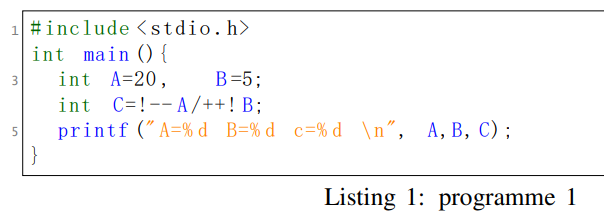
SR01 – Devoir 1

# **Exercice 1**

# Programme 1



Le programme n’a pas l’air de fonctionnner.  
mais basiquement je dirais qu’on fait

-Mettre A à 20 et B à 5

-Faire –A donc renvoie 19 et A = 19

- 19 est un valeur différente de 0 donc considérée comme true

-Donc son inverse !19 revient à !true donc à false

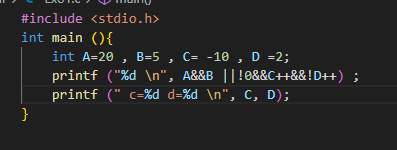
-Dans la même logique !18 donne false donc 0

-Donc ++ !B donne 1 mais n’augmente pas B car il augmente seulement l’inverse de B

-On fait donc 0/1 donc 0 donc C = 0

Donc A = 19, B = 5 et C = 0

# Programme 2



*On obtient :*

*1*

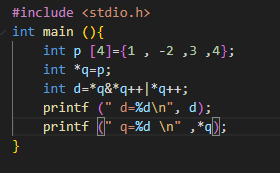
*c=-10 d=2*

Ce qui est logique car :

-On met A à 20, B à 5, C=-10, D=2

On fait A&&B or A et B sont à true car différents de 0 donc ca renvoie true donc vu qu’après c’est un || (OR), pas besoin de regarder ce que ca donne , ca donnera forcément true donc 1 (Ce qu’on voit bien). Le compilateur, dans un soucis de performance va donc arreter de lire ce qu’il y a après le || et c’est donc pour ca que le C++ et le !D++ ne sont pas executés

# Programme 3



*On obtient :*

*d=-1*

*q=3*

Voici pourquoi :

-On créer une liste p = [1, -2, 3, 4]

-On créer un pointer q vers p

-On va commencer à calculer d :

-\*q&\*q++ donne donc \*q&\*q, or un (ET) binaire sur sois meme reviens à sois meme donc \*q

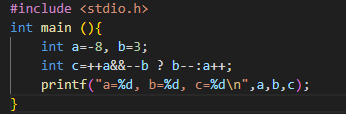
-On a donc \*q|\*(q+1) donc 1|-2 or :

- 1 en binaire = 0001

- -2 en binaire = 1110 donc avec l’opérateur | (OU) on a 1111 donc -1

Pour ce qui est de q , on l’a postincrémenté deux fois donc \*q = \*(qDuDebut +2) = p[2] = 3

# Programme 4



*On obtient :*

*a=-7, b=1, c=2*

On va d’abord chercher à savoir le résultat de la condition :

-On a a qui est préincrémenté et b qui est présoustrait. Donc 7&&2 ce qui donne donc 1&&1 donc 1 donc on effectue B—mais pas a++

Donc

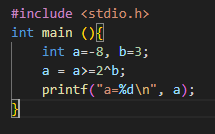
- La variable a a juste une préincrémentation donc a=-8+1=-7

-La variable b a une présoustraction et une postsoustraction donc b=3-1-1=1

-C vaut donc b— (après execution du --b) donc c=b(du départ)-1=3-1=2

# Programme 6

Ce programme ci non plus ne fonctionne pas  
je l’ai donc adapté à ma sauce



*On obtiens :*

*a=3*

Ceci s’explique assez facilement :

