Introduction à la programmation en C

Séance 1 : compilation, lexique, syntaxe, sémantique, types élémentaires, expressions

Luca SAIU

http://ageinghacker.net

IUT de Villetaneuse, Université Paris 13

Septembre 2018

Sommaire

À propos de vous

- Dans quels langages avez-vous programmé ?

À propos de moi

- Luca SAIU
  - `http://ageinghacker.net`
  - Vous trouvez la page web officielle du cours en suivant le lien “*Teaching*” ;
  - Je suis facile à contacter (*n'utilisez pas l'ENT*) ;
- J'aime la programmation
  - C est un des mes langages préférés ;
  - je l'utilise dans mes projets “sérieux” ;
    - Le projet GNU : `http://www.gnu.org`

À propos de moi

- Luca SAIU
  - `http://ageinghacker.net`
  - Vous trouvez la page web officielle du cours en suivant le lien "*Teaching*" ;
  - Je suis facile à contacter (*n'utilisez pas l'ENT*) ;
- J'aime la programmation
  - C est un des mes langages préférés ;
  - je l'utilise dans mes projets "sérieux" ;
    - Le projet GNU : `http://www.gnu.org`

Le langage C

- 1970s, Kernighan et Ritchie
- évolué, maintenant un standard ISO
- Le langage d'implémentation du système d'exploitation original Unix
  - systèmes plus anciens écrits en assembleur, non portables
  - C est également utilisé pour les systèmes style Unix modernes (GNU, BSD, ...)
- Contrôle fin de la machine et des performances...
- ...et portable *si bien utilisé*

Le langage C

- 1970s, Kernighan et Ritchie
- évolué, maintenant un standard ISO
- Le langage d'implémentation du système d'exploitation original Unix
  - systèmes plus anciens écrits en assembleur, non portables
  - C est également utilisé pour les systèmes style Unix modernes (GNU, BSD, ...)
- Contrôle fin de la machine et des performances...
- ...et portable *si bien utilisé*

Le langage C

- 1970s, Kernighan et Ritchie
- évolué, maintenant un standard ISO
- Le langage d'implémentation du système d'exploitation original Unix
  - systèmes plus anciens écrits en assembleur, non portables
  - C est également utilisé pour les systèmes style Unix modernes (GNU, BSD, ...)
- Contrôle fin de la machine et des performances...
- ...et portable *si bien utilisé*

Le langage C

- 1970s, Kernighan et Ritchie
- évolué, maintenant un standard ISO
- Le langage d'implémentation du système d'exploitation original Unix
  - systèmes plus anciens écrits en assembleur, non portables
  - C est également utilisé pour les systèmes style Unix modernes (GNU, BSD, . . .)
- Contrôle fin de la machine et des performances. . .
- . . . et portable *si bien utilisé*

Le langage C

- 1970s, Kernighan et Ritchie
- évolué, maintenant un standard ISO
- Le langage d'implémentation du système d'exploitation original Unix
  - systèmes plus anciens écrits en assembleur, non portables
  - C est également utilisé pour les systèmes style Unix modernes (GNU, BSD, . . .)
- Contrôle fin de la machine et des performances. . .
- . . . et portable *si bien utilisé*

Le langage C

- 1970s, Kernighan et Ritchie
- évolué, maintenant un standard ISO
- Le langage d'implémentation du système d'exploitation original Unix
  - systèmes plus anciens écrits en assembleur, non portables
  - C est également utilisé pour les systèmes style Unix modernes (GNU, BSD, . . .)
- Contrôle fin de la machine et des performances. . .
- . . . et portable *si bien utilisé*

Le langage C : avantages

C est un langage de production, pour des systèmes efficaces :

- expressif, puissant ;
- relativement simple ;
- portable si bien utilisé ;
- donne accès au matériel ;
  - systèmes d'exploitation, pilotes, communication avec les dispositifs, code assembler embarqué
- potentiellement très efficace . . .
  - *[demo : C et Python]*
- Intellectuellement satisfaisant.

Le langage C : avantages

C est un langage de production, pour des systèmes efficaces :

- expressif, puissant ;
- relativement simple ;
- portable si bien utilisé ;
- donne accès au matériel ;
  - systèmes d'exploitation, pilotes, communication avec les dispositifs, code assembler embarqué
- potentiellement très efficace . . .
  - *[demo : C et Python]*
- Intellectuellement satisfaisant.

Le langage C : avantages

C est un langage de production, pour des systèmes efficaces :

- expressif, puissant ;
- relativement simple ;
- portable si bien utilisé ;
- donne accès au matériel ;
  - systèmes d'exploitation, pilotes, communication avec les dispositifs, code assembler embarqué
- potentiellement très efficace . . .
  - *[demo : C et Python]*
- Intellectuellement satisfaisant.

Le langage C : avantages

C est un langage de production, pour des systèmes efficaces :

- expressif, puissant ;
- relativement simple ;
- portable si bien utilisé ;
- donne accès au matériel ;
  - systèmes d'exploitation, pilotes, communication avec les dispositifs, code assembler embarqué
- potentiellement très efficace . . .
  - *[demo : C et Python]*
- Intellectuellement satisfaisant.

Le langage C : avantages

C est un langage de production, pour des systèmes efficaces :

- expressif, puissant ;
- relativement simple ;
- portable si bien utilisé ;
- donne accès au matériel ;
  - systèmes d'exploitation, pilotes, communication avec les dispositifs, code assembler embarqué
- potentiellement très efficace . . .
  - *[demo : C et Python]*
- Intellectuellement satisfaisant.

Le langage C : avantages

C est un langage de production, pour des systèmes efficaces :

- expressif, puissant ;
- relativement simple ;
- portable si bien utilisé ;
- donne accès au matériel ;
  - systèmes d'exploitation, pilotes, communication avec les dispositifs, code assembler embarqué
- potentiellement très efficace . . .
  - *[demo : C et Python]*
- Intellectuellement satisfaisant.

