Directive #define – Opérations sur les bits – Tableaux fixes à une dimension

Exercice 1

Définir la « macro » suivante : `#define carre(x) x * x`
Déclarer une variable « Nb » entière et égale à 5 et afficher la valeur de carre(Nb).
Afficher en suite le carre(Nb+1), quel est le résultat obtenu ? D'où vient le problème à votre avis ?
Proposer une modification de la macro pour qu'elle fonctionne.

Exercice 2

Rappel :

```
// Même si un entier est représenté la plupart du temps en base 10 dans le code C : ex 65535,
// la représentation interne des données est réalisée en binaire, c'est-à-dire que c'est une suite
// de 0 et de 1 qui est manipulée et stockée.
```

Un premier exercice simple va consister à regarder ce que valent les 32 bits d'un entier non signé ayant la valeur 2868838400 en base 10, soit AAFF0000 en hexadécimal. En commençant par le bit de poids faible, afficher ON quand le bit vaut 1 et OFF quand il vaut 0. Présenter le résultat sous la forme de colonnes correctement alignées :

- Afficher tout d'abord le nombre d'octets utilisés pour représenter le nombre entier.
- Afficher le nombre de bits utilisés pour représenter le nombre entier.
- A l'aide d'une boucle, d'un opérateur de décalage à droite >> ou à gauche << et d'un opérateur de bit ET (&), tester la valeur de chaque bit et afficher l'état correspondant (ON/OFF).
- L'affichage se présente comme suit :

```
…
bit  8 : OFF
bit  9 : OFF
bit 10 : OFF
bit 11 : OFF
…
```

- Vérifier à l'écran que tous les bits sont affichés et que l'état correspondant est exact.
- Le programme affiche un message de courtoisie avant de se terminer: Bye !

Exercice 3

Rappel :

```
// Un tableau est une collection de données du même type.
// la taille du tableau est définie à la compilation et ne peut pas être modifiée.
// Un tableau peut avoir plusieurs dimensions.
// Chaque élément du tableau est identifié par un indice allant de 0 à nb éléments -1
// En C, le nombre d'éléments que peut contenir un tableau n'est pas fourni par le langage
// il est toutefois possible de connaitre le nombre d'OCTETS qu'il occupe en mémoire (sizeof)
// ex: // déclaration d'une variable de type tableau d'entiers :
//     int temperature[12];
//     // écriture de valeurs dans le tableau :
//     temperature[0] = 25;
//     temperature[11] = 8;
//     // lecture de valeurs du tableau :
//     ecart_temperature = temperature[11] – temperature[0];
```

Cet exercice a pour but de calculer une moyenne de notes entrées au clavier par l'utilisateur. Dans un premier temps, on choisit de ne pas mémoriser les notes. Puis en vue de produire un état des notes, le programme est complété et dispose d'un tableau. Il sera donc possible de procéder à différents traitements sur les notes sans être obligé de les saisir à nouveau.

- Ecrire un programme qui demande à l'utilisateur d'entrer des notes comprises entre 0 et 20 inclus avec une précision de deux chiffres après la virgule. Le message est " Entrer la note no *[indice]* : "
- Une valeur en dehors de ces limites provoque la question "Elève absent ? ou voulez-vous arrêter la saisie des notes ? A/O/N". Sans action particulière, la saisie s'arrête après l'entrée de la 30ème note. Si la saisie des notes reprend et qu'il ne s'agit pas d'une absence, il faut ressaisir et prendre en compte la note en cours.
- A l'issue de la saisie, le programme affiche le nombre de notes valides, le nombre d'absences, la moyenne des notes valides, la plus petite et la plus grande note valide.
- Déclarer une variable *note* de type tableau et modifier le programme pour y ranger les notes. Les notes des élèves absents sont représentées par la valeur -1 dans le tableau de notes et n'entrent pas en compte dans le calcul de la moyenne.
  Astuce : Utiliser une variable qui indique le niveau de remplissage du tableau de notes.
  Contrainte : Utiliser une directive #define NBMAXNOTES 30 pour définir une constante de taille du tableau note. Dans la suite du code, on ne doit jamais retrouver la valeur 30 autrement que sous la forme NBMAXNOTES.
- Initialiser le tableau de notes avec la valeur arbitraire -2.
- A l'issue de la saisie et en plus des informations précédemment affichées, calculer et afficher l'écart-type avec une précision de deux chiffres après la virgule.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

- Afficher le tableau de notes.
- Afficher un classement des notes par valeurs décroissantes.
  Idée : créer une copie du tableau de notes et un tableau d'index (les indices de chaque note)

qui sera trié en même temps que la copie du tableau *note*. Il ne faut pas trier le tableau de notes lui-même.
L'affichage se fera de la manière suivante :

```
Rang :          No note :       Valeur note :
1               10              18.00
2               7               17.00
3               12              16.00
3               1               16.00
4               4               15.25
..
```

Exercice 4

A partir d'une adresse IPv4 w.x.y.z/*n* en notation CIDR (codée sur 4 valeurs en base 10 comprises entre 0 et 255 / un masque *n* codé sur un entier), afficher l'adresse de réseau et l'adresse de *broadcast* du réseau correspondant.
Utiliser les opérateurs binaires de décalage à droite ou à gauche (<<, >>) ainsi que les opérateurs logiques de bits (~ , | , &).

En ayant initialisé les variables suivantes pour l'adresse 192.168.129.10 / 24:

```
MyAddress_w = 192;
MyAddress_x = 168;
MyAddress_y = 129;
MyAddress_z = 10;

IPv4MaskLength = 24;
```

Lorsque l'adresse IPv4 est représentée sur 32 bits, l'adresse du réseau auquel elle appartient se calcule comme suit (les bits de droite qui concernent la machine sont mis à zéro) :

```
Adresse_reseau = Adresse_machine AND masque
```

L'adresse de broadcast, qui permet de s'adresser à l'ensemble des machines d'un même réseau locale est calculée comme suit :

```
Adresse_broascast = Adresse_reseau OR NOT(masque)
```

Le programme affiche :

```
adresse      IPv4 = 192.168.129.10 / 24
adresse du reseau = 192.168.129.0 / 24
adresse broadcast = 192.168.129.255
```