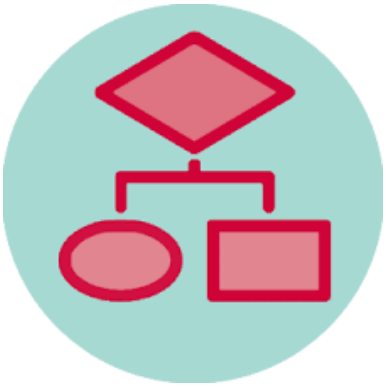
Algorithmique



Cours réalisé par Dr. Mohamed RAMDANI

Établissement: Université Mohamed Khider Biskra

Faculté: Sciences Exactes et Sciences de la Nature et de la Vie

Département: Informatique

Mail: mohamed.ramdani@univ-biskra.dz

Avril 2022

Table des matières

Chapitre 04 : Boucles

1. Introduction

Pour résoudre un problème, on pense parfois : à exécuter un ensemble d'instructions (un bloc d'instructions) plusieurs fois (ex. N fois)

Afin d'éviter de réécrire ce bloc N fois, on utilise ce qu'on a appelle **une boucle.**

Exemple

Si on veut afficher les nombres pairs inférieurs à 100, on doit écrire

et exécuter le bloc d'instructions suivant 50 fois ! :

Écrire("l'entier: ", i, "est un nombre pair."); // i est initialisé à 0

i <-- i + 2;

On utilise une boucle pour faciliter cette écriture **répétitive**

Définition

Une boucle est une instruction qui permet d'exécuter un bloc d'instructions plusieurs fois

Complément : Principe de la boucle

Pour une boucle, l'ordinateur exécute le bloc d'instructions, du **haut en bas** de manière **itérative**, jusqu'à **la satisfaction d'une condition** (un critère d'arrêt) et dans ce cas, on sort de la boucle

Remarque : le critère d'arrêt

La condition de la boucle (le critère d'arrêt) doit se vérifier à un instant donné, pour qu'on puisse sortir de la boucle et donc pour ne pas tomber dans la situation: **d'une boucle infinie**

Remarque

Il est possible qu'on entre pas dans la boucle si la condition n'est pas vérifiée dès le début

2. La boucle Tant que

Syntaxe : en algorithmique

Tant que (Condition) faire

Bloc d'instructions

Fin_tant_que;

Exemple

Algorithme BoucleTQ1;

Var

i: entier;

Début

i <- -1;

Tant que (i <> 10) faire

Écrire("Entrer le chiffre 10");

Lire(i);

Fin_tant_que;

Méthode : Principe

On commence par l'exécution de la 1ère ligne: donc on évalue la Condition;

- Si la Condition est vérifiée alors
  on exécute le Bloc d'Instructions
  lorsqu'on termine l'exécution du bloc: on itère sur la 1ère ligne (on retourne à la 1ère ligne)
  et donc on évalue à nouveau la condition;
  - Si la Condition est vérifiée pour la 2ème fois,
    on exécute à nouveau le Bloc d'Instructions
    et ainsi de suite jusqu'à la non vérification de la condition
    et dans ce cas, on sort de la boucle

Syntaxe : en langage C

while (Condition)

{

Bloc d'instructions

}

Exemple : en langage C

#include<stdio.h>

void main()

{

char c = 'N';

while (c != 'O')

{

MON-TP.....(Bloc)

printf("tu veux sortir: O/N\n");

scanf("%c", &c);

}

}

Remarque

La boucle Tant que commence par l'évaluation d'une condition ;

1. La condition est écrite par le programmeur de manière libre (sous forme d'un prédicat: utilisation des opérateurs logiques et de comparaison)
2. Durant l'écriture de la boucle, il faut être sûr qu'à un moment donné la condition ne soit pas vérifiée et donc on sort de la boucle (pour ne pas itérer infiniment)

Conseil : Auto-Formation

Youtube video

Cf. "tq"

3. La boucle Répéter

Syntaxe : en algorithmique

Répéter

Bloc d'instructions

Jusqu'à (Condition);

Exemple

Algorithme BoucleRp1;

Var

compteur : entier;

Début

compteur 1;

Répéter

Écrire("J'ai affiché la ligne : ", compteur);

compteur = compteur + 1;

Jusqu'à (compteur > 10);

Fin.

Méthode : Principe

On commence par une 1ère exécution du Bloc d'Instructions Puis on évalue la **Condition**;

- Si la Condition n'est pas **vérifiée** alors on exécute le Bloc d'Instructions à nouveau
    - et ainsi de suite jusqu'à :
      la vérification de la Condition
      et dans ce cas (vérification de la Condition), on sort de la boucle.