Le langage C : avantages

C est un langage de production, pour des systèmes efficaces :

- expressif, puissant ;
- relativement simple ;
- portable si bien utilisé ;
- donne accès au matériel ;
  - systèmes d'exploitation, pilotes, communication avec les dispositifs, code assembler embarqué
- potentiellement très efficace . . .
  - *[demo : C et Python]*

- Intellectuellement satisfaisant.

Le langage C : avantages

C est un langage de production, pour des systèmes efficaces :

- expressif, puissant ;
- relativement simple ;
- portable si bien utilisé ;
- donne accès au matériel ;
  - systèmes d'exploitation, pilotes, communication avec les dispositifs, code assembler embarqué
- potentiellement très efficace . . .
  - *[demo : C et Python]*

- Intellectuellement satisfaisant.

Le langage C : désavantages

- Un langage « dangereux »...
  - non nécessairement robuste ;
  - non nécessairement sûr
  - ...mais on peut prévenir ces problèmes ;
- (Pas le plus compact).

Le langage C : désavantages

- Un langage « dangereux »...
  - non nécessairement robuste ;
  - non nécessairement sûr
  - ...mais on peut prévenir ces problèmes ;
- (Pas le plus compact).

Le langage C : désavantages

- Un langage « dangereux »...
  - non nécessairement robuste ;
  - non nécessairement sûr
  - ...mais on peut prévenir ces problèmes ;
- (Pas le plus compact).

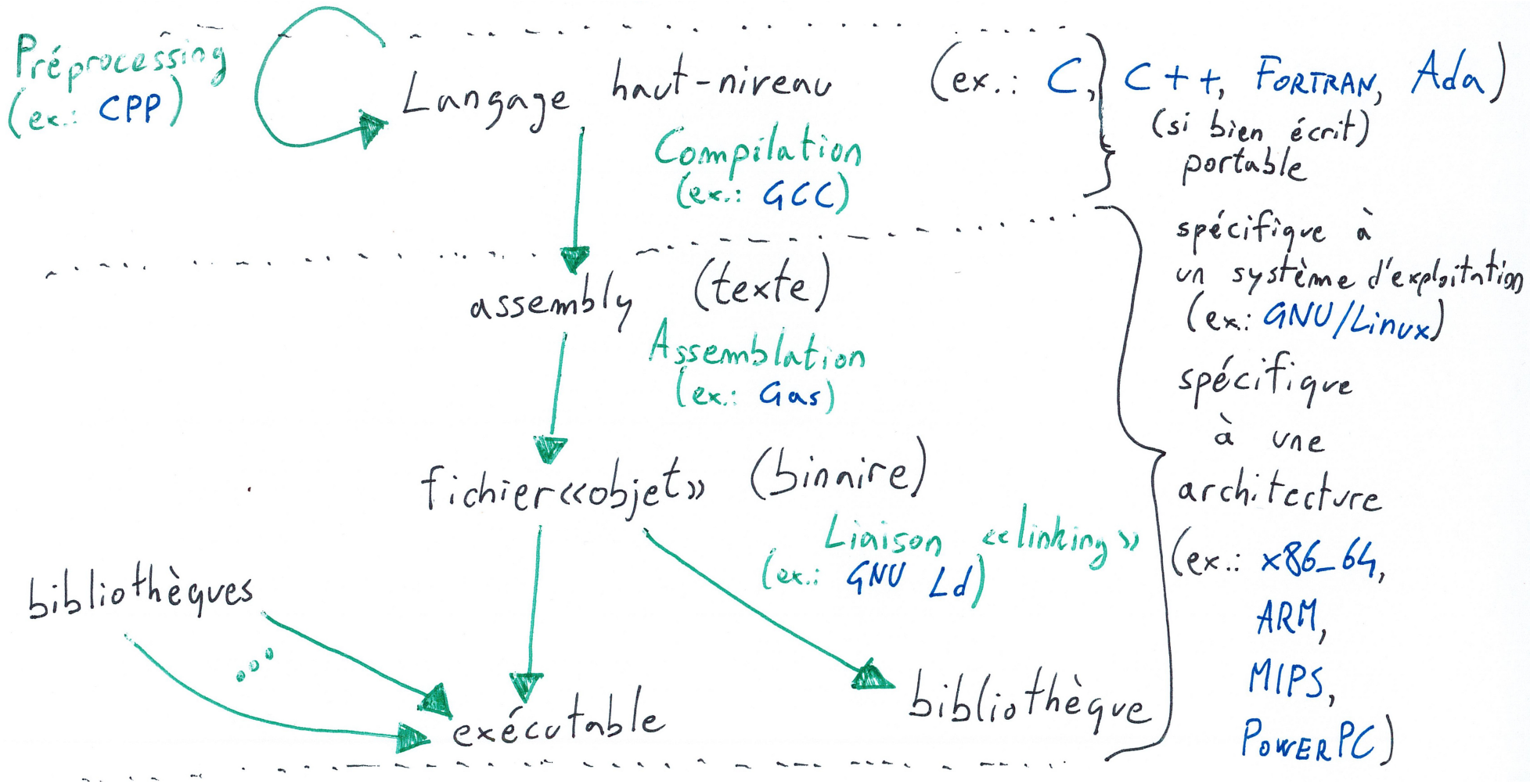
Le langage C : désavantages

- Un langage « dangereux »...
  - non nécessairement robuste ;
  - non nécessairement sûr
  - ...mais on peut prévenir ces problèmes ;
- (Pas le plus compact).

Implémenté par un compilataur

[demo : source, assembleur, langage machine]

Compilation et autres phases



Facile en pratique

GCC appelle automatiquement les autres programmes, en tout cas typique.

Ligne de commande conseillée :

```
gcc autresoptions -o destination source.c
```

pour compiler *source.c* en générant un fichier exécutable *destination* (déjà assemblé et lié).

Le fichier *destination* précedent sera effacé, s'il existe !

Facile en pratique

GCC appelle automatiquement les autres programmes, en tout cas typique.

