– Algorithmique – Travaux Pratiques – Série N°3 – LES BOUCLES –

I – Nouveaux Mots Clés : *do, for, sizeof, while.*

II – Les boucles en algorithmiques et en langage C

Les boucles permettent d'exprimer toute itération (répétition) dans la résolution d'un problème.

Les trois boucles ***Pour***, ***Répéter***, ***Tantque (TQ)*** sont traduites en langage C comme suit :

La boucle POUR :

Pour i ← Valinit **à** ValFin **Pas** = p **Faire** sequence **FPour**	`for ( i = Valinit ; i <= ValFin ; i += p )` `{ sequence }`

Dans la boucle **for :**

✓ Si le **pas** est négatif, le test de fin devient **`i >= ValFin`** et l'instruction de décrémentation est écrite **`i -= p`** (`p est positif`). Exemple : **`for(`** `i= 1 ; i >= -10 ; i -= 2` **`)`**

✓ Si le **pas** est égal à **1**, l'instruction d'**incrémentation** est, en général, écrite **`i++`**.

✓ Si le **pas** est négatif et égal à **-1**, l'instruction de **décrémentation** est, en général, écrite **`i--`**.

La boucle REPETER :

Répéter sequence **Jusqu'à (** condition **)**	**`do`** `{ sequence }` **`while (`** `condition` **`) ;`**

Dans la boucle **do … while :**

✓ La séquence d'instructions est exécutée au moins une fois.

✓ La condition est une expression booléenne.

✓ La répétition s'arrête quand la condition devient fausse (= ***Faux***). C'est-à-dire à l'inverse de la boucle **Répéter**. Donc, pour traduire la boucle **Répéter** en langage C, il faut inverser (complémenter) la condition.

Exemple : Pour lire un entier ***n*** strictement positif :

Répéter Lire(n) **Jusqu'à n > 0**	**`do`** `scanf("%d", &n) ;` **`while( n <= 0 );`**	**`do`** D'une manière générale `{ sequence }` **`while( !(`**`conditionRepeter`**`));`**

La boucle TANTQUE (TQ) :

TQ (condition **) Faire** sequence **FTQ**	**`while (`** `condition` **`)`** `{ sequence }`

Dans la boucle **while :**

✓ La condition est une expression booléenne.

✓ La répétition s'arrête quand la condition devient fausse (= ***Faux***). Si la condition est fausse dès le départ, la séquence d'instructions n'est jamais exécutée.

Remarques Générales :

✓ Dans les trois boucles ***(for, do while, while)***, les parenthèses sont obligatoires **( )**.

✓ Dans les trois boucles, si une séquence est formée d'une seule instruction, les accolades **{ }** deviennent facultatives (non obligatoires).

✓ Si la condition est de la forme ***exp1 && exp2***, l'expression ***exp2*** n'est pas évaluée si l'expression ***exp1*** est égale à ***Faux*** car, au final, l'expression globale donnera ***Faux***.

✓ Si la condition est de la forme ***exp1 || exp2***, l'expression ***exp2*** n'est pas évaluée si l'expression ***exp1*** est égale à ***Vrai*** car, au final, l'expression globale donnera ***Vrai***.

III – Correction De Quelques Erreurs

1- Dans la boucle **for**, il est très fréquent d'utiliser des virgules (,) à la place des points-virgules, ce qui se traduit à la compilation par l'erreur : **" error: expected ';' before ')'"** ***TRADUCTION*** **" ;** manquant avant **')"**. Cette erreur est rencontrée car le compilateur ne trouve pas la syntaxe for(… **;** … **;** …).

2- **Erreur de logique due à une erreur de syntaxe** : Si à la compilation il n'y a aucune erreur, mais à l'exécution, la boucle **for** ne s'exécute qu'une seule fois, il est très probable qu'un point-virgule est ajouté, par erreur, à la suite de la définition des paramètres de la boucle **for**.

