

## 0.1 Structure de contrôle

Nous savons que toute instruction s'achève par un ;. Par conséquent, nous risquons de ne pouvoir exécuter qu'une instruction lors de l'introduction d'une condition. Pour palier à cela il est nécessaire de construire des blocs d'instruction à l'aide de { et }. Dans la suite j'utiliserai **<instructions>** pour signifier :

```
    une_instruction ;  
ou  
    {  
    instruction_1 ;  
    instruction_2 ;  
    .....  
    instruction_N ;  
    }
```

**Note :** je vais parfois utiliser des chevrons (>) pour indiquer les parties importants du code; il s'agit de simples éléments visuels et qui ne font pas partie du code source. En d'autres termes, il ne doivent jamais être saisis.

### 0.1.1 1. Les tests :

#### L'instruction if

Sa syntaxe est :

```
    if(condition)  
    <Instructions>  
    else  
    <Instructions> /*else n'est pas obligatoire*/
```

Exemple dans la fonction calcul :

```
>if( *Max < tempmax ){  
*Max = tempmax ;  
*N=1 ;  
}  
>else if( *Max == tempmax) {++*N ;}  
>else if( *Max > tempmax) {/*Rien à faire*/}  
>if( *ecart < (tempmax - tempmin) ) *ecart= tempmax -  
    tempmin ;
```

Il est possible d'utiliser une autre syntaxe pour if, plus rapide à éditer si jamais il n'y a pas de bloc d'instructions :

```
(condition) ? <instructions> :<instructions>
```

L'instruction de gauche est réalisée si le test est renvoie la valeur vraie, l'instruction de droite si le test renvoie la valeur faux.

### L'instruction switch

L'instruction switch permet par rapport à la valeur d'une variable de réaliser une ou un bloc d'instructions. La syntaxe de switch est donnée par :

```
>switchvariable
{>casevaleur_1 :
<instructions>
>break ;
>casevaleur_2 :
<instructions>
>break ;
[.....]
>casevaleur_N :
<instructions>
>break ;
default ;
<instructions>
>break ;
}
```

La valeur default est utilisée si aucune valeur indiquée à la suite de case n'était égale à la variable.

### 0.1.2 2. Les boucles

#### La boucle for

Voici la syntaxe de cette éternelle boucle « pour » :

```
>for (i=0 ; i<=n_max ; i++)
<instructions>
```

Ici, *i* prendra initialement la valeur 0 et ira jusqu'à la valeur *n\_max* comprise par pas de 1. L'incréméntation de *i* se fait à l'aide de «++» : qu'il soit situé derrière *i* (*i++*) ou devant (*++i*), *i* sera incrémenté après que les instructions aient été effectuées (car la boucle for exécute toujours cette instruction en dernier).

#### La boucle while

Elle correspond à la boucle « tant que ». Sa syntaxe est assez aisée :

```
>while (condition)
<instructions>
```

#### La boucle do while

Elle correspond à « tant que » et sa syntaxe :

```
>do
<instructions>
>while (condition)
```

### Saut et arrêt dans les boucles

Il arrive parfois qu'un particulier se produise dans une boucle conduisant à une erreur, voir à un plantage du programme. Il est toute fois possible d'y remédier dans la mesure où nous aurions connaissance de ce cas (par exemple une division par zéro). Il suffit pour cela de réaliser un test qui permette de traiter le cas particulier à part, puis soit de reprendre la boucle, soit d'en sortir. Les fonctions respectives pour cela sont **continue** et **break**. Voici un exemple d'utilisation pour continue :

```
>for (i=0 ; i<=N_max ; ++i)
{
>if (i==3) {
printf("division par zéro") ;
continue ;
}
res=1/(i-3) ;
}
```

et pour break :

```
>for (i=1 ; i<=N_max+1 ; i++)
{
>if (i==3) {
printf("division par zéro") ;
break ;
}
res=1/(i-3) ;
}
```

«« Précédent<sup>1</sup> Suivant »»<sup>2</sup>

---

<sup>1</sup><http://www.trustonme.net/didactels/151.html>

<sup>2</sup><http://www.trustonme.net/didactels/153.html>