Ligne de commande conseillée :

```
gcc autresoptions -o destination source.c
```

pour compiler *source.c* en générant un fichier exécutable *destination* (déjà assemblé et lié).

Le fichier *destination* précedent sera effacé, s'il existe !

Conseils pratiques : compilation

Vous voulez activer les warnings, et optimiser.

Certains bugs sont plus facilement visibles avec les optimisations activées (*et ils sont des vrais bugs*).

```
gcc -Wall -O2 -o destination source.c
```

Si vous ne comprenez pas un message du compilateur, demandez-moi.

N'ignorez jamais ce que le compilateur vous dit !

Conseils pratiques : compilation

Vous voulez activer les warnings, et optimiser.

Certains bugs sont plus facilement visibles avec les optimisations activées (*et ils sont des vrais bugs*).

```
gcc -Wall -O2 -o destination source.c
```

Si vous ne comprenez pas un message du compilateur, demandez-moi.

N'ignorez jamais ce que le compilateur vous dit !

Conseils pratiques : compilation

Vous voulez activer les warnings, et optimiser.

Certains bugs sont plus facilement visibles avec les optimisations activées (*et ils sont des vrais bugs*).

```
gcc -Wall -O2 -o destination source.c
```

Si vous ne comprenez pas un message du compilateur, demandez-moi.

N'ignorez jamais ce que le compilateur vous dit !

Conseils pratiques : compilation

Vous voulez activer les warnings, et optimiser.

Certains bugs sont plus facilement visibles avec les optimisations activées (*et ils sont des vrais bugs*).

```
gcc -Wall -O2 -o destination source.c
```

Si vous ne comprenez pas un message du compilateur, demandez-moi.

N'ignorez jamais ce que le compilateur vous dit !

Conseils pratiques : édition

Utilisez votre éditeur de texte préféré.

- Mon préféré : GNU Emacs
  - Je ne vais pas le présenter ici :
    - extrêmement puissant et flexible...
    - ...mais il faut lui consacrer du temps pour l'apprendre ;
- suggérés aux débutants : `gedit`, `kate`
  - ...médiocres, mais faciles à utiliser.

En tout cas, activez :

- numérotation de lignes et colonnes ;
- blinking des parenthèses (“*highlight matching brackets*”).

Ne sortez pas de l'éditeur à chaque compilation : gardez l'éditeur toujours ouvert, et toujours visible.

Pas d'accents dans les identifiants (dans les commentaires vous pouvez écrire correctement en français, avec les accents).

Conseils pratiques : édition

Utilisez votre éditeur de texte préféré.

- Mon préféré : GNU Emacs
  - Je ne vais pas le présenter ici :
    - extrêmement puissant et flexible...
    - ...mais il faut lui consacrer du temps pour l'apprendre ;
- suggérés aux débutants : `gedit`, `kate`
  - ...médiocres, mais faciles à utiliser.

En tout cas, activez :

- numérotation de lignes et colonnes ;
- blinking des parenthèses ("*highlight matching brackets*").

Ne sortez pas de l'éditeur à chaque compilation : gardez l'éditeur toujours ouvert, et toujours visible.

Pas d'accents dans les identifiants (dans les commentaires vous pouvez écrire correctement en français, avec les accents).

Conseils pratiques : édition

Utilisez votre éditeur de texte préféré.

- Mon préféré : GNU Emacs
  - Je ne vais pas le présenter ici :
    - extrêmement puissant et flexible...
    - ...mais il faut lui consacrer du temps pour l'apprendre ;
- suggérés aux débutants : `gedit`, `kate`
  - ...médiocres, mais faciles à utiliser.

En tout cas, activez :

- numérotation de lignes et colonnes ;
- blinking des parenthèses ("*highlight matching brackets*").

Ne sortez pas de l'éditeur à chaque compilation : gardez l'éditeur toujours ouvert, et toujours visible.

Pas d'accents dans les identifiants (dans les commentaires vous pouvez écrire correctement en français, avec les accents).

Conseils pratiques : édition

Utilisez votre éditeur de texte préféré.

- Mon préféré : GNU Emacs
  - Je ne vais pas le présenter ici :
    - extrêmement puissant et flexible...
    - ...mais il faut lui consacrer du temps pour l'apprendre ;
- suggérés aux débutants : `gedit`, `kate`
  - ...médiocres, mais faciles à utiliser.

En tout cas, activez :

- numérotation de lignes et colonnes ;
- blinking des parenthèses ("*highlight matching brackets*").

Ne sortez pas de l'éditeur à chaque compilation : gardez l'éditeur toujours ouvert, et toujours visible.

Pas d'accents dans les identifiants (dans les commentaires vous pouvez écrire correctement en français, avec les accents).

Conseils pratiques : édition

Utilisez votre éditeur de texte préféré.

- Mon préféré : GNU Emacs
  - Je ne vais pas le présenter ici :
    - extrêmement puissant et flexible...
    - ...mais il faut lui consacrer du temps pour l'apprendre ;
- suggérés aux débutants : `gedit`, `kate`
  - ...médiocres, mais faciles à utiliser.

En tout cas, activez :

- numérotation de lignes et colonnes ;
- blinking des parenthèses ("*highlight matching brackets*").

Ne sortez pas de l'éditeur à chaque compilation : gardez l'éditeur toujours ouvert, et toujours visible.

Pas d'accents dans les identifiants (dans les commentaires vous pouvez écrire correctement en français, avec les accents).

Conseils pratiques : édition

Utilisez votre éditeur de texte préféré.

- Mon préféré : GNU Emacs
  - Je ne vais pas le présenter ici :
    - extrêmement puissant et flexible...
    - ...mais il faut lui consacrer du temps pour l'apprendre ;
- suggérés aux débutants : `gedit`, `kate`
  - ...médiocres, mais faciles à utiliser.

En tout cas, activez :

- numérotation de lignes et colonnes ;
- blinking des parenthèses ("*highlight matching brackets*").

