UNIVERSITÉ CENTRALE
HONORIS UNITED UNIVERSITIES

Algorithmique et Structures de Données

Année universitaire
2020-2021

Dr. Marwa CHAIEB

Chapitre 3
Les structures itératives

Plan

- Partie 1: Introduction
- Partie 2: Itérations déterministes
- Partie 3: Itérations indéterministes
- Partie 4: De l'algorithmique au langage C

Plan

- Partie 1: Introduction
- Partie 2: Itérations déterministes
- Partie 3: Itérations indéterministes
- Partie 4: De l'algorithmique au langage C

Intoduction

- Lorsque l'on veut répéter plusieurs fois un même traitement, plutôt que de copier n fois la ou les instructions, on peut demander à l'ordinateur **d'exécuter n fois une suite d'instructions**

- Il existe deux grandes catégories d'itérations :
  - Les itérations **déterministes : le nombre de boucle est défini à l'entrée de** la boucle
  - Les itérations **indéterministes : l'exécution de la prochaine boucle est** conditionnée par une expression booléenne

Plan

- Partie 1: Introduction
- Partie 2: Itérations déterministes
- Partie 3: Itérations indéterministes
- Partie 4: De l'algorithmique au langage C

Itérations déterministes

- Il existe une seule instruction permettant de faire des boucles déterministes, c'est l'instruction `pour`
- Sa syntaxe est :

  pour *identifiant d'une variable de type scalaire* ←*valeur de début* **à** *valeur de fin* **faire**
  instructions à exécuter à chaque boucle
  finpour

- dans ce cas la variable utilisée prend successivement les valeurs comprises entre *valeur de début* et *valeur de fin*

Exemple: la factorielle

Ecrire un algorithme qui calcule la factorielle d'un entier.

```
Algorithme Factorielle
Var
n, fact,i : entier
Début
        fact ← 1
        pour i ← 1 à n faire
                fact ← i * fact
        fin _pour
        Écrire ("La factorielle de n est ", fact)
Fin
```

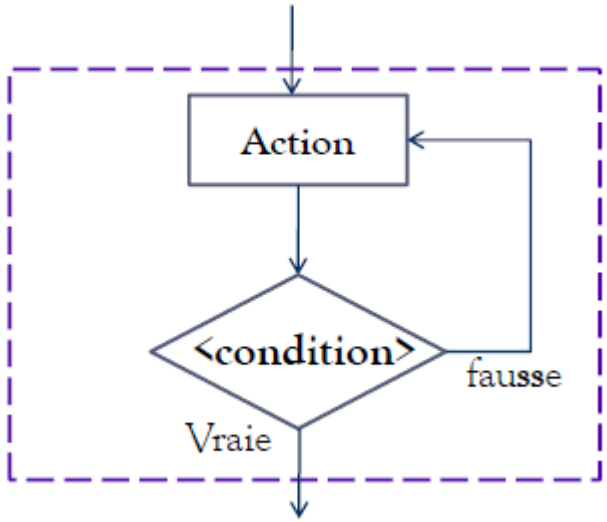
Plan

- Partie 1: Introduction
- Partie 2: Itérations déterministes
- Partie 3: Itérations indéterministes
- Partie 4: De l'algorithmique au langage C

Itérations indéterministes

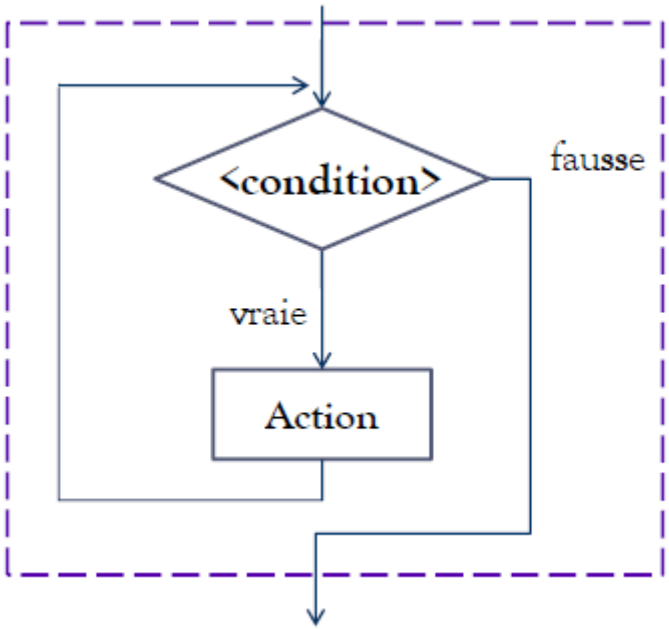
- Il existe deux instructions permettant de faire des boucles indéterministes :
  - L'instruction **tant que** :
    tant que *expression booléenne* **faire**
    instructions
    fintantque
    - qui signifie que tant que l'expression booléenne est vraie on exécute les instructions
  - L'instruction **répéter jusqu'à ce que** :
    repeter
    instructions
    jusqu'a ce que *expression boolèenne*
    - qui signifie que les instructions sont exécutées jusqu'à ce que l'expression booléenne soit vraie

Itérations indéterministes



répéter
Action
jusqu'à (condition)

Les instructions sont répétées jusqu'à la condition soit vérifiée
Action exécutée **au moins une fois**



tant que (condition) **faire**
Action
fin_tant_que

Tant que condition est vraie, les instructions sont répétées
Action exécutée 0 ou plus

Itérations indéterministes

Structure Répéter

Répéter
Instructions
Jusqu'à *Booléen*

- Arrivée à la première ligne **Répéter**, **quelque soit** la valeur du **booléen**, le programme rentre dans la **boucle**.
- La machine exécute **au moins une fois** la série d'instructions de la **boucle**.
- La machine effectue **au moins une itération** dans la boucle.

