# La syntaxe : les diagrammes de Conway

## *1. Grammaire*

Le langage humain est composé de deux choses :

* vocabulaire
* grammaire

Les langages de programmation sont analogues aux langages naturels :

- décrits par une grammaire rigide
- composés d'un vocabulaire restreint.

Le langage C n'échappe pas à cette règle.

## *2. Définition d'une grammaire :*

Une grammaire est définie par un quadruplet.

1. Ensemble de symboles terminaux (Ex. : homme, femme, voiture…)
2. Ensemble de concepts (Ex. : phrase, verbe, complément…)
3. Concept initial, c'est l'élément de l'ensemble que l'on doit développer au départ (Ex.: phrase)
4. Ensemble de règles de grammaire (Ex. : phrase → sujet verbe complément).

## *3. Diagrammes de "Conway" :*

Ce type de représentation est le plus utilisé pour les langages de programmation à cause de sa lisibilité et de sa souplesse d'expression.

* formes arrondies → symboles terminaux (le) (chat)
* formes rectangulaires → concept [sujet] [verbe]
* les éléments sont reliés par des traits et des flèches

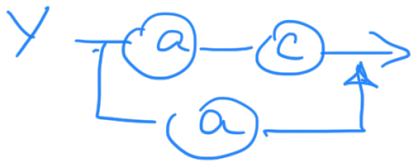
Règles :

- Chaque diagramme décrit un concept.
- Un diagramme se lit dans le sens de la flèche
- Il existe des formes alternatives, itératives et récursives

## *4. Processus de dérivation*

- Le processus de dérivation consiste à remplacer un concept par les éléments rencontrés dans le diagramme qui le décrit.
- Soit un concept (le concept initial) on applique le processus de dérivation jusqu'à qu'il n'y ait plus que des éléments terminaux dans la phrase.
- On obtient alors une phrase du langage