Ne sortez pas de l'éditeur à chaque compilation : gardez l'éditeur toujours ouvert, et toujours visible.

Pas d'accents dans les identifiants (dans les commentaires vous pouvez écrire correctement en français, avec les accents).

Conseils pratiques : édition

Utilisez votre éditeur de texte préféré.

- Mon préféré : GNU Emacs
  - Je ne vais pas le présenter ici :
    - extrêmement puissant et flexible...
    - ...mais il faut lui consacrer du temps pour l'apprendre ;
- suggérés aux débutants : `gedit`, `kate`
  - ...médiocres, mais faciles à utiliser.

En tout cas, activez :

- numérotation de lignes et colonnes ;
- blinking des parenthèses ("*highlight matching brackets*").

Ne sortez pas de l'éditeur à chaque compilation : gardez l'éditeur toujours ouvert, et toujours visible.

Pas d'accents dans les identifiants (dans les commentaires vous pouvez écrire correctement en français, avec les accents).

Une vision formelle des langages

Les langages de programmation, comme les langues humaines (français, chinois, langues de signes, . . .) ont une grammaire.

- Plus simple par rapport aux langues humaines ;
- plus stricte (analysable par une machine, sans ambiguïté).

Une vision formelle des langages

Les langages de programmation, comme les langues humaines (français, chinois, langues de signes, . . .) ont une grammaire.

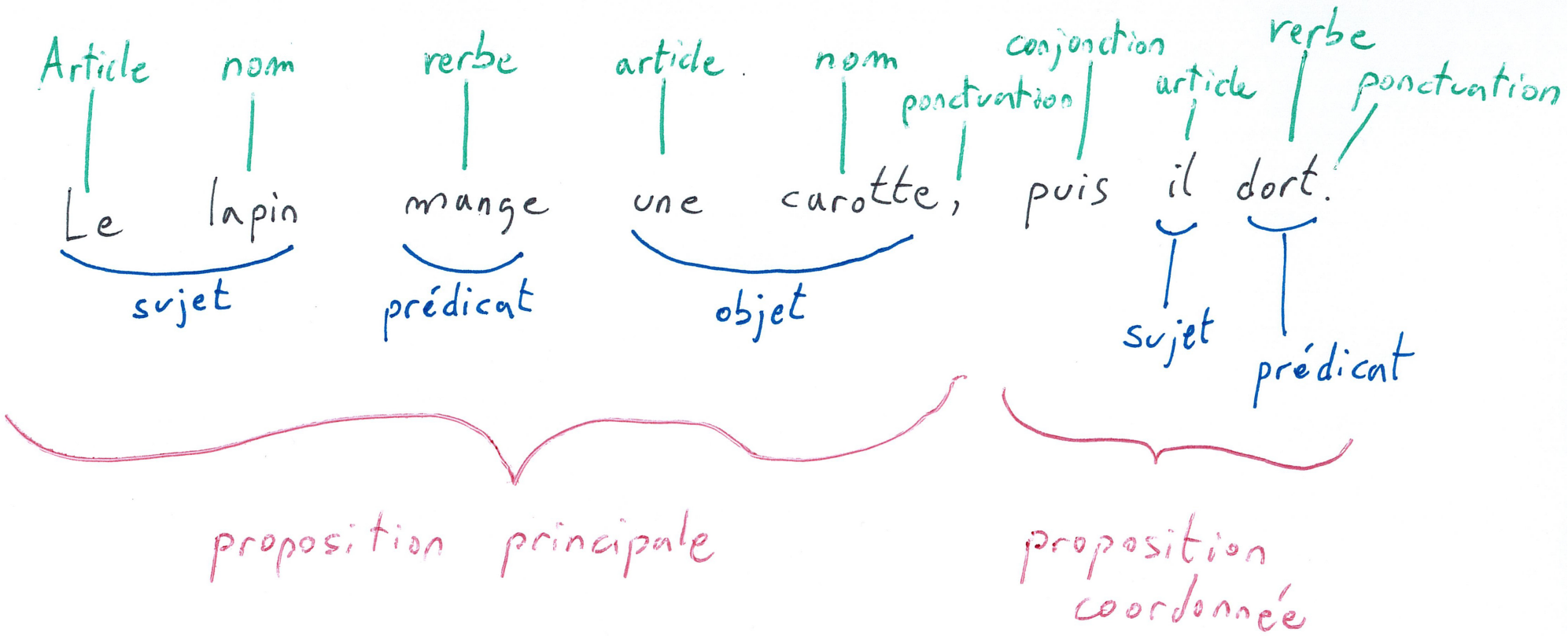
- Plus simple par rapport aux langues humaines ;
- plus stricte (analysable par une machine, sans ambiguïté).

Une vision formelle des langages

Les langages de programmation, comme les langues humaines (français, chinois, langues de signes, . . .) ont une grammaire.