A=-8 or -8 >= 2 donne false donc 0

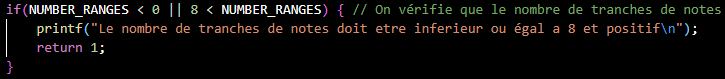
0^b = 0XOR3= 00XOR11 = 11 = 3

# **Exercice 2**

Pour cet exercice, nous nous sommes bien amusés et nous avons donc entrepris de faire un système flexible. Il est en effet possible de choisir jusqu’ou on va avec les tranches et donc de ne plus se limiter à forcément 60.  
Pour jouer sur ce paramètre, il suffit d’augmenter le nombre de tranches (aussi appellées rangées) :



On peut ainsi aller de 1 tranche (donc de 0 à 10) à 8 tranches (de 0 à 80). Aller au-delà ne fait pas planter le programme mais provoques quelques bugs d’affichage. Pou éviter cela nous avons mis en place un processus de vérification au tout début du programme :



Nous avons aussi écrit le programme de manière à ce que l’utilisateur puisse choisir pour combien d’étudiants il souhaite rentrer les notes :



## Vocabulaire du projet

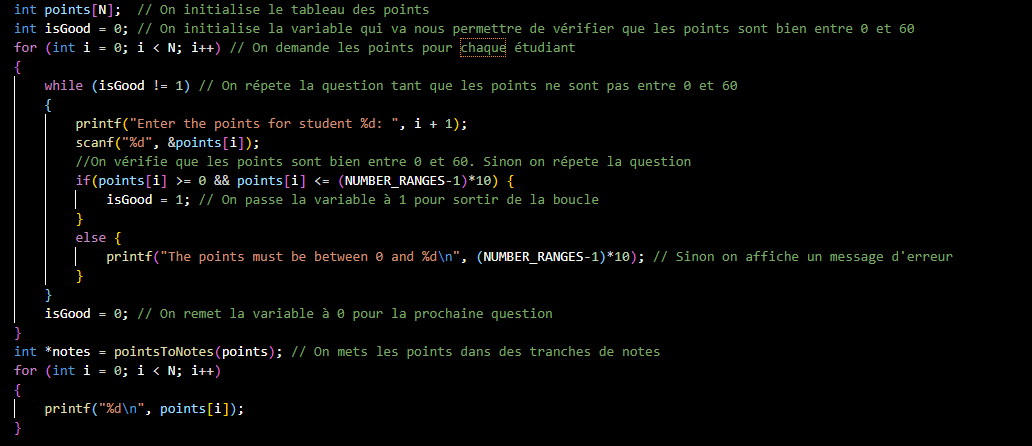
Point = note de l’élève rentré par le professeur (allant donc de 0 à NUMBER\_RANGES \* 10) (ex : 48)

Note = nombres de notes se trouvant dans la tranche de point correspondant (ex : 2 points se trouvent entre 10 et 19)

Etage = représentation graphique d’où l’on se trouve. L’étage est souvent associé à une valeur que peut prendre une note. (ex : 2 donc on affiche « \* » à toutes les notes ayant pour valeur 2)

## 1. 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

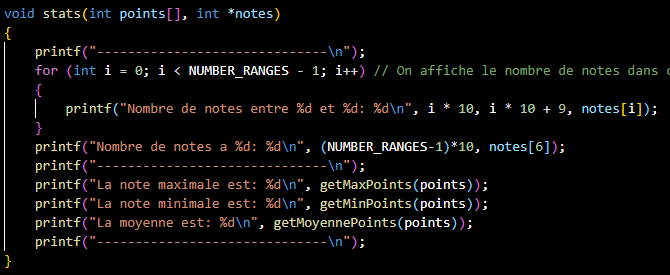


Pour cette partie nous avons mis en place un système permettant de nous assurer que l’utilisateur utilise correctement l’outil et ne commette pas de fautes lors de la saisie.  
On vérifie que l’entrée de l’utilisateur se trouve entre 0 et la valeur maximale (qui se base sur le nombre de tranches) car une note dépassant la tranche maximale (par exemple 76 alors que dans l’énoncé on fait des tranches jusqu’à 60, cela ferais que le point ne s’afficherais pas et risquerait de provoquer des problèmes dans le code).

