on affiche les différentes caractéristiques.

Q6 :

- Pour afficher l'état du parc on initialise à 0, 2 entiers représentant le nombre de voitures disponibles et en location et un réel = somme des distances parcourues par toutes les voitures.
- Pour parcourir toutes les voitures du parc on déclare un pointeur qui pointera d'abord sur la tête du parc. Puis il passera à l'élément suivant après les opérations.
- Pour chaque voiture on incrémente soit le nombre de voiture disponible soit le nombre de voiture en location et on ajoute le nombre de km au compteur général.
- A la fin on affiche les données et pour obtenir le kilométrage moyen on divise juste la somme des distances parcourues par le nombre de voitures.

Q7 :

- Pour sauvegarder les voitures, on utilise un pointeur sur FILE. Ce pointeur contiendra l'adresse en mémoire du début du fichier.
- Ensuite on ouvre le fichier en mode écriture en binaire (''wb''). Et on vérifie qu'il a bien été ouvert.
- On saisit dans le fichier les différents éléments à l'aide de la commande fwrite.
- On ferme le fichier à la fin.

Q8 :

- On crée une fonction afficher_menu() affichant le menu et qui sera appelée après chaque action.
- On alloue dynamiquement une mémoire de n structures Voiture à un pointeur sur la liste chaînée.

BONUS

- Pour récupérer les voitures, on utilise un pointeur sur FILE. Ce pointeur contiendra l'adresse en mémoire du début du fichier.
- Ensuite on ouvre le fichier en mode lecture en binaire (''w*rb''). Et on vérifie qu'il a bien été ouvert. (cad fread(…)==1)
- Grâce à un pointeur sur la structure Voiture on pourra récupérer les différents champs saisit dans le fichier les différents éléments à l'aide de la commande fwrite.
- On ferme le fichier à la fin.

PS : Après chaque allocation de mémoire, la restitution a bien été effectuée.