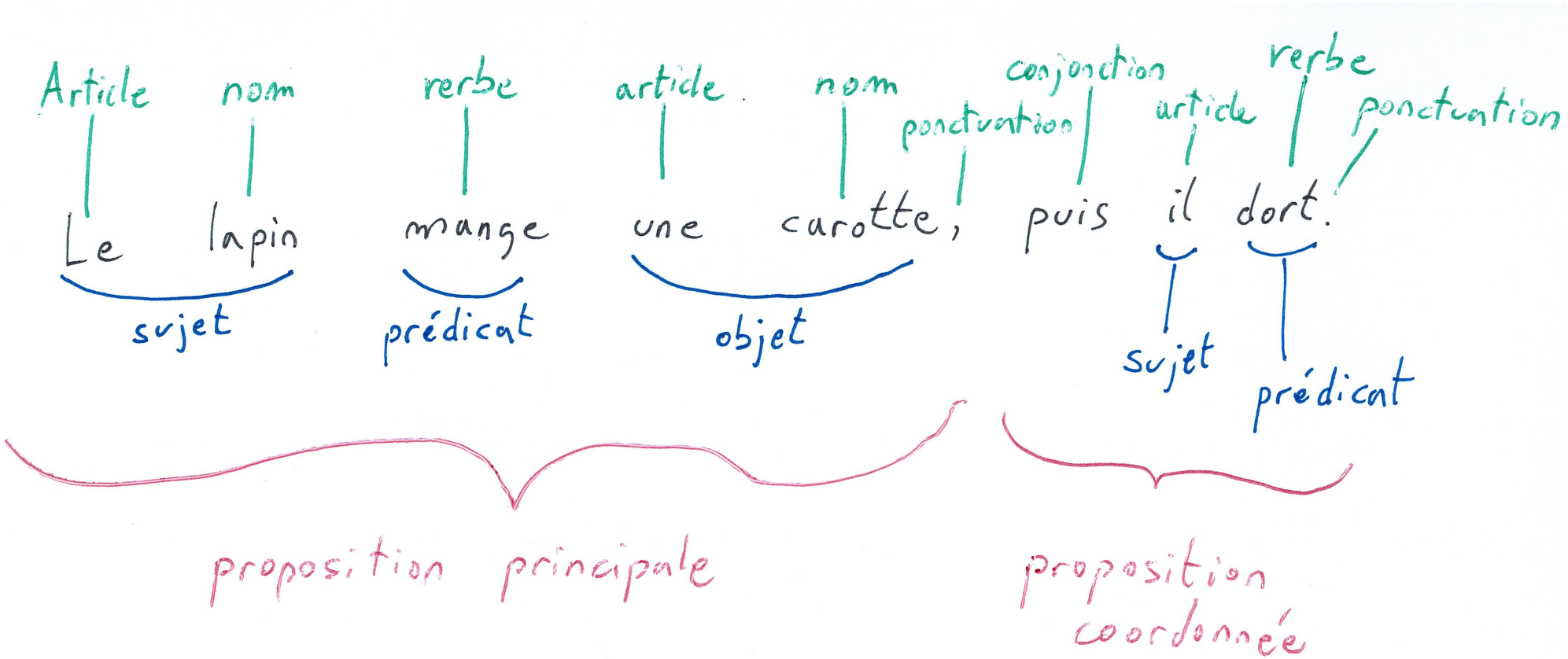
- Plus simple par rapport aux langues humaines ;
- plus stricte (analysable par une machine, sans ambiguïté).

Grammaire (française)



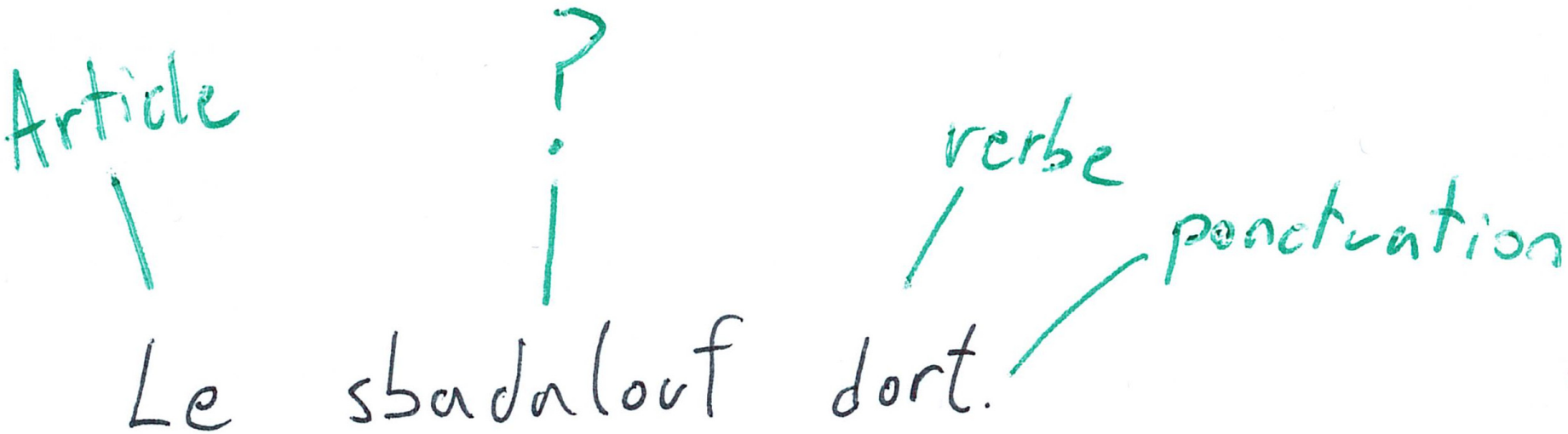
Vous devez maîtriser les catégories syntaxiques de la grammaire du langage C. [correction : bien sûr *il* est un pronom.]

Grammaire (française)



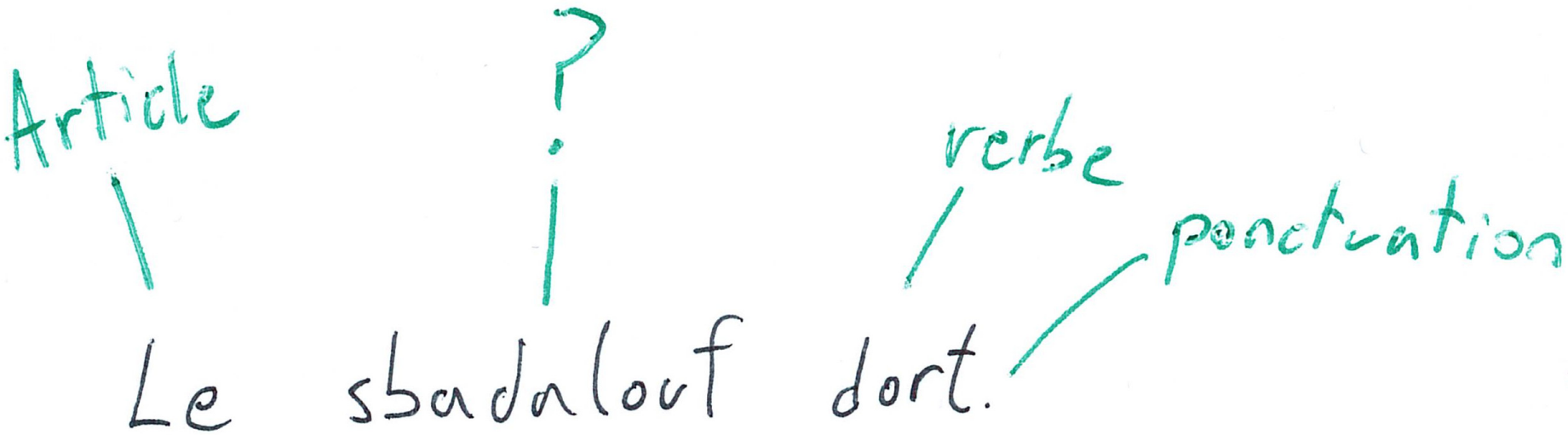
Vous devez maîtriser les catégories syntaxiques de la grammaire du langage C. [correction : bien sûr *il* est un pronom.]

