INPT
LA GRANDE ÉCOLE DES TÉLÉCOMMUNICATIONS ET DES TECHNOLOGIES DE L'INFORMATION

Filière Smart-ICT

Algorithmique et Programmation C

Mr N.EL FADDOULI
elfaddouli@emi.ac.ma
nfaddouli@gmail.com

Année Universitaire:2024/2025

Plan

Le langage C: La boucle for (1/3)

- La boucle **for** est utilisée pour répéter un bloc d'instructions un certain nombre de fois. Elle est principalement utilisée lorsqu'on connaît à l'avance le nombre d'itérations que l'on souhaite effectuer.
- Sa syntaxe est la suivante:

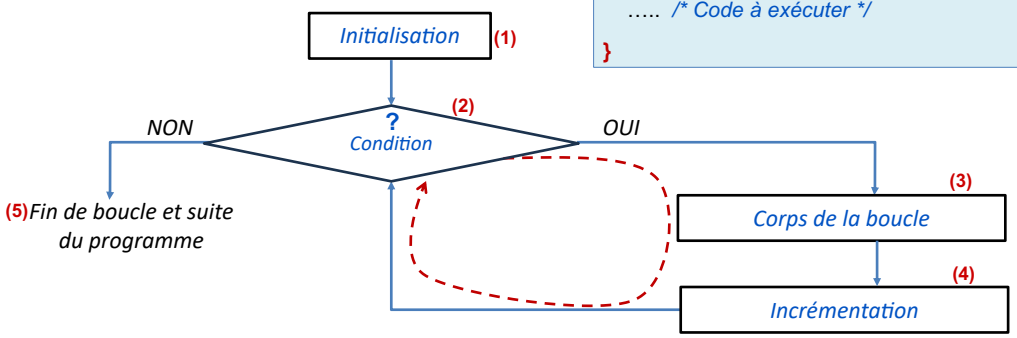
```
for ( initialisation ; condition ; incrémentation ) {
        .....   /* Code à exécuter */
}
```

- **Initialisation** : On initialise une ou plusieurs variables avant que la boucle ne commence. Cela ne s'exécute qu'**une seule fois au début**.
- **Condition** : Tant que cette condition est vraie, le corps de la boucle est exécuté. Lorsque la condition devient fausse, la boucle s'arrête.
- **Incrémentation** : C'est l'opération qui s'exécute à la **fin de chaque itération**. Elle est souvent utilisée pour modifier les variables d'initialisation.

Le langage C: La boucle for (2/3)

- La boucle **for** est exécuté selon l'organigramme suivant:

```
for (initialisation ; condition ; incrémentation) {
    ..... /* Code à exécuter */
}
```



- **Exemple**:

```
for (i=0 , j=10 ; i<=j  ; i++ ,  j-- ) {
    printf("%d ---- %d\n", i , j );
}
```

⇨

```
0 ---- 10
1 ---- 9
2 ---- 8
3 ---- 7
4 ---- 6
5 ---- 5
```

Le langage C: La boucle for (3/3)

☞ **Exemple**: Calcul de la somme S = 1+2+3+…+N

```
for (i=1 , S=0   ; i<=N ; i++) {
    S += i;
}
```

```
i=1 ;
S= 0  ;
for ( ; i<=N ; i++)   S += i;
```

```
i=1 ;
S= 0  ;
for (   ; i<=N ; ) { S += i;
                     i++ ;    }
```

☞ Les accolades sont **optionnelles** si le corps de la boucle contient **une seule instruction**.

☞ L'initialisation, la condition et l'incrémentation peuvent être **vides**.

Exemple:

```
i=1 ;  S= 0  ;
for (   ;   ; ) {   if ( i >N ) break ;
                    S += i;
                    i++ ;    }
```

Sans ce test pour faire **break**, on aura une boucle infinie

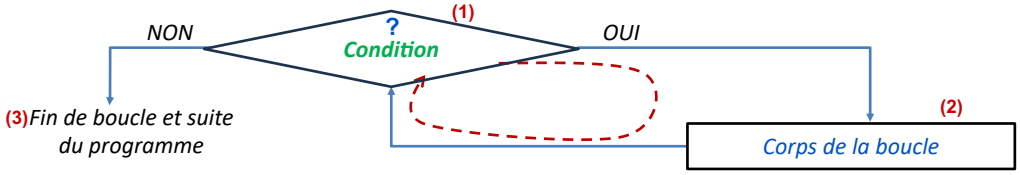
Le langage C: La boucle while (1/2)

☞ La boucle **while** est utilisée pour répéter un bloc d'instructions tant qu'une condition donnée est vraie.