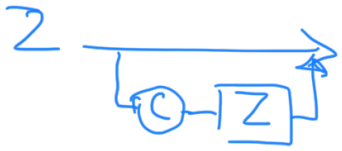
*Exemple*


forme itérative


forme récursive


une suite d'au moins 1 c


une suite éventuellement vide de c

concepts $\{X, Y, Z\}$

éléments terminaux $\{a, b, c\}$

concept initial $X$

$X \rightarrow Y b Z$

$Y \rightarrow a c$

$Z \rightarrow c Z$

$Z \rightarrow c$

$Y b Z$

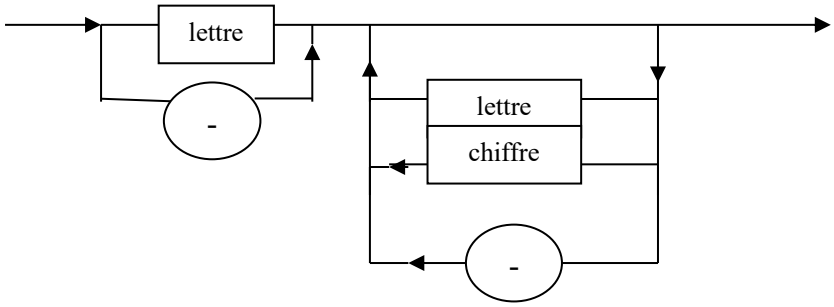
$a c b Z$

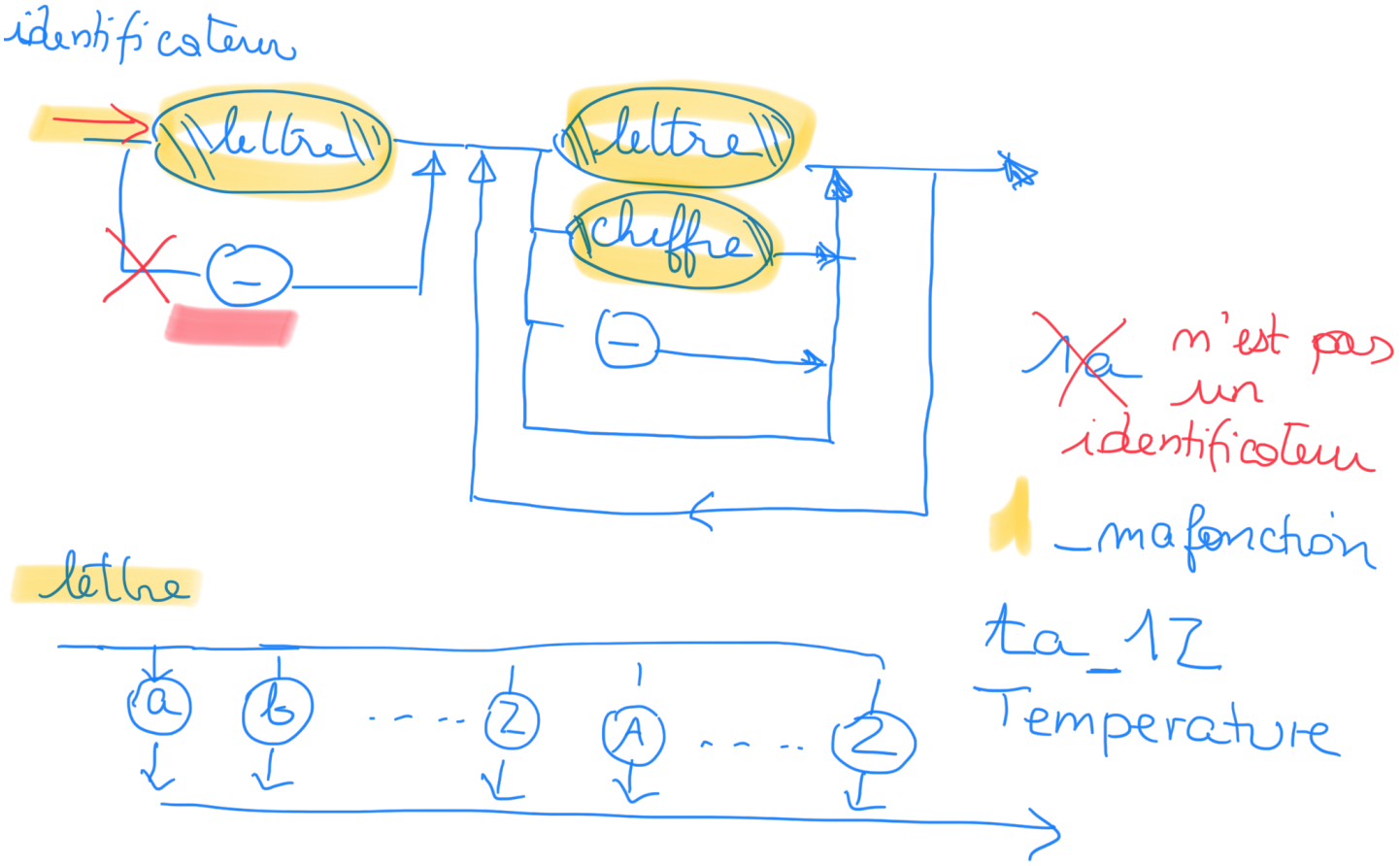
$a c b c Z$

$a c b c c$

Exemple du langage C (diagrammes de Conway sur le site moodle)

Identificateur :



# **Chapitre 2 :** Introduction au langage C

Généralités :

- Conçu dans les années 70 par Dennis Ritchie → écrit pour le système d'exploitation UNIX
- Pas vraiment pédagogique # PASCAL
- Usage général, largement répandu compilateurs performants

Principale différence avec le PASCAL : les pointeurs, Instructions plus bas niveau
Principale différence avec le python : langage compilé, les pointeurs

## *1. Caractéristiques*

1. Toutes les variables, données, constantes doivent être déclarées
2. Leurs types (variables, données) doivent être explicitement définis
3. Possède un type pointeur.

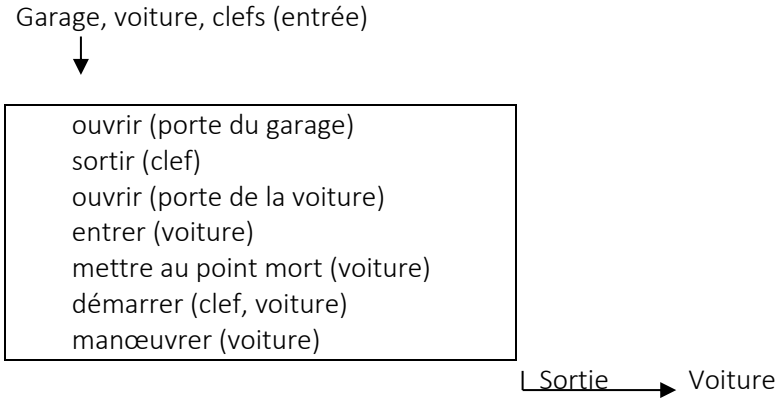
Structure d'un programme.

