COURS ALGORITHME : FONCTION ET BOUCLES

CHARLES 'ARYS' YAICHE

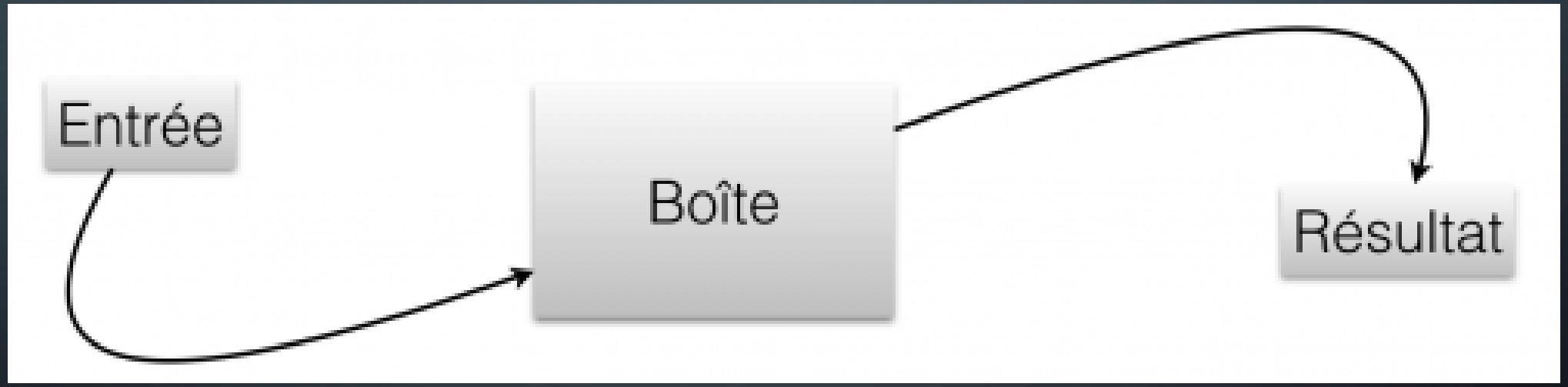
RAPPEL

- introduction au langage algo
- définition d'un algo
- Variable
- Constante
- Type
- structure conditionnel

LES ROUTINES : FONCTIONS ET PROCÉDURES

- Lorsqu'un programme est très long, il n'est pas réaliste de le coder d'un seul tenant.
- Le programme est décomposé en de plus petites unités ou parties réutilisables qui sont ensuite appelées le moment opportun par le programme principal, ces unités sont appelé routines ou sous-programme.
- Une routine évite la répétition inutile de code et permet de clarifier le programme.

LES FONCTIONS



FONCTION ET PARAMÈTRES (ENTRÉE)

- Toute routine peut définir des paramètres, qui sont autant de données en entrée, fournies lors de l'appel à la routine depuis un autre endroit de l'algorithme. La structure de ces paramètres est définie immédiatement après l'entête de la routine (comme indiqué dans la syntaxe des procédures et fonctions ci-dessous). On distingue deux catégories de paramètres : les *locaux* et les *globaux*.
- Les paramètres locaux indiquent que le paramètre effectif passé lors de l'appel à la routine sera en fait copié localement, et que la routine travaillera alors sur la copie locale (on parle de *passage par valeur*). De la sorte, toute modification effectuée par la routine ne sera pas répercutée sur la donnée passée en paramètre. Ce qui fait qu'un paramètre local peut recevoir une expression comme paramètre effectif, et non pas obligatoirement une variable.
- Les paramètres globaux ne gérèrent pas de copie locale du paramètre effectif. Ils ne font qu'un avec la donnée qui leur est passée. En appelant une routine qui a des paramètres globaux, il faut alors fournir pour ceux-ci un nom de variable existante, et non une expression, puisque la routine est susceptible de modifier la variable passée (on parle de *passage par variable* ou *par adresse*).

DÉCLARATION DES PARAMÈTRES

```
parametres locaux
    ident_type  ident_param1, ident_param2, ...
    ...

parametres globaux
    ident_type  ident_param1, ident_param2, ...
    ...
```

DÉCLARATION DE FONCTION

```
algorithme fonction ident_fonction : ident_type_retourné
        [<declarations parametres>]
        [<declarations>]
debut
        <instructions>
fin algorithme fonction ident_fonction
```

PROCÉDURE RETOURNE

- Une fonction retourne obligatoirement une valeur via la procédure retourne
- Le retourne est débranchant (son exécution termine la fonction)

PROCÉDURE

- Une procédure est une routine n'ayant pas de valeurs de retour

```
algorithme procedure ident_procedure
        [<declarations parametres>]
        [<declarations>]
debut
        <instructions>
fin algorithme procedure ident_procedure
```

LES STRUCTURES ITÉRATIVES (LES BOUCLES)

Une **itération** ou **structure itérative** est une séquence d'instructions destinée à être exécutée plusieurs fois. C'est aussi l'action d'exécuter cette instruction. Vous entendrez parler parfois de **structures répétitives**, c'est la même chose dite autrement.

TANT QUE

tant que ... fin tant que

- **Syntaxe**

```
tant que <expression booléenne> faire
        <instructions>
fin tant que
```

- Les instructions sont répétées tant que la condition est vérifiée. Comme le test est au début, les instructions peuvent donc ne jamais être exécutées.
- **Attention :** il est impératif que la condition devienne fausse à un moment. Pour cela il faut que l'expression booléenne contienne au moins une variable qui sera modifiée dans la boucle.

FAIRE … TANT QUE

faire ... tant que

- **Syntaxe**

```
faire
        <instructions>
tant que <expression booléenne>
```

La condition est placée après les instructions, elles sont exécutées donc au moins une fois, et persistent tant que la condition reste satisfaite. Bien entendu, même contrainte quant à l'expression booléenne.

BREAK & CONTINUE

- Faites toujours bien attention à ce que votre **boucle** dispose d'une **condition de sortie.** En effet rien ne vous empêche de faire des boucles infinies. Dans le cas d'une structure itérative « tant que », il suffit que la condition soit toujours VRAIE, Pour sortir d'une **boucle** on utilisera le mot clef **break**
- Pour aller directement à **l'itération** suivante, utiliser le mot clef **continue**

POUR

L'itérative : `pour ... fin pour`

- **Syntaxe**

Elle permet de répéter une série d'instructions un nombre déterminé de fois.

```
pour ident_var ← <expr_debut> jusqu'a <expr_fin> [decroissant] faire
        <instructions>
fin pour
```

QUELLE BOUCLE CHOISIR

- On emploie une **structure « Pour »** lorsqu'on connaît à l'avance le nombre **d'itérations** nécessaires au traitement. Ce nombre peut être fixe ou calculé par avance avant la **boucle,** peu importe. La **boucle "Pour"** est **déterministe** : son nombre **d'itérations** est fixé une fois pour toute et est en principe **invariable** (bien qu'il reste possible de tricher).
- On emploie les **structures « tant que »** et **« faire .. tant que »** lorsqu'on ne connaît pas forcément à l'avance le nombre d'**itérations** qui seront nécessaires à l'obtention du résultat souhaité. par avance), d'enregistrements dans une base de données, ça peut être aussi la lecture des lignes d'un fichier (on n'en connaît pas le nombre, un calcul complexe devant retourner une précision particulière, un nombre de tours dans un jeu (le premier qui gagne sort de la boucle), etc.

DIFFICULTÉ DES EXO

- Facile
- Moyen
- difficile