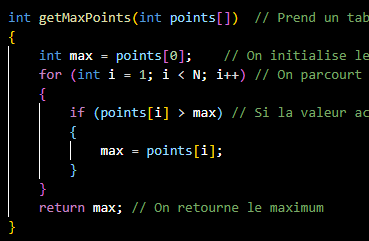
Nous faisons cette vérification grâce à un boucle while associé à un variable (isGood) qui permet à l’utilisateur de pouvoir corriger son erreur tout recommencer.

## 2. Écrire des programmes pour rechercher et afficher :

Pour faire de la place dans le main(), nous avons déplacer toutes les fonctions tournant autours de l’analyse statistique dans la fonction stats()



## La note maximale du devoir SR01



On commence par le premier élément du tableau et on considère que c’est le plus grand du tableau.  
Puis on le compare avec le deuxième élément. Si le deuxième est plus grand alors on va considérer que c’est le deuxième le plus grand.  
Sinon on continue à considéré que le premier est le plus grand.  
On continue avec le troisième, quatrième … jusqu’à la fin.

**Pourquoi le premier élément ?**

On commence par le premier car si on a un tableau à un élément ca nous évite un segmentation fault (si on prennait un autre index) ou une comparaison à null (Si on commencais à max = null) ou d’avoir une réponse fausse (Si on prennait max = 0 mais que tous les nombres étaient négatifs)

Nous avons utilisé la même technique pour la fonction getMinPoints() mais changé le sens de la comparaison et le nom des variables

### La moyenne des notes du devoir SR01.

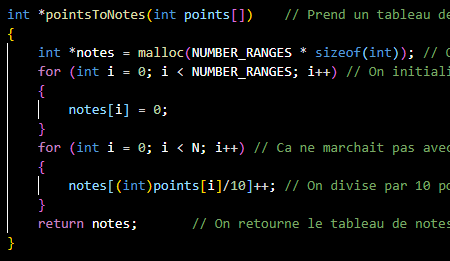
Pour la moyenne, nous faisons la somme de toutes les valeurs du tableau puis nous le divisons par le nombre d’éléments du tableau (donc le nombre d’étudiants)

## 3. A partir des POINTS des étudiants, établir un tableau NOTES de dimension 7 qui

## est composé de la façon suivante:

Une fois les notes rentrées, nous les organisons dans un tableau de point grâce à la fonction pointsToNotes() et nous les affichons pour vérifier qu’il n’y a pas d’erreurs.

Fonctionnement de la fonction pointsToNotes() :



**Entrée :** La liste des points rentrés par l’utilisateur

**Sortie :** le tableau « notes » regroupant chaque point par tranche (sous forme d’histogramme)

Ne connaissant pas à l’avance le nombre de tranches, nous avons choisit une méthode dynamique utilisant malloc afin de créer à chaque fois un tableau notes de la bonne taille (le nombre de tranches multiplié par la taille d’un élément)

On les initialise tous à 0 car parfois l’allocation dynamique provoque des nombres aléatoires due à des résidus dans la mémoire.

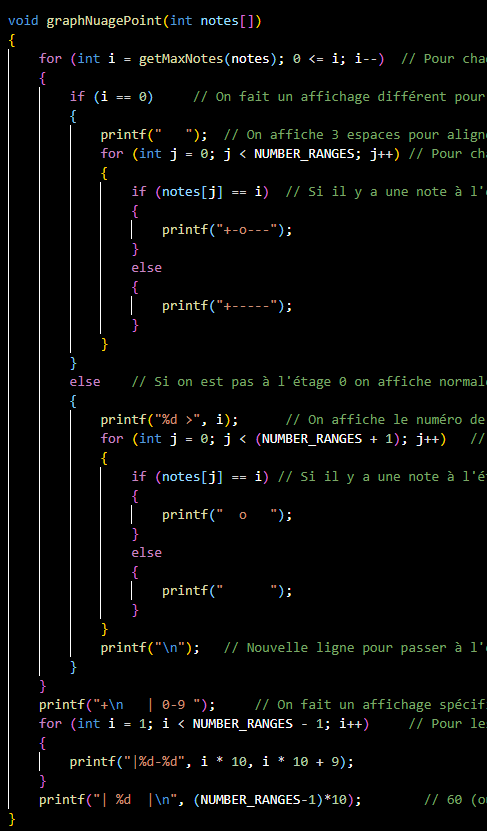
Afin de ranger les notes dans la bonne case, nous nous basons sur l’index. En effet, 48 doit être placé à la case d’index 4 car la case d’index 4 s’occupe de la tranche allant de 40 à 49. Pour obtenir la dizaine nous faisons donc tout simplement une division par 10. Nous rajoutons un ++ pour incrémenter la case correspondante, ce qui permet de créer l’histogramme.