1. Organisation du programme en « blocs d'instructions » emboîtés ou en tâches
2. Utilisation d'identificateurs pour spécifier les blocs ou les tâches
3. Utilisation d'indentations pour visualiser l'architecture du programme
4. Utilisation possible de la récursivité.

La tâche : Une action peut être considérée comme une tâche.

Reprendre l'exemple de la sortie du garage :

Actions :




Si les actions décrites dans une tâche sont connues par le langage, alors on peut directement traduire l'algorithme sinon il faut les décrire sous forme de sous-tâches : fonctions

- tâche correspondant au programme → fonction main ( )
- sous tâche décrite est une fonction → fonction ma_fonction ( )

main est un mot réservé

Chaque tâche est décrite par

- Un entête comprenant nom de la tâche et les interactions de la tâche avec les autres tâches (Entrées/Sorties)
- Un corps (ou bloc) comprenant les objets propres à la tâche et les actions

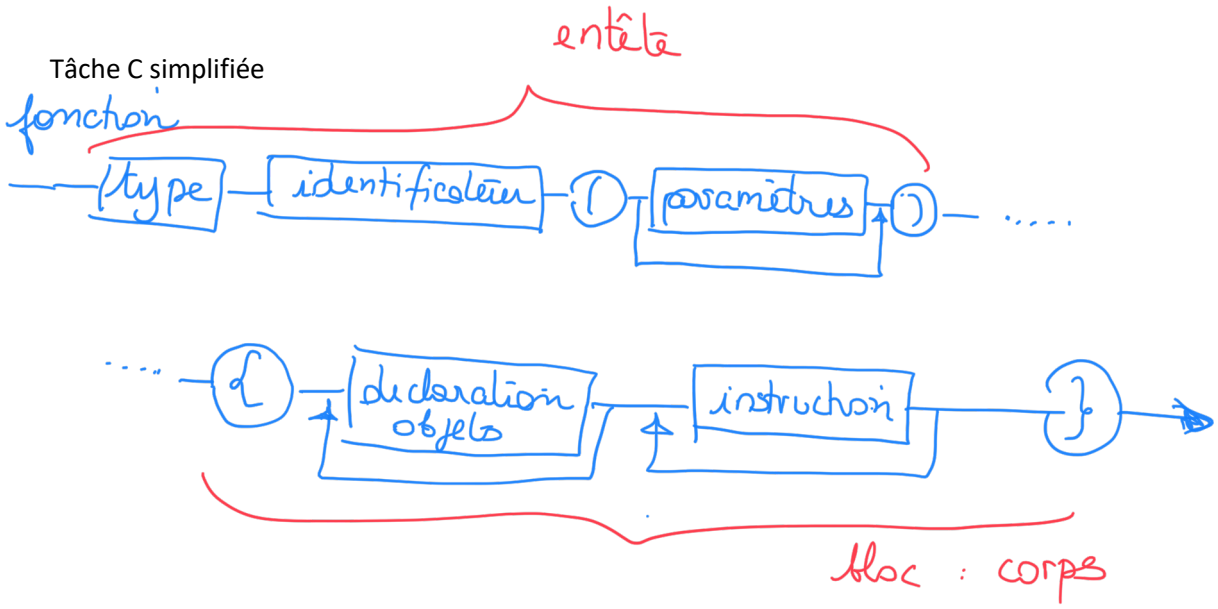
Certaines sous tâches communément utilisées sont définies dans des librairies

Ex. : lecture, écriture (traduction de lire et affiche)
Ex. : fonctions mathématiques

Pour invoquer ces librairies on utilise une directive de compilation

```
#include  <stdio.h>
#include  <math.h>
```

Tâche C simplifiée



## 2. Ecriture d'un programme

```
#include <stdio.h>      /* appel à la librairie d'entrées/sorties */

/* calcul de l'aire d'un cercle */      /* commentaires du programme */
int main( )             /* définition de la tâche principale */
{
    /* déclaration des objets de la fonction */
    const float pi = 3.14159;
    float rayon;     /* donnée */
    float aire;      /* résultat */

    /* saisie des données */
    printf ("  donner le rayon : ") ;
    scanf  ("%f " , &rayon) ;

    /* calcul */
    aire = pi * rayon * rayon ;

    /* affichage */
    printf ("l'aire du cercle = %f " , aire) ;

    return 0 ;    /* valeur retournée par le  programme C */
}
```