Exemple : Pour afficher les nombres entiers compris entre 1 et 10

```
for (i = 1 ; i <= 10 ; i++) ;
{ printf("%d\n", i); }
```

Cette version erronée affiche **11 seulement**

Erreur !→ Correction

```
for (i = 1 ; i <= 10 ; i++)
{ printf("%d\n", i); }
```

Cette version corrigée affiche 1 2 3 4 5 6 7 8 9 10

Explication : Quand le compilateur rencontre ce point-virgule ajouté par erreur, il considère le corps de la boucle **for** vide, c'est-à-dire qu'il ne contient aucune instruction. Ce qui explique que la compilation est OK. Mais à l'exécution : il y a une initialisation (i=1) et des incrémentation (10 fois i++). La boucle s'arrête quand i devient supérieur à 10, c'est-à-dire égal à 11, ce qui explique le résultat d'affichage. **Attention !!!**

3- **Erreur de logique due à une erreur de syntaxe** : Si à l'exécution, la boucle **while** ne s'arrête pas (***boucle infinie***), il est très probable qu'un point-virgule est ajouté, par erreur, après la condition.

Exemple : Pour afficher les nombres entiers compris entre 1 et 10 et après l'initailisation `i = 1 ;`

```
while( i <= 10 ) ;
{ printf("%d\n", i);  i++; }
```

Cette version **erronée** n'affiche **RIEN**

Erreur !→ Correction

```
while( i <= 10 )
{ printf("%d\n", i); i++ ;}
```

Cette version **corrigée** affiche 1 2 3 4 5 6 7 8 9 10

Explication : A la compilation, le corps de la boucle **while** est considéré comme vide. A l'exécution, comme la condition ne change pas, la boucle **while** ne s'arrête pas et elle n'affiche rien. **Attention !!!**

IV – LES EXERCICES – LES BOUCLES

PARTIE 1 : Soit N un entier positif (de type **int**). Écrire un programme C pour les problèmes suivants :

Q1- Afficher les nombres **impairs** inférieurs ou égal à N.

***Q2-** (*Optionnel*) Afficher les nombres **pairs** inférieurs ou égal à N.

Q3- Calculer la **somme** S = 1 + 2 +3 + … + N.

***Q4-** (*Optionnel*) Calculer de la **somme** $S = 2^2 + 4^2 + 6^2 + \ldots$ en prenant N termes.

Q5- Calculer la **puissance** d'un nombre **réel** X c'est-à-dire $X^N = X * X * \ldots * X$, N fois

Q6- Calculer la **factorielle** de N. C'est-à-dire N ! = N * (N-1) * … * 3 * 2 * 1 (0 ! = 1 et 1 ! = 1).

- Afficher les valeurs de 0!, 1!, 2!, 3!, … 15!, 16! , 17! Que remarquez-vous ?
- Remplacer le type ***int*** par ***long*** (puis ***long long***). Afficher 15!, 16! , 17!

De nouveau, que remarquez-vous ?

> **Remarque** : Dans l'instruction printf, pour afficher une variable de type ***long*** utiliser le format ***%ld*** et pour le type ***long long*** utiliser ***%lld.***

- Utiliser l'opérateur (ou fonction) ***sizeof***(x) qui renvoie le nombre d'octets occupés par la variable ou le type x. **Exemple** : ***sizeof(int)*** donne le nombre d'octets occupés par chaque variable de type ***int***.

Quelle est l'explication des problèmes rencontrés ?

***Q7-** (*Optionnel*) Calculer la somme S = 1! + 2! + 3! + … + N!

Q8- Calculer la valeur approchée $e^x \cong 1+\frac{x}{1}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!}$ (***x*** : un nombre réel, ***n*** : un entier positif).

Q9- Comparer le résultat obtenu par votre programme avec la valeur donnée par la fonction ***exp(x)***. Il faut inclure la bibliothèque **math** en ajoutant la ligne `#include <math.h>` au début du programme.

- Pour un **x** donné, chercher la valeur de **n** qui permet d'avoir une précision de $\mathbf{10^{-5}}$. C'est-à-dire la différence entre la valeur approchée calculée et la valeur donnée par exp(x) soit $\leq \mathbf{10^{-5}}$.

