On se propose d'écrire un programme qui calcule la somme de N entiers entre 0 et 20 La saisie des nombres et le calcul de la somme se feront dans une même boucle.

◇ Dans une première version du programme, N est demandé à l'utilisateur avant la saisie des valeurs

Solution

Tout d'abord une petite analyse algorithmique. Comment calculer une somme ? Par exemple comment calculer :

$$S = x_0 + x_1 + x_2 + \dots + x_n$$

$$S = \sum_{k=0}^n x_k$$

C'est à dire :

L'algorithme consiste à effectuer les actions les unes après les autres. Ainsi pour calculer 1+2+3, on calcule 1+2, puis on ajoute 3 pour obtenir le résultat.

De manière générale, on calcule $x_0 + x_1$, puis on ajoute x_2 , puis on ajoute x_3 , ... puis on ajoute x_n .

Il faut bien entendu conserver la valeur x_0+x_1 dans une variable pour pouvoir y ajouter x_2 . Afin de ne pas utiliser trop de mémoire, on conserve la même variable dans laquelle sont ajoutées au fur et à mesure les x_k tout au long du calcul. Ce type de variable est appelée un accumulateur que nous noterons S (comme somme). Il existe dans les microprocesseurs des variables particulières servant à ce type de calcul.

L'algorithme pour ajouter toutes les valeurs va donc consister en : S reçoit $S + x_k$, et ce pour tous les x_k à sommer. Cette structure algorithmique est celle d'une boucle, que nous pourrions écrire ainsi :

pour tous les k allant de 0 à n
 S reçoit $S + xk$

Bien entendu, il faut d'abord prendre la précaution d'initialiser l'accumulateur sous peine d'avoir un résultat faux. Comme il s'agit d'une somme, on initialise S à zéro.

On peut traduire cet algorithme ainsi en langage C.

```
#include<stdio.h>

int main(void)
{   int N,i,somme,x;

    printf("Entrez le nombre de valeurs : "); // Saisie
    du nombre de valeurs
    scanf("%d",&N);

    for ( i=0 , somme=0 ; i<N ; i++ ) // 3 zones séparées
    par ;
    {   printf("Entrez la valeur %d : ",i+1);
        scanf("%d", &x ) ;
        somme = somme + x ;
    }

    printf("La somme est : %d \n", somme );

    return 0;
}
```

Analyse du programme :

On procède d'abord à la saisie du nombre de valeurs à saisir. Noter que si l'on entre $N = 3$, la boucle **for** ira de 0 à 2 inclus, ce qui fait bien trois valeurs. Par contre afin de ne pas afficher *entrez la valeur 0*, mais *entrez la valeur 1*, c'est $i+1$ qui est transmis au

champ `%d` du `printf`.

Pour simplifier le programme, et comme *il n'est pas demandé de conserver les valeurs lues au clavier*, nous utilisons la même boucle pour faire la saisie et le calcul de la somme.

Noter la structure de la boucle `for` (syntaxe ici). Il y a trois zones dans une boucle `for`, et chaque zone est séparée des autres par un **point virgule** (d'où leur mise en gras et bleu dans le code ci-dessus), dans chaque zone, les différentes initialisations ou les différentes incrémentations sont séparées par des *virgules*.

Nous profitons de cette possibilité pour initialiser dans la boucle `for` notre accumulateur appelé ici `somme`. *Cette syntaxe n'est donnée ici que pour renforcer l'étude de la boucle for, elle n'est pas conseillée*. Je vous conseille l'initialisation hors de la boucle `for` comme ceci par exemple :

```
somme=0 ;  
for ( i=0 ; i<N ; i++ ) // 3 zones séparées par ;
```

Par contre, il est plus que fortement recommandé d'initialiser les accumulateurs juste avant leur boucle de sommation !

◇ Dans une seconde version du programme, la saisie des valeurs dure tant que l'on n'entre pas un code d'arrêt, ce code sera le nombre 99.

Cette fois-ci, on ne peut utiliser une boucle `for` pour la sommation, le nombre de valeurs à saisir et sommer n'est pas connu. Il faut donc utiliser une boucle conditionnelle : `while` ou `do ... while`, leurs syntaxes sont ici : `while`, `do...while`.

