

CYCLE 4 : BOUCLES TANT QUE

Répéter dans un programme avec une Instruction itérative

- 1- Comment écrire une instruction TANT QUE
- 2- Comment le CPU exécute une boucle TANT QUE
- 3- EXEMPLES :
 - Table de multiplication
 - Programme avec reprise
 - Saisie validée



CYCLE 4 : BOUCLES TANT QUE

Répéter dans un programme avec une Instruction itérative

- 1- Comment écrire une instruction TANT QUE

```
do { } while ();  
while () { }
```



Instruction FAIRE - TANT QUE



```
int main()
{
    /* Répétition sur condition : FAIRE - TANT QUE
        do
        {
            instructions;
        } while (condition);
    */


    int    a=2;
    do
    {
        printf("passage boucle");    a= a-1;
    } while (a>0);
}
```

Instruction TANT QUE




```
int main()
{
    /* Répétition sur condition : TANT QUE
        while (condition)
        {
            instructions;
        }
    */


    int    a=2;
    while (a>=0)
    {
        printf("passage boucle");    a= a-1;
    }
}
```


 **À NOTER !!**

```
do
{
    instructions;
} while (condition);
```


 **UN ; à la fin du *while()***
seulement pour le *do*
***while* !!!!**

```
while (condition)
{
    instructions;
}
```






Règle de programmation : *lisibilité code*
Indentations dans les accolades du *while*


 5

Rôle des instructions TANT QUE



Le TANT QUE répète l'exécution d'un bloc d'instructions tant qu'une condition est vraie

(boucle conditionnelle)

 6

CYCLE 4 : BOUCLES TANT QUE

Répéter dans un programme avec une Instruction itérative

2- Comment le CPU exécute une boucle TANT QUE

Principe d'exécution de l'instruction FAIRE – TANT QUE



```
do  
{ instructions;  
} while (condition);
```



*Le CPU recommence l'exécution du bloc d'instructions **tant que la condition de poursuite de boucle est VRAIE**. Dès que la condition de fin de boucle devient VRAIE, la boucle s'arrête :*

- Le CPU passe le DO et exécute les instructions de boucle
- Le CPU évalue la condition de poursuite de boucle : condition
- Si condition VRAIE, le CPU réexécute la boucle
- Lorsque le CPU revient au WHILE, le CPU évalue condition
- Quand la condition devient FAUSSE, le CPU sort du TANTQUE.

Principe d'exécution de l'instruction TANT QUE



```
while (condition)
{
    instructions;
}
```



*Le CPU recommence l'exécution du bloc d'instructions **tant que la condition de poursuite de boucle est VRAIE**. Dès que la condition de fin de boucle devient VRAIE, la boucle s'arrête :*


- Le CPU évalue la condition de poursuite de boucle : condition
- Si condition VRAIE, le CPU exécute la boucle, puis revient au WHILE
- De retour au WHILE, le CPU évalue condition
- Quand la condition devient FAUSSE, le CPU sort du TANTQUE.



Précision sur le premier passage dans une TANT QUE

```
do
{
    instructions;
} while (condition);
```

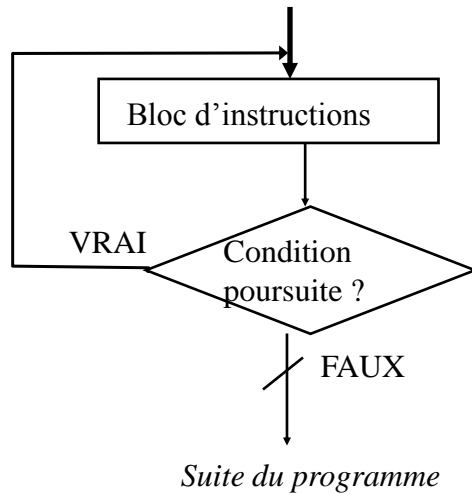
```
while (condition)
{
    instructions;
}
```

 Le do {} while s'exécute **toujours au moins une fois**



Organigramme de l'instruction FAIRE - TANT QUE

```
do
{   instructions;
} while (condition);
```



