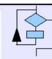
Algorithmique & Langage C
IUT GEII S1

Les boucles

Notes de cours
(troisième partie)

3

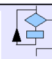
cours_algo_lgc3.08.odp



Licence

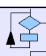
•Paternité - Pas d'Utilisation Commerciale -
• Partage des Conditions Initiales à l'Identique 2.0 France

•Vous êtes libres :
• * de reproduire, distribuer et communiquer cette création au public
• * de modifier cette création, selon les conditions suivantes :
•
•**Paternité**. Vous devez citer le nom de l'auteur original.
•
•**Pas d'Utilisation Commerciale**.
•Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.
•
•**Partage des Conditions Initiales à l'Identique**.
•Si vous modifiez, transformez ou adaptez cette création,
•vous n'avez le droit de distribuer la création qui en résulte
•que sous un contrat identiqueà celui-ci.
• * A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les
• conditions contractuelles de mise à disposition de cette création.
•* Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droit
•Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur :
•copies réservées à l'usage privé du copiste, courtes citations, parodie...)
•voir le contrat complet sous : http://fr.creativecommons.org/contrats.htm



Plan du cours

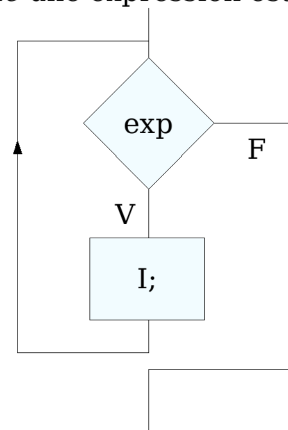
- Boucle while...
- Algorithmes à connaître
- Analyser un programme
- Boucle do...while
- Boucle for...
- Le "hasard" , l'ordinateur et le C
- Le temps universel



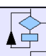
© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Boucle while...

Permet de répéter l'exécution d'une séquence d'instructions *tant que* une expression est vraie.

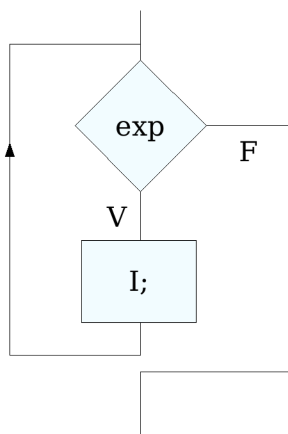


Si exp est vraie la séquence I; est exécutée. Puis l'expression est à nouveau évaluée et l'exécution de I; se poursuit *tant que* exp est vraie.



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Boucle while




```
while ( exp )
 {
    I;
 }
```


© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Boucle while

Si l'expression est fausse dès la première fois, la séquence d'instructions n'est jamais exécutée.

Si l'expression est toujours vraie, la séquence est exécutée infiniment. On a une *boucle infinie*.

En général, les instructions dans le corps de la boucle modifient une variable intervenant dans l'expression et après quelques itérations l'expression devient fausse.

Une boucle **while** permet des répéter des instructions **de 0 fois à l'infini**.

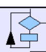
© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Exemple : boucle while

```
int i = 0 ;

while ( i < 3 )
{
 printf("A");
 i = i + 1 ;
}
```

```
int i = 3 ;

while ( i > 0 )
{
 printf("A");
 i = i - 1 ;
}
```

```
int i = 1 ;

while ( i <= 3 )
{
 printf("A");
 i = i + 1 ;
}
```

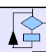


© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Algorithme à connaître

Pour répéter N fois une séquence Ins; :

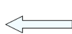
```
int i = 0;

while ( i < N )
{
    Ins ;
    i = i + 1 ;
}
```



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Analyser un programme

```
int i = 0 , j = 6 ;

while ( i <= j )
{
 printf("%d%d",j,i);
 i = i + 1 ;
 j = j – 2 ;
}
printf("%d",i);
```



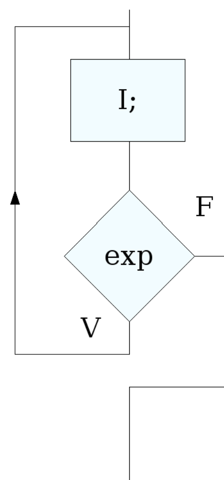
i	j	i<=j
0	6	V
1	4	V
2	2	V
3	0	F

écran

6041223

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Boucle do...while




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

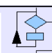
© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Boucle do...while

Comme l'expression est évaluée après l'exécution de la séquence I, la séquence est toujours exécutée au moins une fois.

Si l'expression est toujours vraie, on a une boucle infinie.

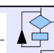
La boucle **do...while** permet de répéter des instructions de **une fois à l'infini**.



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Exemple : do...while

```
int i = 0 ;

do
{
 printf("A");
 i = i + 1 ;
}
while ( i < 3 ) ;
```

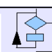


© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Toutes les boucles sont équivalentes

Un algorithme utilisant une boucle peut toujours être transformé pour produire le même résultat en utilisant l'autre boucle.

Cependant, il vaut mieux utiliser la boucle while (ou for) et réserver la boucle do...while lorsque les instructions du corps de boucle doivent être exécutées au moins une fois au minimum.

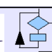


© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Un cas ou do...while est judicieuse

Demander une valeur dans un intervalle et recommencer tant que l'utilisateur répond une valeur incorrecte.

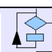
Ex : demander une valeur réelle comprise entre 1 et 2.



