UNIVERSITÉ ALIOUNE DIOP DE BAMBEY
UFR SCIENCES APPLIQUÉES ET TECHNOLOGIE DE L'INFORMATION ET DE LA COMMUNICATION
DÉPARTEMENT INFORMATIQUE

Chapitre 3 STRUCTURES DE CONTRÔLE

Programmation en LANGAGE C
AMRT 2
2016-2017

Les structures de contrôle permettent d'agir sur **l'ordre** ou la **fréquence d'exécution** des instructions d'un programme.

Il existe deux grands types de structures de contrôle:

- **les structures conditionnelles**
- **les structures répétitives, encore appelées boucles.**

I. Structures conditionnelles

Les structures conditionnelles permettent d'exécuter des instructions différentes en fonction de certaines conditions.

I-1. L'instruction if else

L'instruction **if else** permet de conditionner l'exécution d'un ensemble d'instructions à la valeur d'une condition (expression booléenne).

La syntaxe générale de cette instruction est la suivante:

```
if (condition)
{
traitement1
}
else
{
traitement2
}
```

La condition doit être entre parenthèses.

Les traitements peuvent être constitués d'une instruction simple ou d'un ensemble d'instructions (bloc d'instructions).

Si un traitement comporte une seule instruction, on peut omettre les accolades.

La condition est d'abord évaluée. Si elle est vraie, traitement1 est exécuté sinon, traitement2 est exécuté puis le contrôle passe à l'instruction qui suit.

Remarque 1

la syntaxe de cette instruction n'impose en soi aucun point-virgule, si ce n'est ceux qui terminent naturellement les instructions simples qui y figurent.

Exemple 1

```
…
printf("entrez un nombre") ;
scanf("%d", &n) ;
if(n > 0) //dans le cas où la condition n>0 est vraie
printf("valeur positive") ;
else //dans le cas où la condition n>0 est fausse
printf ("valeur négative ou nulle") ;
…
}
```

Exemple 2

```
#include <stdio.h>
main( )
{
int nb1,nb2, res;
char op;
printf("Entrez deux nombres");
scanf("%d%d", &nb1, &nb2);
```

```
printf("entrez la première lettre de l'opération voulue: somme ou produit");
fflush(stdin);
op = getchar( );
if (op = = 's')
{res = nb1 + nb2;
printf("la somme est %d", res);
}
else
{res = nb1 * nb2;
printf("le produit est %d", res);
}}
```

Remarque 2

La clause **else n'est pas obligatoire**. Ainsi, on peut avoir le schéma suivant:

if (condition)
{
traitement
}

Si la condition est vraie, le traitement est exécuté puis le contrôle passe à l'instruction qui suit dans le programme.

Si la condition est fausse, le traitement n'est pas exécuté et le contrôle passe à l'instruction qui suit dans le programme.

Remarque 3

L'instruction if else peut contenir des clauses else if suivant le schéma suivant :

```
if (condition1)
{
traitement1
}
else if (condition2)
{
traitement2
}
...
else if (conditionN)
{
traitementN
}
else
{
Traitement
}
```

Les conditions sont évaluées dans l'ordre d'apparition. Dès qu'une condition est vraie, le traitement associé est exécuté. L'instruction suivante à exécuter sera alors celle qui suit dans le programme. Si aucune condition n'est vérifiée, alors **le traitement associée au else, s'il existe, est exécuté**.

1-2. L'instruction switch

Lorsqu'une expression conditionnelle ne se réduit pas à une alternative, on peut utiliser l'instruction **switch.** En fait, elle permet de choisir le traitement à effectuer en fonction de la valeur d'une variable ou d'une expression. Cette instruction permet parfois de remplacer avantageusement une instruction **if else.**

La syntaxe de cette instruction est

```
switch (expression)
{
case valeur1 :
traitement1;
break;
case valeur2 :
traitement2;
break;
...
case valeurN :
traitementN;
break;
default :
traitement;
}
```

L'exécution commence par les instructions de la première valeur qui correspond à la valeur l'expression du switch et continue jusqu'à un break ou la fin du switch. **Ainsi, si on ne met pas de break, les instructions du cas suivant seront également exécutées.**

La clause **default** propose un traitement à effectuer **dans le cas où la valeur de *expression ne* correspond à aucune des valeurs proposées** dans les case successifs. Cette partie n'est pas obligatoire.

Exemple

Programme qui affiche le mois en toute lettre selon son numéro. Le numéro du mois est mémorisé dans la variable mois.

```
...
printf("Donner le numéro du mois")
scanf("%d", &mois)
switch (mois)
{
case 1 : printf("Janvier"); break;
case 2 : printf("Février"); break;
...
case 12 : printf("Décembre"); break;
default : printf("Un numéro de mois doit être compris entre 1 et 12"); break;
}
...
```

Remarque

Avec une instruction if else, on aurait :

```
…
printf("Donner le numéro du mois") ;
scanf("%d ", &mois) ;
if (mois = = 1)
printf("Janvier") ;
else if (mois = = 2)
printf("Février") ;
else if (mois = = 3)
printf("Mars") ;
else if (mois = = 4)
printf("Avril") ;
…
else if (mois = = 11)
printf("Novembre") ;
else if (mois = = 12)
printf("Décembre") ;
else
printf("Un numéro de mois doit être compris entre 1 et 12") ;
…
```

II. Structures répétitives

Les structures répétitives, appelées aussi **boucles, permettent de répéter un traitement** (c'est-à-dire une instruction simple ou composée) autant de fois qu'il est nécessaire: soit un nombre déterminé de fois, soit tant qu'une condition est vraie.