Interprétez cette phrase



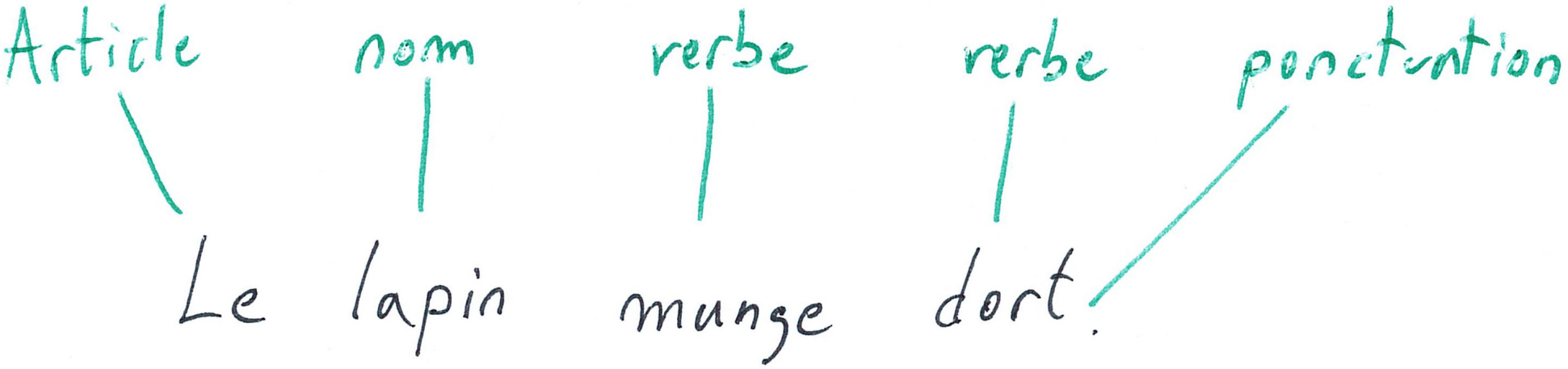
Une erreur de lexique.

Interprétez cette phrase



Une erreur de lexique.

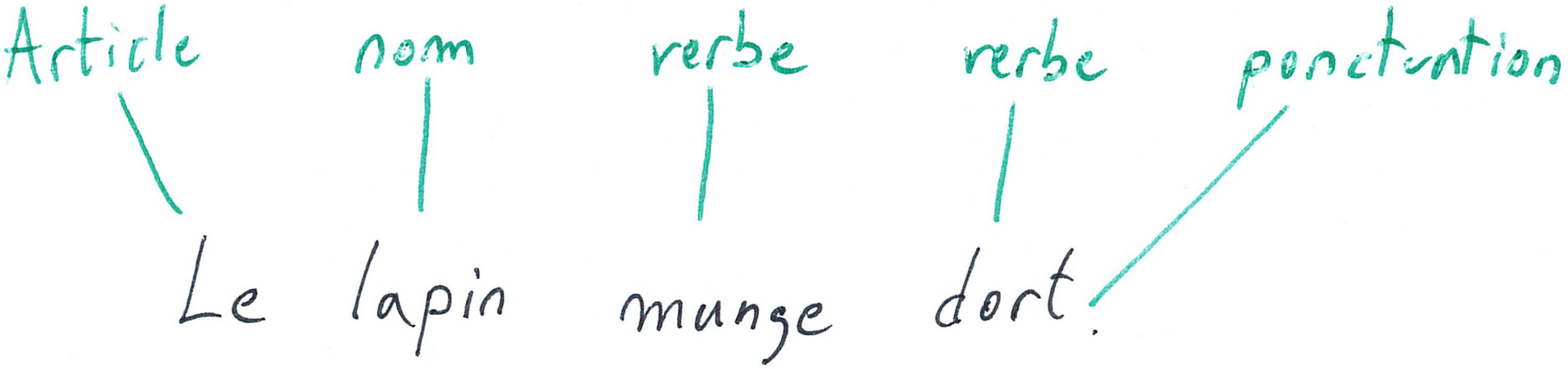
Interprétez cette autre phrase



Une erreur de syntaxe, plus subtile par rapport aux erreurs de lexique :

- tout mot pris individuellement est correct . . .
- . . . mais la phrase est mal formée : elle ne respecte pas la grammaire de la langue française.