Nom de la tâche → principale → main
Objets entrant - sortant → aucun ( )
Type de la tâche → int (elle retourne un entier qui sera analysé par le système d'exploitation : windows, linux ,macOs etc….)
Objets de la tâche ⇒ une constante pi
⇒ deux variables rayon, aire
actions : écriture, lecture, calcul, écriture

## 3. *Classes d'unité syntaxique*

Le langage « C » comporte 6 classes d'unités syntaxiques :

- les séparateurs
- les mots réservés
- les indicateurs
- les opérateurs
- les commentaires
- les délimiteurs

### 3.1. Les séparateurs :

Ils servent à délimiter les unités syntaxiques :

- espaces, plusieurs espaces, tabulation ou aller à la ligne.

On peut en mettre autant que l'on veut.

ex float rayon;
floatrayon;
float rayon;
float
rayon;

### 3.2. Les commentaires :

Un programme sans commentaire n'est pas un programme correct.

Les commentaires sont compris entre /* et */

```
/***************************/
/*                         */
/*    Calcul d'une aire    */
/*                         */
/***************************/
```

Ils sont souvent positionnés en fin de ligne.

```
x = NB % 10 ;  /* x = dernier chiffre de droite */
```

x = NB % 10; // sur une seule ligne

Il ne faut pas imbriquer les commentaires entre eux

/* ..... /* ..... */ .....*/ ?

### 3.3. Les identificateurs :

Ils permettent de référencer les différents objets du C

- Constantes
- Variables
- Les fonctions.

*(Voir Diagrammes de CONWAY)*

Les mots réservés du langage sont interdits comme identificateur.

### 3.4. Les mots réservés :

Symboles terminaux du langage, ils sont nécessaires à la sémantique du langage.
Ce sont :
- spécificateur de type d'objet : `int, char, float, double, struct` ...
- spécificateur de classe d'allocation d'un objet : `auto, extern, typedef`…
- opérateurs symboliques : `if, else, while, switch`…
- étiquettes : `default`…

### 3.5. Les opérateurs :

Ils sont représentés par 1 ou 2 caractères spéciaux.
Il existe 3 classes d'opérateurs :
Unaires, ils sont devant ou derrière un identificateur -, *, +, ++,
Binaires, mettent en relation deux termes ou expressions == , + , * , /, >> …
Ternaires, mettent en relation 3 termes ou expressions ? :

Le chapitre suivant décrit en détail les opérateurs du langage C.

### 3.6. Les délimiteurs :

; : termine une déclaration de variables ou une instruction
, : sépare deux éléments dans une liste
( ) : encadre une liste d'arguments ou de paramètres
[ ] : encadre une dimension ou l'indice d'un tableau
{ } : encadre un bloc d'instructions ou une liste de valeurs d'initialisations

## *4. Description des objets d'un bloc*

* Les variables sont les outils de base de la programmation.
* Les variables regroupent les variables et certaines données vues dans le chapitre *algorithmes*
* Déclarer une variable = définir une carte d'identité de la variable
* Toute variable doit être déclarée avant usage.

### 4.1. Déclaration d'une variable

Syntaxe : (cf diagramme de Conway)

Carte d'identité
- nom ( identificateur)
- type
- adresse → donnée par le compilateur
- valeur

Exemple `int x;`

mémoire

1020 nom : i
1024 type : entier
1028 adresse : 1020 (non géré par le programmeur)
valeur : ?

- identificateur : Commence par une lettre puis une suite de chiffres ou de lettres sans accent. Nombre maximum de caractères : défini par le compilateur majuscules et minuscules différenciées.
- adresse : lieu de stockage dans la mémoire. Avant d'être stockée une variable est codée suivant son type. Son codage peut utiliser un ou plusieurs « mots » mémoires.


- types ⇒ types simples

Les types de base du langage « C » définissent la taille de l'emplacement mémoire et leur codage. Il existe des mécanismes de conversion de type.