Structure Tant que

Tant que *Booléen* **Faire**
Instructions
Fin Tant que

- Arrivée à la première ligne **Tant que**, pour rentrer dans la boucle, la valeur du **booléen doit être VRAI**.
- La machine peut **ne jamais exécuter** la série d'instructions de la **boucle**.
- La machine peut **ne jamais effectuer** d'**itération** dans la **boucle**.

Exemple: contrôle de saisie

Écrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

```
Algorithme Saisie_Nombre_1

  Var : N : entier

  Début
     Répéter
        Ecrire ("Entrez un nombre entre 1 et 3")
        Lire (N)
     Jusqu'à (N >= 1 ET N <= 3)
  Fin
```

Exemple: contrôle de saisie

Écrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : "**Entrez un nombre plus petit !**", et inversement, "**Entrez un nombre plus grand !**" si le nombre est inférieur à 10.

Exemple: contrôle de saisie


```
Algorithme Saisie_Nombre_2

 Var : N : entier

 Début
    Répéter
       Ecrire ("Entrez un nombre entre 10 et 20")
       Lire (N)
       Si N < 10 Alors
          Ecrire ("Entrez un nombre plus grand !")
        SinonSi N > 20 Alors
          Ecrire ("Entrez un nombre plus petit !")
       Fin Si
    Jusqu'à (N >= 10 ET N <= 20)
Fin
```

Exemple: somme

Ecrire un algorithme qui permet le calcule de la somme des n premiers entiers positifs

```
Algorithme Somme
Var: S, i, n: entier
Début
     Répeter
          Ecrire(« Saisir n" )
          Lire(n)
    Jusqu'(n>=0)
          S←0
          pour i←0 à n faire
                    S← S+i
          Fin pour
fin
```

Exemple: le plus grand entier

Ecrire un algorithme qui détermine le plus grand entier ***e*** strictement positif tel que ***e!<=n***

```
Algorithme val_Fact
Var: fact, e, n: entier
Début
    Répeter
        Ecrire(« Saisir n" )
        Lire(n)
    Jusqu'(n>=0)
        fact←1
        e← 1
        tantque (fact<=n) faire
                e←e+1
                fact←fact*e
        Fin tantque
e←e-1
fin
```

Plan

- Partie 1: Introduction
- Partie 2: Itérations déterministes
- Partie 3: Itérations indéterministes
- Partie 4: De l'algorithmique au langage C

Traduire les itérations

for

L'instruction Pour est traduite par l'instruction `for`, qui a la syntaxe suivante :

```
for( initialisation ;
     Condition de répétition;
     operation_effectuée_à_chaque_itération )
  instruction ;
```

Exemple

```
for(i=0;i<10;i++)
  printf ("%d\n", i);
```

Traduire les itérations

while

L'instruction `Tant...que` est traduite par l'instruction `while`, qui a la syntaxe suivante :

```
while(condition)
    instruction ;
```

Exemple

```
i=0;
while(i<10){
   printf ("%d\n", i);
   i++;
}
```

Traduire les itérations

do..while

L'instruction **répéter** est traduite par l'instruction **do..while**, qui a la syntaxe suivante :

```
do
      instruction ;
while( condition );
```

Exemple

```
i=0;
do {
   printf ("%d\n", i);
   i++;
} while (i<=10);
```

Instructions while, do/while et for


```
int N=10;

int i;


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

```
int N=10;

int i;


i = 1;
do {
 printf("%d",i);
 i++;
} while ( i<=N );
```

```
int N=10;

int i;


for(i=1;i<=N;i++)
 printf("%d",i);
```

Exercice 1

Exercice :

Ecrire un algorithme et sa traduction en C d'un programme qui calcule le PGCD de 2 entiers a et b.

PGCD(a,b) = PGCD(a-b,b) si a > b
= PGCD(a,a-b) si a< b

```
main()
{
    int a,b ;
    scanf(''%d %d'',&a,&b) ;
            while(a !=b)
            {
            if (a > b)  a=a-b;
            else b=b-a;
            }
   Printf("%d",a);
}
```

Exercice 2

Exercice :

Ecrire un algorithme puis un programme C qui compte la fréquence des voyelles A, a, E, e, I, i, O, o, U et u dans une saisie d'un texte qui se termine par le caractère '#' (lecture caractère par caractère), puis affiche le résultat sous la forme suivante :

A, a : 5
E, e : 8
I, i : 0
O, o : 3
U, u : 10

Les voyelles minuscules et majuscules sont comptées ensemble.

Utiliser la fonction C getchar() qui permet de lire un caractère saisie au clavier