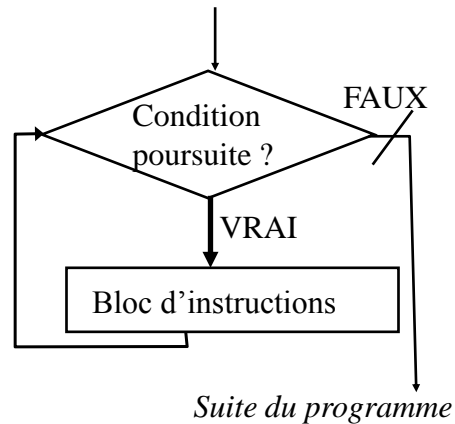
Simulation d'exécution du FAIRE – TANT QUE

| INSTRUCTION | PROCESSEUR |
|-----------------|---|
| 1- do | exécution des instructions de la boucle |
| 2- while | évaluation condition logique de poursuite : VRAI |
| 3- do | exécution des instructions de la boucle |
| 4- while | évaluation condition logique de poursuite : VRAI |
| 5- do | exécution des instructions de la boucle |
| 6- while | évaluation condition logique de poursuite : FAUX |
| 7- | suite du programme après la fin du TANTQUE |



Organigramme de l'instruction TANT QUE

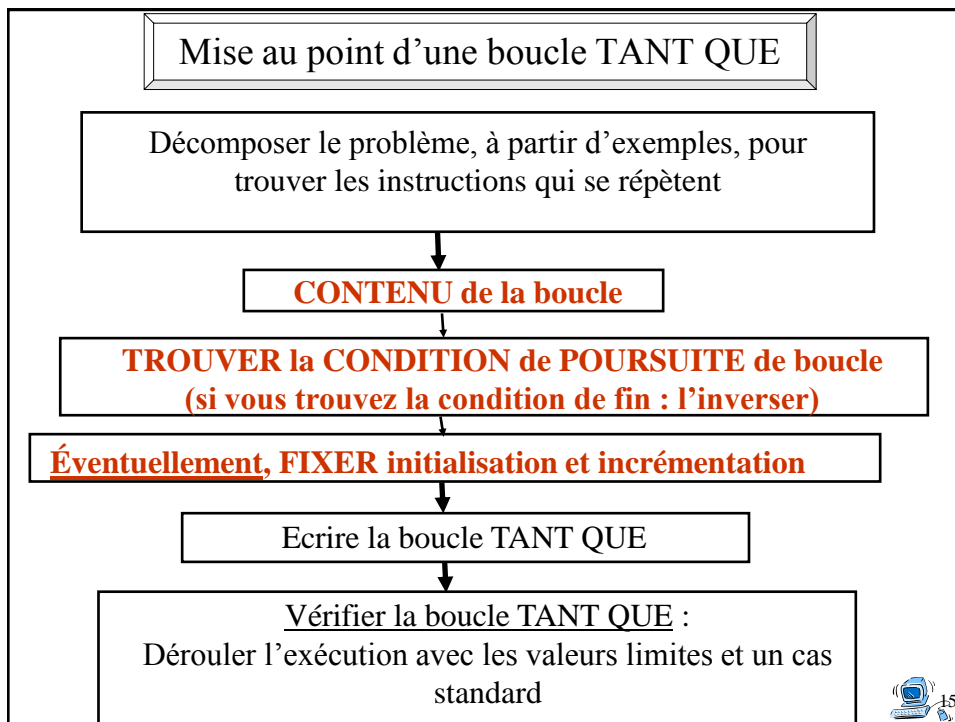
```
while (condition)
{
    instructions;
}
```



Simulation d'exécution du TANT QUE

| INSTRUCTION | PROCESSEUR |
|---------------------|---|
| 1- while | évaluation condition logique de poursuite : VRAI |
| 2- { | exécution des instructions de la boucle |
| 3- } : while | évaluation condition logique de poursuite : VRAI |
| 4- { | exécution des instructions de la boucle |
| 5- } : while | évaluation condition logique de poursuite : FAUX |
| 6- | suite du programme après la fin du TANTQUE |






CYCLE 4 : BOUCLES TANT QUE

Répéter dans un programme avec une Instruction itérative

3- EXEMPLES :