Syntaxe : en langage C

```
do
{
Bloc d'instructions
} while (Condition);
```

Exemple : en langage C

```
#include<stdio.h>
void main()
{
int compteur = 0;
do
{
printf("J'ai affiché la ligne: %d \n", compteur+1);
compteur++; //ou compteur = compteur + 1;
} while (compteur < 10);
}
```

Remarque

1. La boucle Répéter s'exécutera au moins une fois (l'évaluation de la condition se fait à la fin de la 1ère itération)
2. La condition est un prédicat (s'écrit par le programmeur de manière libre)

Conseil : Auto-Formation

Youtube video

Cf. "repeter"

4. La boucle Pour

Syntaxe : en algorithmique

```
Pour (<varia_entier> <-- <val_inf> jusqu'à <val_sup> [pas <val_pas>]) faire
Bloc d'instructions
Fin_pour;
```

Remarque

Si on ne mentionne pas la partie: [pas <val_pas>] (le cas par défaut), la valeur: val_pas = 1

Exemple

Algorithme Ex1Pour;

Var

i: entier;

Pour (i<-- 1 jusqu'à 10) faire

Écrire("J'ai affiché la ligne : ", i);

Fin_pour;

Méthode : Principe

La boucle **Pour** permet d'exécuter un Bloc d'Instructions de manière itérative, un nombre de fois (connu a priori),

- Au début de l'exécution de Pour: la valeur de varia_entier = val_inf, dans ce cas, on commence d'exécuter le Bloc d'Instructions pour la 1ère fois,
- par la suite, la valeur de varia_entier est incrémentée (varia_entier = val_inf + pas), dans ce cas, on exécute à nouveau le bloc
- pour la 3ème fois, la valeur varia_entier est incrémentée à nouveau (varia_entier = varia_entier + pas) de ainsi de suite ;
- On exécute le bloc pour la dernière fois lorsque varia_entier = val_sup (et donc, on sort de Pour).

Syntaxe : en langage C

for (<initialisation> ; <condition de répétition> ; <compteur>)

{

Bloc d'instructions

}

1. **Initialisation**: une instruction utilisée pour initialiser la variable incrémentale (la donnée) de la boucle
2. **Condition de répétition**: une condition utiliser pour décider de poursuivre les itérations ou non
3. **Compteur**: une instruction utilisée pour réinitialiser la donnée de la boucle (donner la valeur suivante de la donnée)

Exemple : en langage C

#include <stdio.h>

void main()

{

int i;

for (i = 0; i < 10 ; i++)

```
{
printf("J'ai affiché la ligne: %d
\n", i );
}
}
```

Conseil : Comment choisir une boucle Pour, Tant que ou Répéter?

1. Le choix de Pour : si on connaît d'avance le nombre d'itérations
2. Le choix de Tant que: si on ne connaît pas d'avance le nombre d'itérations et/ou si on a une condition (un prédicat) qui utilise les opérations logiques et/ou de comparaison
3. Le choix de Répéter: si on doit au moins exécuter une itération

Conseil : Auto-Formation

Youtube video

Cf. "pour"

5. Les boucles imbriquées

Une boucle peut contenir une autre boucle comme étant une de de ses instructions : on les appelle dans ce cas des **boucles imbriquées**,

Définition

Une boucle imbriquée est une boucle complètement inclue dans une autre boucle, de sorte que cette dernière ne peut poursuivre son itération qu'après la sortie de la première (exécution de toutes les itérations de la première),

Syntaxe

```
Début_Boucle_1
......
......
Début_Boucle_2
.....
.....
Fin_Boucle_2;
.....
.....
Fin_Boucle_1;
```

Méthode : Principe de fonctionnement:

On commence par l'exécution de la première boucle (Boucle_1)

1. et donc la première itération de cette Boucle_1
2. dans cette itération, on exécute toute la Boucle_2 (c'est-à-dire toutes les itérations de la Boucle_2)
3. et lorsqu'on sort de la Boucle_2, on termine la première itération de la Boucle_1 et on exécute la deuxième itération de la Boucle_1
4. et puis on rentre dans la Boucle_2 pour exécuter une autre fois toutes les itérations de la Boucle_2 puis on sort de cette Boucle_2 et on termine l'exécution de la Boucle_1,
5. et on passe à la 3ème itération de la Boucle_1
6. et ainsi de suite jusqu'à la fin de la Boucle_1

Remarque

Il ne faut jamais écrire deux boucles imbriquées qui se croisent; c'est-à-dire : on commence la Boucle_1, puis dans laquelle on commence la Boucle_2 et puis on termine la Boucle_1 avant la terminaison de la Boucle_2.

Conseil : Auto-Formation

Livre en Or

[cf. Livre]

6. Exercice : Structure d'un Algorithme

```
Addition;

Algorithme

x<- x+1;

Lire(x);

Ecrire(x);

Fin.

Ecrire("msg");

Var

x: entier;

Début
```