II-1. La boucle while

La boucle **while permet de répéter un traitement tant qu'une condition est vraie.**

Si, dès le début, la condition est fausse, alors le traitement ne sera pas exécuté.

Syntaxe

```
while(condition)
{
traitement
}
```

Les conditions doivent être entre parenthèses.

Si un traitement comporte une seule instruction, on peut omettre les accolades.

Exemple

Programme qui calcule le cube des nombres non nuls saisis par l'utilisateur.

Tant que le nombre saisi par l'utilisateur n'est pas nul, on affiche son cube et on recommence.

```
#include<stdio.h>
main( )
{ int x ;
printf("Ce programme calcul le cube des nombres que vous entrez. Pour arrêter tapez 0.") ;
printf("Entrez un nombre") ;
scanf("%d", &x) ;
while (x != 0)
{
printf("le cube de %d est %d", x, x*x*x) ;
printf("Entrez un nombre ou 0 pour arrêter") ;
scanf("%d", &x) ;
}
printf("Fin") ;
}
```

II-2. La boucle for

La boucle **for** permet de **répéter une instruction un nombre connu de fois.**

Syntaxe

for(instruction1; condition ; instruction2)

{

traitement

}

Le déroulement se fait ainsi:

- instruction1 est exécutée **une seule fois.**
- Puis il y a l'enchaînement suivant:

La condition est évaluée.

Si elle est fausse, on sort de la boucle.

Si elle est vraie, le traitement de la boucle est exécuté, puis instruction2 est exécutée et on recommence en évaluant à nouveau la condition, etc …

En général

instruction1 permet d'initialiser une variable compteur

instruction2 permet d'incrémenter ou de décrémenter une variable compteur

Exemple

```
...
for(x = 0; x <= 20; x = x+1)
{
printf("%d", x);
}
```

A la sortie de la boucle, x vaut 21.

Remarque

La boucle for permet de faire la même chose que la boucle while mais de façon plus rapide, du moins lorsque le nombre de répétitions est connu.

En effet, avec une boucle while, on pourrait faire la même chose que l'exemple ci-dessus, mais il faudrait initialiser la variable compteur et l'incrémenter explicitement.

```
...
x = 0;
while (x <= 20)
{
printf("%d", x);
x = x+1; // incrémentation explicite
}
...
```

II-3. La boucle do…while

Cette boucle sert à répéter une instruction tant qu'une condition est vraie.

Syntaxe

do

{

traitement

}**while (condition);**

Le traitement est exécuté **une première fois, puis la condition est vérifiée.** Si elle est vraie, on retourne au début de la boucle et le traitement est répété. Si la condition est fausse, on sort de la boucle.

A chaque fois que le traitement est exécuté, la condition est de nouveau vérifiée à la fin.

Exemple

Programme qui l'aire d'un cercle dont le rayon est saisi par l'utilisateur et qui permet de recommencer plusieurs fois.

```
#include<stdio.h>
main( )
{
float rayon ;
char reponse ;
printf("Calcul de l'aire d'un cercle") ;
do
{
printf("Entrez le rayon d'un cercle en cm") ;
scanf("%f", &rayon) ;
printf("L'aire de ce cercle est %f cm2", rayon*rayon *3.14) ;
printf("Voulez-vous l'aire d'un autre cercle? (o/n)") ;
fflush(stdin);
reponse = getchar( );
}
while (reponse == 'o') ;
printf("Au revoir!") ;
}
```

III. Imbrication de structures de contrôles

Dans les parties traitement des structures de contrôle qu'on a vues, il peut y avoir d'autres structures de contrôle. On parle alors d'imbrication de structures (c'est-à-dire que les structures sont incluses les unes dans les autres).

Exemple

```
if (condition1)
{
…
if (conditionN)
{
…
}
…
}
```

- Lorsque l'on imbrique des boucles for, il faut utiliser des compteurs différents pour chaque boucle.

Exemple

```
for( j = 1 ; j <= 3 ; j++)
{
for (i = 1 ; i<= 5 ; i++)
printf("*");
printf(" \n");
}
```

- Tous les types de boucles peuvent s'imbriquer entre eux. La seule règle à respecter est que les boucles ne doivent pas se chevaucher: elles doivent s'emboîter.

IV. Instructions de rupture de séquence en langage C

IV-1. L'instruction break

L'instruction break peut être employée à l'intérieur de n'importe quelle boucle. **Elle permet d'interrompre le déroulement de la boucle,** et passe à la première instruction qui suit la boucle. En cas de boucles imbriquées, break fait sortir de la boucle la plus interne.

Exemple:

```
main()
{
int i;
for (i = 0; i < 5; i++)
{
printf("i = %d\n", i);
if (i == 3)
break;
}
printf("valeur de i a la sortie de la boucle = %d\n", i);
return 0;
}
```

affiche à l'écran:

i = 0

i = 1

i = 2

i = 3

la valeur de i a la sortie de la boucle = 3

IV-2. L'instruction continue

L'instruction **continue permet de passer directement à l'étape suivante d'une boucle sans exécuter les autres instructions de la boucle qui la suivent.**

Exemple:

```
main()
{
int i;
for (i = 0; i < 5; i++)
{
if (i == 3)
continue;
printf("i = %d\n", i);
}
printf("valeur de i a la sortie de la boucle = %d\n",i);
return 0;
}
```

affiche à l'écran:

i = 0

i = 1

i = 2

i = 4

la valeur de i a la sortie de la boucle = 5