- Table de multiplication
- Programme avec reprise
- Saisie validée



Affichage de la table de multiplication par 2

```

int main()
{
    int produit, i;

    /* comptage et affichage des produits */

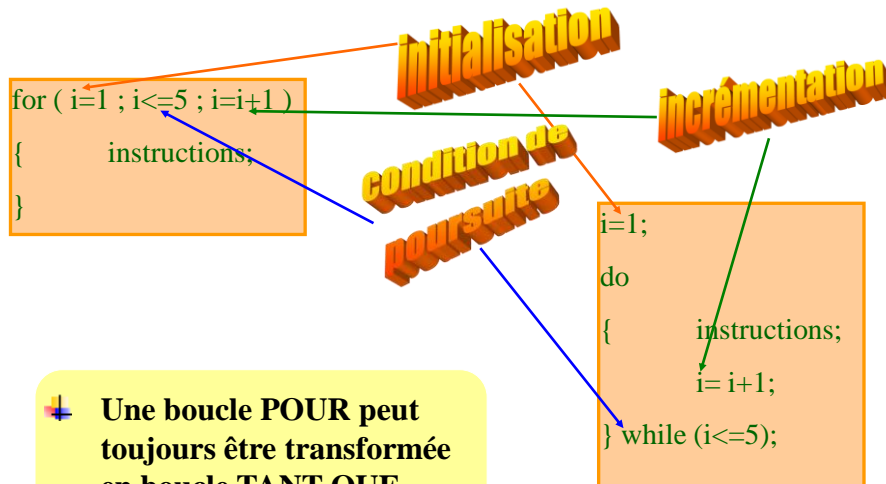
    i = 0;
    do
    {
        produit = i * 2;
        printf("%d x 2 = %d\n", i, produit);
        i++;
    } while (i < 11);
}

```

initialisation: i=0 → `i = 0;`
exécution boucle → `{`
incrément: i=i+1 → `i++;`
rebouclage → `} while (i < 11);`
fin → `}`

| Instruction | Processeur | Mémoire |
|---|--|---------|
| 1- Déclaration variables | Réservation mémoire | |
| 2- Initialisation : i = 0 | affectation | |
| 3- do -produit= i*2; -printf(); -i= i+1; | a- calcul: 0x2 b- affectation c- affichage d- calcul: 0+1 e- affectation | |
| 4- while | évaluation condition poursuite 1<11 : VRAI | |
| 5- do -produit= i*2; -printf(); -i= i+1; | a- calcul: 1x2 b- affectation c- affichage d- calcul: 1+1 e- affectation | |
| ... | ... | |
| 6- while | éval condition poursuite 11<11 : FAUX | |
| 7- | suite après TANTQUE | |

Equivalence entre les boucles POUR et TANT QUE



Différence entre les boucles POUR et TANT QUE

- La boucle **TANT QUE** permet de faire plus de choses : la reprise dépend d'une condition logique quelconque.
- On pourrait tout écrire avec des **TANT QUE**.
- Quand on sait combien de fois la boucle doit tourner, la **POUR** est plus simple.



GESTION D'UN MENU

Le programme réalise une calculatrice avec reprise

```
int main()
{
    float    a, b, resu ;      int    choixMenu;

    do          // reprise du programme tant que l'utilisateur ne veut pas sortir
    {
        printf("1) addition,\n2) soustraction\n3) SORTIR\n\tChoix :");
        scanf(" %d",&choixMenu);
        printf("Donner 2 réels:"); scanf("%f%f",&a,&b);

        if (choixMenu==1)      // addition
        {
            resu= a+b;
        }

        if (choixMenu==2)      // soustraction
        {
            resu= a-b;
        }

        printf("Resultat de l'opération : %.2f",resu);

    } while (choixMenu!=3);
}
```

Précisions sur la gestion d'un menu

La boucle de reprise du programme permet d'éviter d'avoir à relancer le programme (RUN) pour une nouvelle exécution.



22

Démo Reprise de Programme

CalculatriceReprise.exe

Le programme de calculatrice recommence si l'utilisateur veut faire un nouveau calcul.



SAISIE VALIDEE

Le programme saisit un entier $\in [5,150]$

```
int main()
{
    int    x;

    do      // reprise de saisie si saisie invalide
    {
        printf("Donner une valeur comprise entre 5 et 150 : ");
        scanf("%d",&x);

        if (x<5 || x>150 ) // message d'erreur si saisie invalide
        {
            printf("Erreur de saisie");
        }

    } while (x<5 || x>150 );
}
```



Précisions sur la SAISIE VALIDEE

- ☐ le programmeur doit vérifier la validité de toutes les saisies du programme et faire ressaisir en cas d'erreur
- ☐ le programmeur doit bien informer l'utilisateur de ce qui doit être tapé au clavier et des erreurs qu'il commet



Démo Saisie Validée

SaisieValide.exe

Le programme saisit un réel dans $[-50.5, 100.25]$ exclusivement.