☞ Sa syntaxe est la suivante:

```
while (condition ) {
    ….. /* Code à exécuter */
}
```

☞ Elle est exécutée selon l'organigramme suivant:



☞ **Exemple**: Calcule de S =1+2+3+….+N

```
S=0 ;  i=1 ;
while (i <=N ) {  S+= i ;
                  i++; }
```

Le langage C: La boucle while (2/2)

- Il faut s'assurer que la condition deviendra fausse à un moment donné, sinon on aura une boucle infinie.

Exemple:

```
i=1;                    (true)
while ( 1 ) {
    printf("Je suis l'itération: %d\n", i);
    ….. /* suite du code à exécuter */
    i++;
}
```

```
Je suis l'itération: 1
Je suis l'itération: 2
Je suis l'itération: 3
Je suis l'itération: 4
……
```

C'est une boucle **infinie**.

Si on veut qu'elle s'arrête à un moment donné, il faut ajouter dans son corps un test pour faire **break** qui force l'arrêt de la boucle:

if (*condition*) break;

Par exemple: if (i>10) break;

- ⚠ Si la condition est fausse dès le départ, le corps de la boucle ne sera pas exécuté

  ⇨ Le corps de la boucle **while** est exécuté **0** ou plusieurs fois.

- Le corps de la boucle peut inclure à son tour une autre boucle. On aura donc des **boucles imbriquées**. Par exemple pour calculer le nombre de diviseurs **de chaque entier** d'un intervalle [A,B]

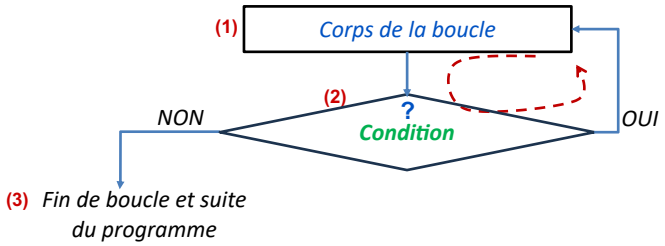
Le langage C: La boucle do-while (1/2)

- La boucle **do-while** est utilisée pour répéter un bloc d'instructions tant qu'une condition donnée est vraie en **commençant par exécuter d'abord le bloc** d'instructions **avant de vérifier si la condition** est toujours vraie..

- Sa syntaxe est la suivante:

```
do {
    ….. /* Code à exécuter */
} while (condition ) ;
```

- Elle est exécutée selon l'organigramme suivant:



- ⚠ Le corps de la boucle **do-while** est exécuté **au moins une fois** même si la condition est fausse dès le départ.

Le langage C: La boucle do-while (2/2)

☞ **Exemple:** Exiger la saisie d'un entier **N**∈[2, 10] pour pouvoir poursuivre l'exécution.

```
do {
        printf("Donnez un entier :");
        scanf("%d",&N) ;
   } while (N<2 | | N>10) ;
```

☞ **Exemple:** Afficher les chiffres d'un entier N en commençant du dernier jusqu'au premier

```
printf("Donnez un entier :");
scanf("%d",&N) ;
do {
        printf("%d \n",   N%10);
        N=N/10;
   } while (N>0) ;
```

Pour avoir le dernier chiffre de N

Pour éliminer le dernier chiffre de N

Le langage C: L'instruction break dans une boucle

☞ L'instruction **break** permet de **forcer l'arrêt** d'une boucle quelle que soit sa condition d'exécution. Elle interrompt immédiatement la boucle et le programme continue à partir du code suivant la boucle.

☞ Elle est généralement exécutée si une **condition** donnée est vraie

☞ **Exemple:** Lecture de N entiers et arrêt au 1ier entier saisi qui est multiple de 3

```
int i, N, k;
printf("Donnez le nombre d'entiers");  scanf("%d", &N);
for(i=1; i<=N; i++) { printf("donnez un entier: ");
                      scanf("%d", &k);
                      if(k%3==0)  break;  /* Arrêt de la boucle même si  i≤ N */
                    }
if (i>N)   printf("Aucun multiple trouvé \n");
else       printf("Le premier trouvé est = %d", k);
```

Sans ce test et break, le corps de la boucle sera exécuté N fois

Le langage C: L'instruction continue dans une boucle

- L'instruction **continue** permet de **sauter directement à la prochaine itération** d'une boucle sans exécuter le reste du code dans le corps de la boucle pour cette itération spécifique.
- Elle est utile lorsqu'on veut ignorer certaines parties de la boucle sous certaines conditions.
- Elle est généralement exécutée si une **condition** donnée est vraie
- **Exemple:** Lecture de N entier et calcul de la somme des impairs.

```
int i, N, k, S=0;
printf("Donnez le nombre d'entiers");  scanf("%d", &N);
for(i=1; i<=N; i++) { printf("donnez un entier: ");
                      scanf("%d", &k);
                      if(k%2==0) continue;  /* passer à l'itération suivante */
                      S+=k;
                     }
printf("La somme des impairs est: %d \n", S)
```

Sans ce test et **continue**, tous les entiers saisis (k) seront ajoutés à S