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

do...while

```
float x ;

do
    {
      printf("Entrez une valeur entre 1 et 2 : ");
      scanf("%f",&x);
     }
     while ( x < 1.0 ||  x>2.0 ) ;

printf("Merci !");
```

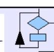


© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

do...while

Demander un nombre entre 1 et 2, cinq fois au maximum.

```
float x ;
int n = 5 ;
do  {
        printf("Entrez une valeur entre 1 et 2 : ");
        scanf("%f",&x);
        n = n – 1 ;
    }  while ( ( x < 1.0 ||  x>2.0 ) && (n > 0) ) ;

if ( x>=1.0 &&  x<=2.0 ) {
                printf("Merci !");  }
else
             {
               printf("Trop d'erreur !");  }
```



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

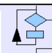
Boucle for

La boucle for est équivalente à la boucle while.

```
int i = 0 ;

while ( i < 3 )
{
 printf("A");
 i = i + 1 ;
}
```

=

```
int i = 0 ;

for ( ; i < 3 ; )
{
 printf("A");
 i = i + 1 ;
}
```



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Boucle for

while en "condensé"

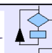
```
ins1 ;
while ( exp2 )
{
   ins ;
   ins3 ;
}
```

=

```
for (  ins1 ; exp2 ; ins3 )
{
    ins ;
}
```

Avantage : on ne risque pas d'oublier l'initialisation (Ins1)

Attention : notez bien que ins3 est exécutée après ins



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Boucle for

Qu'affiche ce programme ?

```
int i, j = 5 ;

for ( i = 0 ;  j < 9 ;  i = i + 1 )
{
    printf("%d %d", i , j ) ;
    j = j + 1 ;
}
printf("%d %d", i , j ) ;
```


© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Opérateurs ++ et --

Ces opérateurs incrémentent ou décrémentent une variable entière.
Ils peuvent être placés avant ou après une variable entière.
Placé avant l'opération est effectuée avant l'évaluation de l'expression.
Placé après l'opération est effectuée après l'évaluation de l'expression.

Ex : int i ;

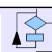
l'expression i++ a la valeur de i
et comme effet d'incrémenter i

l'expression ++i a la valeur de i+1
et comme effet d'incrémenter i

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Opérateurs ++ et --

Expression	Effet	Valeur
i++	i= i+1	i
++i	i = i+1	i + 1
i- -	i = i-1	i
- -i	i = i-1	i - 1

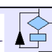


© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Opérateurs ++ et --

Qu'affiche ce programme ?

```
int i=0 , j=0 ;

for (  ; i < 5 ;  i++ )
{
    printf("%d %d", i, j++ ) ;

}
printf("%d %d", --i , j-- ) ;
```



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Opérateurs ++ et --

L'instruction

```
i = i + 1 ;
```

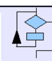
est toujours équivalente à

```
i++;
```

On s'interdit de réutiliser une variable sur laquelle on a appliqué ++ (ou --) dans une même expression, car le résultat serait indéterminé.

```
int i=2 , j ;
j  = (i++) + (2 * i) ;        j vaut ????
```

(la valeur de j dépend de l'ordre d'évaluation des opérandes de l'addition !)



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

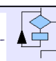
Le hasard

Le hasard intervient dans de nombreux domaines de l'informatique : simulation, cryptographie, ...

Il est impossible de produire des nombres au hasard avec un algorithme.

Tout est prévisible dans le fonctionnement d'un ordinateur.

Même si c'est difficile en pratique, il est possible de connaître l'état de la mémoire après un certain temps de fonctionnement.



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

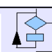
Le hasard

On peut simuler une production de nombres au hasard avec un algorithme en essayant de respecter au mieux deux propriétés essentielles :

1/ chaque nombre a la même probabilité de sortir.

2/ connaissant la suite de nombres tirés depuis de le début, il est impossible (*difficile en pratique*) de prévoir le prochain nombre qui va sortir.

Une suite produite par un algorithme qui vérifie ces propriétés est dite *pseudo-aléatoire*.



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

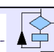
Suite pseudo-aléatoire

La fonction C **rand()** fournit un nombre peusdo-aléatoire compris entre 0 et RAND_MAX (32767 avec Dev-Cpp).

La suite peusdo-aléatoire obtenue dépend d'un paramètre initial, la graine *(seed)* fixé par la fonction **srand()**.

Pour des valeurs identiques de la graine la suite obtenue est toujours la même.

Si on n'utilise pas srand() la graine par défaut est 1.
(Ne pas utiliser srand() équivaut à faire srand(1);)



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

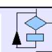
rand , srand

Chaque exécution avec la même graine produit toujours la même séquence. Pour obtenir des suites différentes à chaque exécution, on peut modifier la graine à chaque fois en utilisant par exemple l'horloge de la machine :

```
srand ( time(NULL) ) ;
```

Pour obtenir des nombres entre 0 et N, il suffit de prendre le reste dans la division par N+1 du nombre fournit par rand :

```
int x ;

  x = rand() % (N+1) ;  //  0 < x < N
```



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

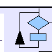
le temps

Le temps universel coordonné (UTC) est une échelle de temps adoptée comme base du temps civil international par un grand nombre de pays.

Dans les systèmes compatibles POSIX (comme linux, windows, MacOS X,BSD,...) ce temps est codé par le nombre de seconde écoulées depuis le
 1er janvier 1970 à 0heure 0minute 0 seconde.

La valeur courante peut être obtenue par la fonction time.

```
#include <time.h>

time_t t ;
t = time ( NULL ) ;
printf( " %ld" , t ) ;
```



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

le temps processeur (temps CPU)

La fonction clock() permet de mesurer le temps processeur réellement consommé par une portion de programme :

```
#include <time.h>

clock_t t0, t1 ;
double duree ;

t0 = clock() ;

.... ;
.....;
.....;

t1 = clock() ;

duree = (double) ( t1 – t0 ) / CLOCKS_PER_SEC ;
printf("Duree = %f s\n", duree );
```

(lignes « .... ; » : portion de programme dont on veut mesurer le temps CPU)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license