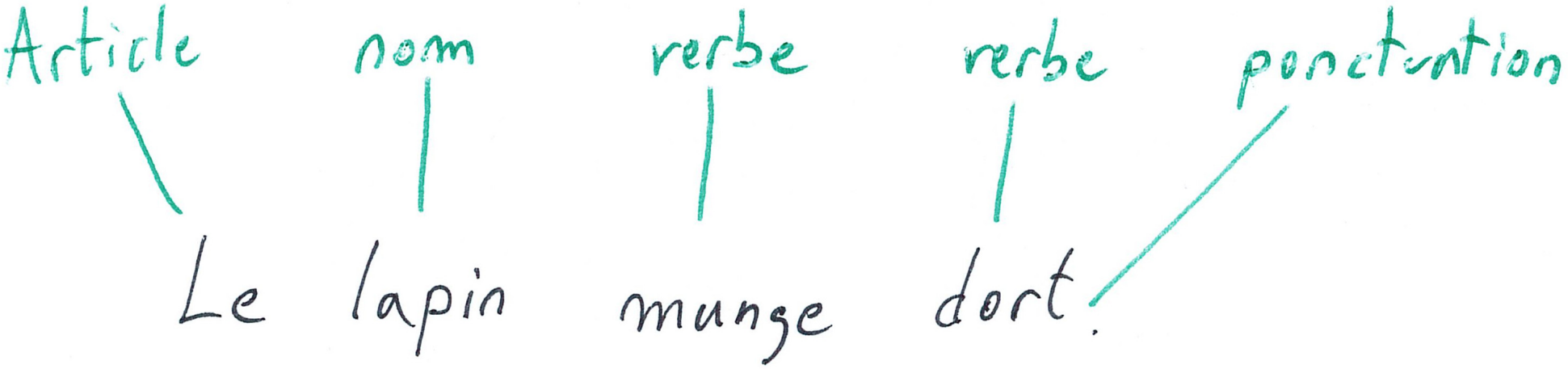
Interprétez cette autre phrase



Une erreur de syntaxe, plus subtile par rapport aux erreurs de lexique :

- tout mot pris individuellement est correct . . .
- . . . mais la phrase est mal formée : elle ne respecte pas la grammaire de la langue française.

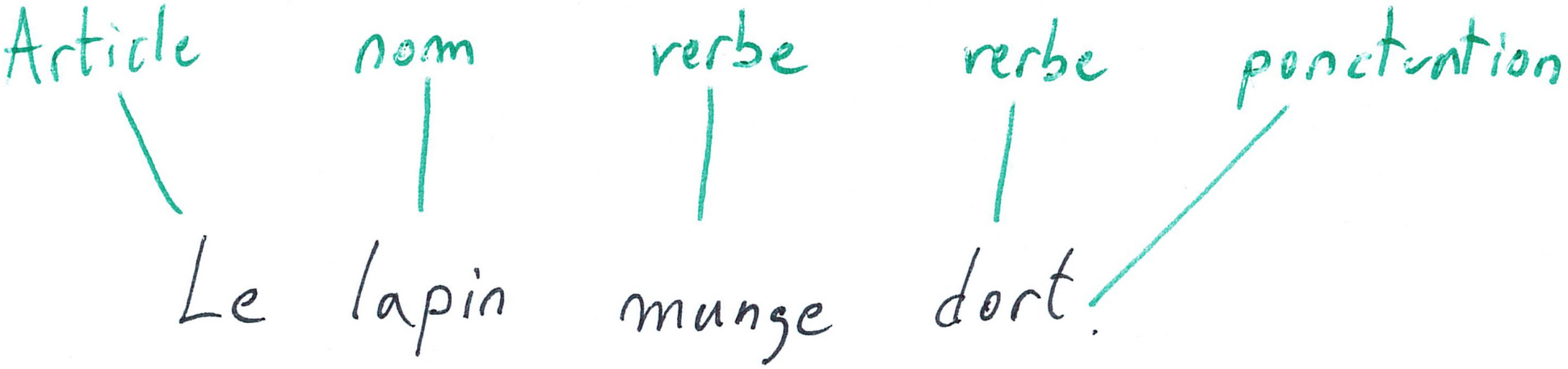
Interprétez cette autre phrase



Une erreur de syntaxe, plus subtile par rapport aux erreurs de lexique :

- tout mot pris individuellement est correct . . .
- . . . mais la phrase est mal formée : elle ne respecte pas la grammaire de la langue française.

Interprétez cette autre phrase



Une erreur de syntaxe, plus subtile par rapport aux erreurs de lexique :

- tout mot pris individuellement est correct . . .
- . . . mais la phrase est mal formée : elle ne respecte pas la grammaire de la langue française.

Lexique et syntaxe du langage C

Je ne vous donnerai pas une définition très formelle.

- lexique (la forme des mots acceptables)
- syntaxe proprement dite (règles pour combiner des mots, et règles pour combiner des phrases)

Un programme contenant des erreurs de syntaxe « ne signifie rien », et ne peut pas être exécuté. Le compilateur se refusera de l'accepter.

Lexique et syntaxe du langage C

Je ne vous donnerai pas une définition très formelle.

- lexique (la forme des mots acceptables)
- syntaxe proprement dite (règles pour combiner des mots, et règles pour combiner des phrases)

Un programme contenant des erreurs de syntaxe « ne signifie rien », et ne peut pas être exécuté. Le compilateur se refusera de l'accepter.

Lexique et syntaxe du langage C

Je ne vous donnerai pas une définition très formelle.

- lexique (la forme des mots acceptables)
- syntaxe proprement dite (règles pour combiner des mots, et règles pour combiner des phrases)

Un programme contenant des erreurs de syntaxe « ne signifie rien », et ne peut pas être exécuté. Le compilateur se refusera de l'accepter.

Lexique et syntaxe du langage C

Je ne vous donnerai pas une définition très formelle.

- lexique (la forme des mots acceptables)
- syntaxe proprement dite (règles pour combiner des mots, et règles pour combiner des phrases)

Un programme contenant des erreurs de syntaxe « ne signifie rien », et ne peut pas être exécuté. Le compilateur se refusera de l'accepter.

La sémantique d'un langage de programmation

Étant donné un programme syntaxiquement correct, on peut définir son comportement (ou *sa sémantique*). On pourrait le faire de façon mathématique, mais je vais le faire informellement dans ce cours.

```
int
successeur (int n)
{
  return n + 1;
}
```

« Cette fonction renvoie le successeur valeur de son seul paramètre (de type `int`) en tant qu'un autre `int`, sauf en cas de débordement de capacité où le comportement est indéfini ».