Solution avec `do...while` :

```
#include<stdio.h>
```

```

int main(void)
{   int somme,x;

    somme = 0 ;
    do
    {   printf("Entrez une valeur : ");
        scanf("%d", &x ) ;
        somme += x ; //somme = somme + x ;
    } while ( x != 99 ) ;

    printf("La somme est : %d \n", somme-99 );

    return 0;
}

```

Analyse du programme :

L'idée est d'effectuer la saisie et la somme tant que 99 n'a pas été saisi.

Notez l'utilisation de l'opérateur combiné pour la somme.

Lors de l'affichage du résultat, comme nous sommes sur que 99 a été entré (puisque l'on est sorti de la boucle `do...while`) la valeur affichée est diminuée de 99 pour ne pas fausser le calcul.

Le flux d'instructions reste dans la boucle `do...while` tant que la condition est vraie, comme le critère d'arrêt est la saisie de 99, on demande à rester dans la boucle tant que la valeur saisie est différente de 99 soit ici : `x != 99` (soit x non égal à 99).

Enfin, ici pas de choix, il faut initialiser l'accumulateur somme avant la boucle.

Cette méthode a des avantages, on peut saisir autant de valeurs

que l'on veut, pour peu qu'elle soit différente de 99...

Le même programme pourrait être fait avec une boucle `while` et non `do...while`, ce qui donnerait :

```
somme = 0 ;
x = 1 ;
while ( x != 99 )
{   printf("Entrez une valeur : ");
    scanf("%d", &x ) ;
    somme += x ; //somme = somme + x ;
}
```

L'utilisation de la boucle `while` impose des modifications au programme :

1/ comme le test a lieu en début de boucle et non en fin de boucle, il faut initialiser la valeur de x pour être sur d'entrer une fois dans la boucle.

2/ Il ne faut pas de `;` après le `while` alors qu'avec une boucle `do...while`, ce `;` est requis.

◇ Prévoyez un test pour que x, la valeur saisie, reste dans les limites imposées $0 \leq x \leq 20$, d'abord avec le programme du point 1, puis avec le programme du point 2.

Solution

```
#include<stdio.h>
```

```
int main(void)
```

```
{   int N,i,somme,x;
```

```
    somme = 0 ;
```

```
    do
```

```
    {   printf("Entrez une valeur : ");
```

```
        scanf("%d", &x ) ;
```

```

    if ( ( x == 99 ) || ( ( x >=0 ) && ( x <= 20 ) ) )
    {
        somme += x ; //somme = somme + x ;
    }

} while ( x != 99 ) ;

printf("La somme est : %d \n", somme-99 ) ;

return 0 ;
}

```

Analyse du programme :

La seule différence avec le programme précédent est le test à l'aide de l'instruction `if` (syntaxe ici). Dans une première approche, on ne se contente de ne pas prendre en compte les saisies qui ne sont pas comprises entre 0 et 20 et qui ne sont pas 99.

Le test est complexe. On considère que la saisie est bonne si elle vaut 99 ou qu'elle est dans l'intervalle allant de 0 inclus à 20 inclus soit : $(x == 99)$ OU $(x \text{ dans l'intervalle } [0 , 20])$

Dire que $(x \text{ dans l'intervalle } [0 , 20])$ correspond à dire que "x supérieur ou égal à zéro et x inférieur ou égal à 20" soit $((x >= 0)$ ET $(x <= 20))$.

Deux points sont à noter :

1/ comme il n'y a pas de priorité entre les opérateurs logiques (en fait tous sont au même niveau de priorité) il faut utiliser des parenthèses pour régler l'ordre des tests.

2/ Les opérateurs logiques sont : OU : `||` (ALT+SHIFT+L sur un clavier Mac OS), et ET : `&&`. Ces opérateurs sont doublés. Les opérateurs non doublés existent mais ne sont pas au programme de cette unité.

Un rappel sur les opérateurs logiques est ici.

Il est possible d'améliorer un peu le programme pour qu'il se comporte de façon plus efficace en informant l'utilisateur que la saisie est erronée :

```
#include<stdio.h>

int main(void)
{   int N,i,somme,x;

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

        if ( ( x>=20) || (x<=0) )
        {   if ( x != 99 )
                printf("Entrez une note entre 0 et 20
\n");
            }
        else
        {   somme += x ;
            }
        } while ( x != 99 ) ;

    printf("La somme est : %d \n", somme );

    return 0;
}
```

En décomposant le test en cascade de tests, il est possible d'afficher un message approprié. De plus comme la somme n'est effectuée que si x est dans le bon intervalle, il n'est plus besoin de corriger la somme lors de l'affichage.