Q10- Calculer le n^ieme terme de la suite de ***Fibonacci*** définie par : $\begin{cases} U_0 = 0 & Si\ n = 0 \\ U_1 = 1 & Si\ n = 1 \\ U_n = U_{n-1} + U_{n-2} & Si\ n \geq 2 \end{cases}$

PARTIE 2 : Nombres Parfaits

Q1- Ecrire un programme C qui permet d'afficher tous les diviseurs d'un entier N.

Un nombre est dit parfait s'il est égal à la somme de tous ses diviseurs excepté lui-même.

Exemples : **6** est parfait car **6 = 1 + 2 + 3**. Les diviseurs de 6 sont : 1, 2, 3 et 6 (exclu).

28 est parfait car **28 = 1 + 2 + 4 + 7 + 14**. Les diviseurs de 28 sont : 1, 2, 4, 7, 14 et 28 (exclu).

Q2- Ecrire un programme C qui permet de vérifier si un entier N est parfait ou pas.

Q3- Généraliser le programme précédent pour afficher tous les nombres parfaits **≤ NMax**.

PARTIE 3 : Nombres premiers

Un nombre est premier s'il n'admet que deux diviseurs : 1 et lui-même.

Q1- Ecrire un programme C qui permet de vérifier si un entier **N** est premier.

***Q2-** (*Optionnel*) Modifier le programme précédent pour afficher les nombres premiers ≤ **NMax=100**.

PARTIE 4 : Racine carrée : Pour calculer une valeur approchée de la racine carrée de n'importe quel réel positif ***a***, on construit la suite (x_n) définie comme suit : $\begin{cases} x_0 = 1 & n = 0 \\ x_n = \dfrac{(x_{n-1})^2 + a}{2x_{n-1}} & n = 1, 2, \ldots \end{cases}$

On arrête le calcul des termes de la suite à l'étape ***n*** (*n* est inconnu) tel que $| a - (x_n)^2 | \leq e$, où ***e*** est la tolérance de l'erreur (on prendra $e = 10^{-6}$). Cette valeur x_n est la valeur approchée de la racine carrée de ***a***.

Ecrire un programme C qui calcule la racine carrée d'un réel positif ***a*** par cette méthode.

Pour la valeur absolue, vous pouvez utiliser la fonction **fabs** de la bibliothèque **math**.

PARTIE 5 : Nombres Symétriques

Soit **N** un nombre entier positif.

Q1- Ecrire un programme C qui permet d'afficher les chiffres qui composent le nombre **N** ainsi que sa longueur. **Exemples** : - Si **N = 17➔** on affiche les chiffres **7** puis **1** et **La longueur = 2**.

- Si **N = 695➔** on affiche les chiffres **5** puis **9** puis **6** et **La longueur = 3**.

Q2- Ecrire un programme C qui permet de calculer puis afficher **le nombre inverse de N**.

Exemple : Si **N = 695➔** son nombre inverse **= 596**.

Un nombre N est dit **symétrique** s'il est égal à son inverse.

Exemples : Les nombres suivants sont symétriques : **1, 2, 3, 44, 55, 161, 717, 8228, 94549.**

Q3- Modifier le programme précédent pour afficher un message indiquant si N est symétrique ou pas.

Q4- Modifier le programme précédent afin qu'il chercher et affiche tous les nombres symétriques d'une longueur donnée.

OPTIONNELLE - PARTIE 6 : Chiffre de chance : Pour trouver le **chiffre de chance** d'une personne, on calcule la somme des chiffres de sa date de naissance. Au nombre obtenu, on refait le même traitement jusqu'à ce qu'on trouve un nombre composé d'un seul chiffre. Ce nombre est le chiffre de chance de la personne.

Exemple : Si la date de naissance est **29/09/1999** (Lue au clavier**29091999)**.

- On calcule la somme des chiffres de la date : **2+9+0+9+1+9+9+9 = 48**
- Le nombre **48** est composé de deux chiffres, on refait le même traitement : **4 + 8 = 12**.
- Le nombre **12** est composé de deux chiffres, on refait le même traitement : **1 + 2 = 3**.
- Le nombre **3** est composé d'un seul chiffre, c'est le chiffre de chance recherché.

Q1-Ecrire un programme C qui calcule la somme des chiffres d'un nombre entier **N**.

Q2-Modifier le programme précédent pour calculer le chiffre de chance à partir d'une date.