## 4.2. Les types simples :

Ils dépendent de la machine.

entiers :

| | | |
|---|---|---|
| int | (- 32768 → + 32767) | 16 bits |
| short (½ * int) | (- 128 → + 127) | 8 bits |
| long 2 x int | (- 2147483648 → + 2147483647) | 32 bits |

ex : int i, j, x ;

Caractères :

char 8 bits

ex : char a, b ;

Réels : float ± 3.4 x $10^{38}$ 6 chiffres après la virgule
double ± 1.8 x $10^{308}$ 15 chiffres après la virgule

Vide void

Ex : void ma_fonction()

Remarque :
Les types entiers peuvent être attribués du qualificatif unsigned précisant que les valeurs seront positives unsigned short i ⇒ (0 → 255).

## 4.3. Les valeurs littérales (vues en TD)

Ce sont les valeurs qui peuvent prendre les variables ou constantes.

Entiers : 123 -34 décimal
023 octal
0x7FF hexa

Entiers longs : 123456789L

Réels : 9.73 3.0e6 12.e6
10.1E-3 25.

Caractères : `a´ `z´ `J´ `$´

Chaînes de caractères : " Je suis en LO01 et j'aime ça "
`a´ → caractère a
"a" → chaîne de caractères a \0
Toute chaîne est toujours terminée par \0

Caractères spéciaux

\n retour à la ligne
\t tabulation
\" double quote
\0 fin de chaîne

Initialiser la variable à la déclaration

```
int   i = 0 ;     float e = 2.7179 ;
char   a ='a' ;    int heure =60*60 ;
```

## 4.4. Portée des identificateurs

Les variables ou constantes sont déclarées à l'intérieur d'un bloc.
La validité de ces identificateurs est limitée à ce bloc.

```
bloc  1 ( )
{ int année ;
}
bloc 2 ( )
{ float température ;
                    ⇐ année n'est pas connue
}
ou    bloc 1 ( )
{ int salaire ;
 salaire = 10 00 ;
}
bloc 2 ( )
{float salaire ;
salaire = 3 205.25 ;
}
```

## 5. *Les opérateurs du langage C*

Les opérateurs permettent de construire des expressions à partir de constantes et de variables. Chaque opération est évaluée et produit un résultat.

### 5.1. Opérateurs arithmétiques

5+2
7

+
–
*
/
% (modulo)

$y = a * a * a ; \Rightarrow y = pow (a, 3)$

$a + b * c \quad \Rightarrow a + (b * c)$

5.0 + 2.0
7.0

Le résultat est du même type que les opérandes (entiers ou réels)

opérandes doivent être du même type

### 5.2. Opérateurs relationnels

>
>=
<
<=
==
!=

égalité

différence

$a > b$

opérateurs de comparaison utilisés principalement dans les tests

5 >= 3
1 (vrai)

8 != 8
0 (faux)

Remarque importante : le résultat d'une opération relationnelle est un entier et prend la valeur 1 (VRAI) ou 0 (FAUX)

### 5.3. Opérateurs logiques

! négation (NOT)

&& ET logique (AND)

|| OU logique (OR)

*Exemple:* `if (x < y && y < z) ……`

| et | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| ou | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

| ! | |
|---|---|
| 0 | 1 |
| 1 | 0 |

## 5.4. Opérateurs de manipulation de bits (non vus en LO01)

| | | |
|---|---|---|
| & | ET | |
| I | OU | |
| ^ | ou logique exclusif (EOR) | x « 3 |
| ~ | complémentation à 1 | |
| « | décalage vers la gauche | |
| » | décalage vers la droite | |

## 5.5. Les opérateurs d'affectation

### Opérateur d'affectation =

*variable = expression*

l'expression de droite est évaluée puis stockée dans la variable

x = 3 ;
m = c > = 'a' && c < = 'z' ;
x = y = z = 0 ; de la droite vers la gauche.

*variable op= expression*

équivalent à *variable = variable op (expression)*

+ =
- =
* =
\ =
% =

ex : j * = i + 1 ⇒ j = j * (i + 1)

### Opérateurs d'incrémentation et de décrémentation :

| | |
|---|---|
| + + | incrémente de 1 la variable |
| -- | décrémente de 1 la variable |

+ +x et x++ ⇒ x = x+ 1 ;
--x et x-- ⇒ x = x - 1 ;

En préfixe : --x ou ++x la variable est incrémentée ou décrémentée avant son utilisation.
En suffixe : x++ x -- la variable est incrémentée ou décrémentée après son utilisation.