
TD 8 : Les boucles en langage C.

Quelques bonnes habitudes à prendre (dj une boucle !).

Avant chaque écriture de programme il faut successivement:

- (a) Spécifier le problème à résoudre.
- (b) Écrire un algorithme.
- (c) Écrire le programme associé.
- (d) Simuler l'exécution du programme avec des valeurs assez variées.
- (e) S'il y a des erreurs revenir à (a).

Exercice 1. (Un peu d'chauffement)

- a) Écrire un programme en C qui fait la somme des 10 premiers nombres entiers positifs.
- b) Même question pour calculer la moyenne de N nombres reals entrés par l'utilisateur, N tant lui aussi fournit par l'utilisateur.
- c) Rapeller brièvement les différents types de boucle, leur mode d'utilisation et leur différence. Donnez des rponses différentes aux questions a), b) et c) avec les autres types de boucles.

```
#include <stdio.h>
main() {
    int i,nombre;
    float somme, tas, moyenne;
    /* tas: nombre rel courant */

    /* Nombre de reals ajouter: nombre */
    printf("Combien y a t-il de nombres reals considrer ?\n");
    scanf("%d",&nombre);
    printf("Vous avez donn %d nombres\n",nombre);

    /* Calcul de leur somme */
    somme=0;
    for (i=1; i<=nombre; i=i+1)
    {
        printf("Donnez le nombre %d:\n",i);
        scanf("%f",&tas);
        somme=somme+tas;
    };
    /* Calcul de leur moyenne */
    moyenne=somme/nombre;

    printf("La moyenne de ces %d nombres est: %f\n",nombre,moyenne);
}
```

Exercice 2. (Tables de multiplication)

1) Ecrire un programme C qui affiche les 9 tables de multiplication pour les entiers de 1 à 9. Chaque table comporte 9 éléments, sous la forme suivante:

```
1 2 3 4 5 6 7 8 9
2 4 6 8 ...
⋮
9 18 27 36 ...
```

2) Ecrire un programme C qui affiche m tables de multiplication pour les entiers de 1 à m avec n éléments dans chaque table. On demandera $m \leq 20$ (Utilisez un **do while**).

```
#include<stdio.h>
int main() {
    int i,j,n,m;
    printf("Le nombre des tables: \n");
    scanf("%d",&n);

    /* Donner la bonne valeur de m. */
    do{
        printf("Le nombre de valeurs inferieur ou egal a 20: \n");
        scanf("%d",&m);
    }
    while(m>20);

    /* Impressions des tables */
    printf("----- \n");
    for(i=1;i<=n;i++) {
        for(j=1;j<=m;j++) {
            printf("%d \t",i*j);
        }
        /* Le symbol \t sert a faire des espaces blancs */
        printf("\n");
    }
    printf("----- \n");
    return (1);
}
```

Exercice 3. (La banque)

La banque X nous accorde un prêt si la somme de vos intérêts dépasse 1000 euros. L'intérêt est de 3.5 % par an.

Voici un exemple pour vous guider dans vos affaires financières:

```
Somme initiale place: 2000 euros
1-ère année :          intret = (2000 x 3.5)/100 = 70
2-ème année :          intret = (2070 x 3.5)/100 = 72.45
...
On arrête quand :      intret > 1000 euros.
```

En suivant l'exemple précédent écrire un algorithme puis un programme C qui lit la somme d'argent placée initialement, puis détermine le nombre d'années nécessaires pour bénéficier d'un prêt.

Solution

```
#include<stdio.h>
int main() {
    int annee;
    float somme, interet;
    interet=0. ; annee=0;
    printf("Somme initiale: \n");
    scanf("%f",&somme);
    printf("-----\n");
    printf("Somme initiale =%f\n",somme);
    // printf("Annee\tInteret\tSomme\n");
    printf("-----\n");
    while(interet<1000) {
        annee=annee+1;
        interet=somme*(.035);
        somme=somme+interet;
        printf("Annee=%d\tinteret=%f\tsomme=%f\n",annee, interet, somme );
    }
    printf("-----\n");
    return (1);
}
```

Exercice 4. (Un drôle de fermier)

Un fermier fait l'élevage de moutons et de dindons et, au moment de payer ses impôts, il déclare curieusement: j'ai dans mon élevage 36 têtes et 100 pattes !

Trouvez un algorithme pour déterminer le nombre de moutons et de dindons, puis le traduire en C selon le schéma suivant (*M*: moutons, *D*: dindons, *p*: pattes)

Si M=36	alors D=0	donc $p= 36 \times 4 = 144$	impossible
Si M=35	alors D=1	donc $p= 35 \times 4 + 1 \times 2 = 142$	impossible
Si M=34	alors D=2	donc $p= 34 \times 4 + 2 \times 2 = 140$	impossible
.....			
jusqu' ce que $p= 100$.			

