◊ Editez le programme du point 2 de l'exercice 2. Modifiez-le pour qu'il affiche en fin de programme le nombre de valeurs saisies et la somme.

Solution

Par rapport au programme du point 2 de l'exercice 2 (sans contrôle que valeurs saisies sont entre 0 et 20) il n'y a qu'à rajouter un compteur de valeurs à l'aide d'une variable judicieusement appelée cpt....

```
#include<stdio.h>

int main(void)
{   int N,somme,x;
    int cpt = -1 ;

    somme  = 0 ;
    do
    {   printf("Entrez une valeur : ");
        scanf("%d", &x  ) ;
        somme += x ;

        cpt++ ; // cpt = cpt +1

    } while ( x != 99 ) ;

    printf("La somme est : %d \n", somme-99 );
    printf("Le nombre de valeurs saisies est : %d \n",
cpt );

    return 0;
}
```

Analyse du programme :

Pas de grosses différences avec le programme précédent hormis l'ajout de cpt.

Noter l'initialisation de cpt à -1. Cette astuce sert à compenser le fait que pour sortir de la boucle il faudra entrer 99. Par conséquent le compteur comptera une valeur de trop. De plus on compte en fin de boucle. Du coup cpt contient le numéro de la prochaine valeur à saisir et non le numéro de la valeur que l'on vient de saisir.

Il y a donc un décalage de 2 entre le nombre de valeurs comptées par cpt et le nombre réel de valeurs saisies. Le listing de débogage du programme avec un point d'arrêt avant l'instruction cpt++ et après cette instruction permet de suivre l'évolution de la variable cpt.

```
Entrez une valeur : 2
    13          cpt ++ ;
1: cpt = -1
    15      } while ( x != 99 ) ;
1: cpt = 0

Entrez une valeur : 12
    13          cpt ++ ;
1: cpt = 0
    15      } while ( x != 99 ) ;
1: cpt = 1

Entrez une valeur : 18
    13          cpt ++ ;
1: cpt = 1
    15      } while ( x != 99 ) ;
1: cpt = 2

Entrez une valeur : 99
    13          cpt ++ ;
1: cpt = 2
    15      } while ( x != 99 ) ;
1: cpt = 3
```

```
La somme est : 32
Le nombre de valeurs saisies est : 3

Program exited normally.
```

En bleu, les messages affichés par le programme, en vert les réponses de l'utilisateur, en gris clair, les lignes en début desquelles la valeur de cpt est affichée (ligne 13 avant l'incrément, ligne 15 après l'incrément), en rouge les valeurs prises par cpt.

Enfin, sans le décalage de -1 imposé à l'initialisation, cpt vaudrait 4 en fin de programme.

La valeur prise par cpt lors des saisies : cpt vaut -1 puis 0 et enfin 1.

Si l'on pré-compte les valeurs en plaçant l'instruction cpt++ avant la saisie :

```
    do
    {   cpt++ ;
        printf("Entrez une valeur : ");
        scanf("%d", &x  ) ;
        somme += x ;
    } while ( x != 99 ) ;
```

les valeurs saisies auront les numéros 0,1 et 2.

Cette remarque est importante si l'on veut stocker les valeurs saisies dans une variable dimensionnée.

◊ Calculer la valeur moyenne et affichez-là. Que constatez-vous ?

La valeur moyenne d'une série de valeur est donnée par :

$$\bar{x} = \frac{1}{n} \sum_{k=1}^{n} x_k$$

Il suffit donc de diviser la somme pour le nombre de valeurs saisies pour obtenir la moyenne, d'où le programme :

Solution

```
01 #include<stdio.h>
02
03 int main(void)
04 {   int N,somme,x;
05     int cpt = -1 ;
06
07     somme  = 0 ;
08     {   printf("Entrez une valeur : ");
09         scanf("%d", &x  ) ;
10         somme += x ;
11
12         cpt ++ ;
13  } while ( x != 99 ) ;
14
15     printf("La somme est : %d \n", somme-99 );
16     printf("Le nombre de valeurs saisies est : %d \n",
cpt );
17     printf("La moyenne des valeurs saisies est : %d
\n", (somme-99) / cpt );
18
19     return 0;
20 }
```

Analyse du programme :

Pas grand chose à dire sur ce programme juste l'ajout du dernier printf, et le fait que les résultats retournés sont faux, ainsi que l'exemple d'exécution suivant le montre :

```
Entrez une valeur : 11
```

```
Entrez une valeur : 11
Entrez une valeur : 10
Entrez une valeur : 99
La somme est : 32
Le nombre de valeurs saisies est : 3
La moyenne des valeurs saisies est : 10
```

Or la moyenne des trois valeurs 11, 11, 10 est 10,6666 et non 10.

On pourrait croire que l'erreur est due au fait que l'on utilise un champ %d pour afficher le calcul. Modifions le programme en rajoutant une variable au début de la fonction main, par exemple ligne 5 :

```
float moy ;
```

puis en calculant et affichant la moyenne (ligne 17 par exemple) :

```
moy = (somme-99) / cpt ;
printf("La moyenne des valeurs saisies est : %f \n",
(somme-99) / cpt );
```

Mais même en ce cas le résultat reste faux.

En fait la difficulté provient de la façon que le langage C a de réaliser les calculs.

En C, l'opérateur s'adapte aux opérandes.

Or somme et cpt sont des entiers, en conséquence la division calculée est la division entière : 11+11+10 = 32, et 32/3 valent 10 avec un reste de 2.

Il y a plusieurs solutions pour contourner cette "difficulté".
1/ Déclarer les variables en float au lieu d'int.
2/ S'il s'agit de nombres codés en dur, rajouter un point décimal (ex : 3./2 vaut 1.5 alors que 3/2 vaut 1)

3/ Faire du **transtypage**.

C'est cette dernière solution que nous allons utiliser.

Transtypage :

En réalité les données en mémoire de l'ordinateur sont codées sous la forme d'une suite de 0 et 1 au format binaire. Sur les systèmes récents, les machines utilisent des **mots** de 64 bits, chaque bit pouvant valoir 1 ou 0.

Ce qui fait qu'un mot est compris comme un entier, ou un réel, voire un caractère ne relève que de la volonté du programmeur qui a déclaré une variable dans un type particulier. Mais il reste possible d'interpréter **LOCALEMENT** le mot en question dans le type que l'on veut à l'aide du transtypage.

La syntaxe est la suivante :

```
( type )expression
```

Le fait de préciser devant une expression un type entre parenthèses fait que localement l'expression prend le type spécifié. Ainsi (float)32/3 retourne un float qui vaut 10,666.

Le programme doit être simplement corrigé ligne 17 comme suit :

```
printf("La moyenne des valeurs saisies est : %f\n",
(float)(somme-99) / cpt );
```

pour retourner le résultat correct.