4. Établir un graphique en nuage de points représentant le tableau notes. Utilisez le

symbole ’o’ pour représenter le point dans le graphique et affichez le domaine des

notes en dessous du graphique.

Nous avons rangé tout le code dans la fonction graphNuagePoint()



Le graph étant responsive, il doit s’adapter pour que toutes les notes rentrent.  
Pour cela, nous devons connaitre la valeur maximale que va atteindre les valeurs du tableau notes.  
J’ai donc fait une fonction getMaxNotes() qui fonctionne à peu près comme getMaxPoints() mais utilise NUMBER\_RANGES au lieu de N pour connaitre le nombre de valeurs dans le tableau.

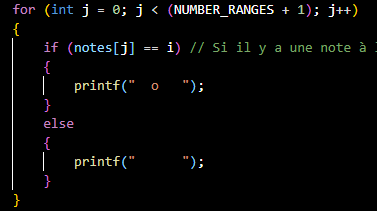
**Affichage de l’axe vertical (valeurs de notes) :**

**Pour la plupart des étages (i != 0) :**

On affiche l’étage où nous nous trouvons



On parcours toutes les notes et si la valeur de la note vaut la même valeur que l’étage, alors on affiche « o », sinon on affiche «  «



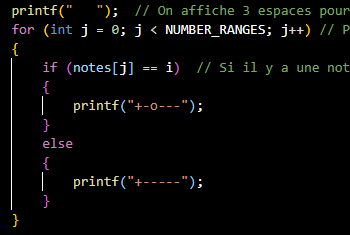
Puis on passe à l’étage suivant :



**Pour l’étage 0 :**

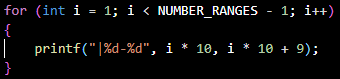
On le détecte avec 

L’étage 0 est particulier car il est rempli de «-«  au lieu d’espaces et nous n’affichons pas le numéro de l’étage



**Affichage de l’axe horizontal (notes):**

**Pour la plupart des tranches :**



On affiche leur bornes supérieures et inférieures grâce à une multiplication par 10 (et une addition par 9 pour la borne max), ça nous permet d’obtenir par exemple « |50-59| «

**Problème de 0-9 :**



« 0-9 » n’a pas le même nombre de caractère que ceux des dizaines. On le fait donc à ma main et on en profite pour ajouter le + et le retour à la ligne qu’il fallait mettre juste avant.

**Le problème de 60 (ou autre dernier nombre) :**

Tout comme 0-9, le dernier nombre pose problème car il a un affichage particulier.  
On résout le problème en le faisant « à la main »



## 5. Établir un graphique en bâtons représentant le tableau NOTES. Utilisez les symboles

## ou autres pour la représentation des bâtons et affichez le domaine des notes en

## dessous du graphique.

On fait la même chose que pour graphNuagePoint()

Ce qu’on change c’est qu’on affiche « #### » au lieu de « o » et que l’on l’affiche si la valeur de la note est supérieure ou égale à la valeur de l’étage. Comme ca on « remplis » les valeurs d’en dessous avec « #### »

On simplifie aussi l’étage 0 car quand on a une note à 0 on ne l’affiche juste pas, contrairement au graphNuagePoints()