Solution

```
#include<stdio.h>
int main() {
    int M, p, solution;
    M=36;
    p=2*M+72;
    solution = 1;

    /* Solution =1 si ce Pb. a une solution et 0 sinon */
    /* M=nb de Montons; p=Nb de pattes.
    /* De plus Nb de dindons D verifie: D = 36 - M */
    /* Donc p = 4*M + 2*D= 4*M + 2*(36 - M) = 2*M+72 */

    while(p!=100 && M>1) {
        M=M-1; p=2*M+72;
```

```

}
if(M==1 && p!=100) solution=0;
if(solution==1) {
    printf("----- Solution: ----- \n");
    printf("Nb. de Moutons: %d\t Nb. de Dindons: %d\n",M,36-M);
    printf("----- \n");
}
else printf("----- Ce Pb. n'a pas de solution ! ----- \n");
return(1);
}

```

Exercice 5. (Puissances de 2)

En utilisant les puissances de 2 successives, écrire un programme *C* qui calcule le plus grand nombre positif possible, gnr par le type **int**.

```

#include <stdio.h>
main()
{
    int j,z,i,k;
    i=2;          /* i represente la puissance de 2 */
    k=0;
    z=0;
    while(i>0){
        j=i;
        i=2*i;
        k=k+1;
    }
    /* La valeur de i devient ngative (dpassement de capacit). */
    /* j sauvegarde la valeur precedente de i */
    /* k est un compteur qui sert mmemoriser la puissance de 2. */

    z=j-1;
    z=z+j;
    /* t=z+1; */

    /* Remarquez qu'on ne peut pas faire directement z=2*j-1 */
    /* a cause du depassement de capacite. */

    printf("\n\nNotons POWER, la + grande puissance de 2 en int\n");
    printf("BIG, le + grand nombre positif possible, on obtient:\n\n");
    printf("    Power2 = %d = 2 puissance %d\n",j,k);
    printf("    2 fois Power2 = %d.\n\n",i);
    printf("    BIG = %d\n",z);
    printf("    Mais BIG + 1 = %d\n",z+1);
    printf("    BIG = (2 puissance %d) - 1. \n\n",k+1);
}

```

Exercice 6. (Nombres premiers)

Écrire un programme *C* qui teste si un nombre est premier ou pas, puis un programme *C* qui teste tous les nombres entre 1 et *N*, *N* tant fix par l'utilisateur. On simulera l'algorithme pour *N* = 10.

N.B: Un nombre premier est un nombre qui n'est divisible uniquement par 1 et par lui même (1 est considr comme premier).

```

/* ***** */
/*          Question 1:          */
/* Programme C qui teste si N est premier, */
/* sinon, il donne tous ses diviseurs      */
/*          */
/* ***** */
*/

```

```

#include<stdio.h>
int main() {
    int i,N, premier;
    premier=1;
    do{
        printf("Donnez un nombre entier positif: \n");
        scanf("%d",&N);
    }
    while(N<=0);
    if(N==2) premier=1;
    if(N%2==0 && N>2) {
        premier=0;
        printf("%d est pair\n",N);
    }
    /* Recherche de diviseurs impairs >=3 */
    if(N>2) {
        for(i=3;i<N;i=i+2) {
            if(N%i==0) {
                premier=0;
                printf("\t%d divise %d\n",i,N);
            }
        }
    }
    if(premier==0) printf("Donc %d n'est pas premier\n",N);
    if(premier==1) printf("%d est premier\n",N);
    return(1);
}

```

Voici un exemple realise avec ce programme:

Donnez un nombre entier positif:

```

54879111
    3 divise 54879111
    7 divise 54879111
    9 divise 54879111
   17 divise 54879111
   21 divise 54879111
   51 divise 54879111
   63 divise 54879111
  119 divise 54879111
  153 divise 54879111
  357 divise 54879111
 1071 divise 54879111
 51241 divise 54879111
153723 divise 54879111
358687 divise 54879111
461169 divise 54879111

```

```

871097 divise 54879111
1076061 divise 54879111
2613291 divise 54879111
3228183 divise 54879111
6097679 divise 54879111
7839873 divise 54879111
18293037 divise 54879111
Donc 54879111 n'est pas premier

```

```

/* ***** */
/*          Question 2:          */
/* Programme C qui lit N >= 4, et donne */
/* la liste de tous les premiers < N      */
/*          */
/* ***** */

#include<stdio.h>
int main() {
    int i,N, nb, premier;

    /*          nb: nombre courant          */
    /*          premier=1 si nb est premier et 0 sinon */

    do{
        printf("Donnez un nombre entier positif: \n");
        scanf("%d",&N);
    }
    while(N<=0);

    printf("\nListe des nombres premiers impairs < %d:\n",N);
    /*          On suppose N >= 4          */

    for(nb=3; nb<N; nb=nb+2) {
        premier=1;
        if(nb%2==0) premier=0;
        for(i=3;i<nb;i=i+2) {
            if(nb%i==0) premier=0;
        }
        if(premier==1) printf("\t%d \n",nb);
    }
    return(1);
}

```

Exercice 7. (A ne pas faire, sauf si on veut réfléchir un peu)

On suppose qu'on utilise la boucle **for** que lorsque l'on connaît les bornes. Pensez-vous que l'on peut toujours changer une boucle **while** par une boucle **for**

Indication: essayer avec l'algorithme suivant (et le programme *C* associé) : (x entier ≥ 1)

```

Si x=1 alors stop
Sinon
    Tant que (x>1) Faire
        si x pair alors x <--- x/2
        si x impair alors x <--- 3x+1

```

Fin Tantque
FinSinon

Solution En essayant quelques exemples, $2 \leq x \leq 16$, on s'aperçoit rapidement que le nombre d'itérations est très aléatoire. Il faut se méfier des apparences, malgré l'étonnante simplicité de cet algorithme ... on ne sait pas, à ma connaissance (non actualisée il est vrai) si cet algorithme se termine ! Une quipiste Stanford a montré que cet algorithme se termine pour $x \leq 10^{200}$ mais pour les autres ? On ne connaît donc pas le nombre d'itérations de la boucle en fonction de x , donc pas la complexité. A fortiori, on ne peut pas utiliser une boucle **for** classique avec des bornes connues à l'avance.