**Chapitre 3 : Les Boucles**

1. **La boucle Tant que**

Syntaxe en Algo :

**Tant que** condition **faire**

instruction

**Fin Tant que**

Syntaxe en C et PHP :

**while** (condition) **{**

instruction ;

**}**

La boucle tant que exécute une ou plusieurs instructions autant de fois qu’une condition reste vérifiée (vrai ou faux). Si la condition est au départ non vérifiée, il n’y aura aucune exécution de l’instruction.

Dans le corps de la boucle, il faut prévoir une instruction d’évolution de la condition. Si la condition n’évolue pas, la boucle devient une boucle infinie.

Représentation en organigramme :



**Exemple** : Écrire un algorithme qui affiche les nombres pairs compris entre 2 et 20.

| Algo : Pairs  Déclaration  nb : entier  Début  nb <- 2  Tant que nb <= 20 faire  Afficher (“Nombre pair :”, nb)  nb <- nb + 2  Fin tant que  Fin Pairs |
| --- |

**Traduction en C :**

| #include <stdio.h>  int main (){  int nb ;  nb = 0 ;  while ( nb <= 20 ) {  printf(“Nombre pair : %d”, nb);  nb = nb + 2 ;  }  return 0 ;  } |
| --- |

**Le schéma général de la boucle :**



1. **La boucle Faire - Tant que**

**Syntaxe en Algo :**

**Faire**

instruction

**Tant que** condition

**Syntaxe en C et PHP :**

**do** {

instruction ;

} **while** (condition) ;

Cette boucle est un schéma inverse de la boucle Tant que : Le test de la condition est postérieur à l’exécution de l’instruction. Donc, l’instruction est exécutée au moins une fois sinon plusieurs fois.

De même que la boucle Tant que, on doit prévoir une instruction d’évolution de la condition dans le corps de la boucle.

**Représentation en organigramme :**



**Exemple** : Écrire un algorithme qui affiche les nombres pairs compris entre 2 et 20.

| Algo : Pairs  Déclaration  nb : entier  Début  nb <- 2  **Faire**  Afficher (“Nombre pair :”, nb)  nb <- nb + 2  **Tant que** nb <= 20  Fin Pairs |
| --- |

**Traduction en C :**

| #include <stdio.h>  int main (){  int nb ;  nb = 0 ;  do {  printf(“Nombre pair : %d”, nb);  nb = nb + 2 ;  } while ( nb <= 20 );  return 0 ;  } |
| --- |

**Le schéma général de la boucle :**



1. **La boucle Pour**

**Syntaxe en Algo :**

**Pour** indice **allant de** VD **à** VF **pas de** Pas **faire**

instruction

**Fin Pour**

* VD : valeur début
* VF : valeur de fin

**Syntaxe en C et PHP :**

for (initialisation ; condition ; évolution ) {

instruction ;

}

Cette boucle est déterministe, elle exécute une instruction un nombre de fois connu à l’avance, calculé à l’aide de la valeur début, la valeur de fin et le pas.

Le pas peut être une valeur positive ou négative. Il indique l’évolution de la boucle pour atteindre la valeur de fin. Le pas par défaut est fixé à 1 ou (-1).

**Représentation en organigramme :**



**Exemple** : Écrire un algorithme qui affiche les nombres pairs compris entre 2 et 20.

| Algo : Pairs  Déclaration  nb : entier  Début  **Pour** nb **allant de** 2 à 20 **pas de** 2 **faire**  Afficher (“Nombre pair :”, nb)  **Fin Pou**r  Fin Pairs |
| --- |

**Traduction en C :**

| #include <stdio.h>  int main (){  int nb ;  for (nb = 2; nb <= 20 ; nb = nb + 2 ) {  printf(“Nombre pair : %d”, nb);  }  return 0 ;  } |
| --- |

**Série d’exercices :**

**Exercice 1** : Ecrire un algo, un Prog C et un Dev PHP qui saisit un nombre entier et une limite et affiche la table de multiplication de ce nombre de 1 jusqu’à la limite.

Solution en Algo :

| Algo : multiplication  Déclaration  nb, limite : entier  mult : entier  res : entier  Début  Afficher ("Donner le nombre entier :")  Saisir (nb)  Afficher ("Donner la limite de la table :")  Saisir (limite)  mult <- 1  Tant que mult <= limite faire  res <- nb \* mult  Afficher (mult, " \* ", nb , " = ", res )  mult <- mult + 1  Fin tant que  Fin Multiplication |
| --- |

Traduction en C :

| #include <stdio.h>  int main (){  int nb, limite, mult, res ;  printf("Donner un nombre entier :");  scanf("%d", &nb);  printf("Donner une limite:");  scanf("%d", &limite);  mult = 1 ;  while (mult <= limite){  res = nb \* mult;  printf("%d \* %d = %d \n ", mult, nb, res);  mult = mult + 1;  }  return 0;  } |
| --- |

**Exercice 2 :** Écrire un Algo, un Prog C et dev PHP qui permet de saisir un entier et d’afficher la liste de ses diviseurs.

Exemple : le nombre 10 a pour diviseurs : 1, 2, 5 et 10.

**Solution avec la boucle tant que :**

| Algo : diviseurs  Déclaration  nb, div : entier  Début  Afficher ("Donner un entier : ")  Saisir (nb)  Afficher ("La liste des diviseurs :")  div <- 1  Tant que div <= nb faire  Si nb % div = 0  Alors Afficher (div, " est un diviseur")  Finsi  div <- div + 1  Fin tant que  Fin Diviseurs |
| --- |

**Solution avec la boucle Pour :**

| Algo : diviseurs  Déclaration  nb, div : entier  Début  Afficher ("Donner un entier : ")  Saisir (nb)  Afficher ("La liste des diviseurs :")  Pour div Allant de 1 à nb faire  Si nb % div = 0  Alors Afficher (div, " est un diviseur")  Finsi  Fin pour  Fin Diviseurs |
| --- |

Traduction en C :

| #include <stdio.h>  int main(){  int nb, div ;  printf("Donner un entier :");  scanf("%d", &nb);  for (div = 1; div <= nb ; div = div +1){  if (nb % div == 0){  printf("\n Diviseur : %d", div);  }  }  return 0;  } |
| --- |

**Exercice 3 :** On surveille la température d’un malade pendant 24 heures, toutes les deux heures. Écrire un ALgo, un Prog C et un Dev PHP qui permet de saisir les températures et d’afficher la température moyenne, la température max et la température min.

**Solution en Algo :**

| Algo : Temperature  Déclaration  temp, max, min, moyen, somme : réel  heure : entier  Début  somme <- 0  max <- 0  min <- 50  Pour heure Allant de 1 à 12 faire  Afficher ("Donner la température pour l'heure :", heure\*2)  Saisir (temp)  somme <- temp + somme  Si temp > max  Alors max <- temp  Fin si  Si temp < min  Alors min <- temp  Fin si  Fin pour  moyen <- somme / 12  Afficher ("la temperature moyenne est de : ", moyen )  Afficher ("La temperature max est de : ", max)  Afficher ("La temperature min est de : ", min)  Fin Temperature |
| --- |

Solution en C