La sémantique d'un langage de programmation

Étant donné un programme syntaxiquement correct, on peut définir son comportement (ou *sa sémantique*). On pourrait le faire de façon mathématique, mais je vais le faire informellement dans ce cours.

```
int
successeur (int n)
{
  return n + 1;
}
```

« Cette fonction renvoie le successeur valeur de son seul paramètre (de type `int`) en tant qu'un autre `int`, sauf en cas de débordement de capacité où le comportement est indéfini ».

Erreurs de sémantique

On a aussi des erreurs de sémantique :

- « la carotte mange le lapin » est incorrect, car les carottes n'ont pas de bouche ;
- en C, la division par zéro a un comportement indéfini (“undefiend behavior”).

Les erreurs de syntaxe ne sont un problème que pour les débutants.

Vous allez bientôt passer plein de temps à chercher les erreurs sémantiques dans vos programmes.

- Votre programme est accepté par le compilateur, mais il ne fait pas ce que vous souhaitez quand vous l'exécutez...
- ... **C'est normal, même pour les experts !**
  Welcome to the world of *debugging*.

Erreurs de sémantique

On a aussi des erreurs de sémantique :

- « la carotte mange le lapin » est incorrect, car les carottes n'ont pas de bouche ;
- en C, la division par zéro a un comportement indéfini (“undefiend behavior”).

Les erreurs de syntaxe ne sont un problème que pour les débutants.

Vous allez bientôt passer plein de temps à chercher les erreurs sémantiques dans vos programmes.

- Votre programme est accepté par le compilateur, mais il ne fait pas ce que vous souhaitez quand vous l'exécutez...
- ... **C'est normal, même pour les experts !**
  Welcome to the world of *debugging*.

Erreurs de sémantique

On a aussi des erreurs de sémantique :

- « la carotte mange le lapin » est incorrect, car les carottes n'ont pas de bouche ;
- en C, la division par zéro a un comportement indéfini (“undefiend behavior”).

Les erreurs de syntaxe ne sont un problème que pour les débutants.

Vous allez bientôt passer plein de temps à chercher les erreurs sémantiques dans vos programmes.

- Votre programme est accepté par le compilateur, mais il ne fait pas ce que vous souhaitez quand vous l'exécutez…
- … **C'est normal, même pour les experts !**
  Welcome to the world of *debugging*.

Erreurs de sémantique

On a aussi des erreurs de sémantique :

- « la carotte mange le lapin » est incorrect, car les carottes n'ont pas de bouche ;
- en C, la division par zéro a un comportement indéfini (“undefiend behavior”).

Les erreurs de syntaxe ne sont un problème que pour les débutants.

Vous allez bientôt passer plein de temps à chercher les erreurs sémantiques dans vos programmes.

- Votre programme est accepté par le compilateur, mais il ne fait pas ce que vous souhaitez quand vous l'exécutez…
- … **C'est normal, même pour les experts !**
  Welcome to the world of *debugging*.

Erreurs de sémantique

On a aussi des erreurs de sémantique :

- « la carotte mange le lapin » est incorrect, car les carottes n'ont pas de bouche ;
- en C, la division par zéro a un comportement indéfini (“undefiend behavior”).

Les erreurs de syntaxe ne sont un problème que pour les débutants.

Vous allez bientôt passer plein de temps à chercher les erreurs sémantiques dans vos programmes.

- Votre programme est accepté par le compilateur, mais il ne fait pas ce que vous souhaitez quand vous l'exécutez…
- …**C'est normal, même pour les experts !**
  Welcome to the world of *debugging*.

Erreurs de sémantique

On a aussi des erreurs de sémantique :

- « la carotte mange le lapin » est incorrect, car les carottes n'ont pas de bouche ;
- en C, la division par zéro a un comportement indéfini (“undefiend behavior”).

Les erreurs de syntaxe ne sont un problème que pour les débutants.

Vous allez bientôt passer plein de temps à chercher les erreurs sémantiques dans vos programmes.

- Votre programme est accepté par le compilateur, mais il ne fait pas ce que vous souhaitez quand vous l'exécutez...
- . . . C'est normal, même pour les experts !
  Welcome to the world of *debugging*.

Erreurs de sémantique

On a aussi des erreurs de sémantique :

- « la carotte mange le lapin » est incorrect, car les carottes n'ont pas de bouche ;
- en C, la division par zéro a un comportement indéfini (“undefiend behavior”).

Les erreurs de syntaxe ne sont un problème que pour les débutants.

Vous allez bientôt passer plein de temps à chercher les erreurs sémantiques dans vos programmes.

- Votre programme est accepté par le compilateur, mais il ne fait pas ce que vous souhaitez quand vous l'exécutez...
- ... **C'est normal, même pour les experts !**
  Welcome to the world of *debugging*.

Types

C est un langage typé statiquement, mais faiblement.

Aujourd'hui on travaillera juste avec des types fondamentaux prédéfinis :

- `int` : nombre entier ;
- `float` : nombre *flottant* ;
- `char` : caractère (un seul caractère [simplification !]) ;
- `bool` : booléen (vrai ou faux [simplification !]) ;
- `char *` : chaîne de caractères ([simplification !]) ;

Commentaires

Les commentaires sont des morceaux d'un programme qui sont effectivement *ignorés par le compilateur*. Ils ne servent que *aux lecteurs humaines* de votre code.

Utilisez-les pour des explications, ou pour désactiver temporairement du code.

On commence un commentaire par `/*`, et on le termine par `*/`.

Exemple :

```
/* Bonjour, je suis un commentaire. */
```

C'est interdit d'utiliser `*/` dans le texte d'un commentaire.

Lexique C : constantes entières (littéraux entiers)

Entiers :

- Écrits en base 10, si le premier chiffre n'est pas 0 : `42`, `7` ;
- Écrits en base 16, avec le préfixe `0x` : `0xf` est 15 ; `0x31f` est $15 \cdot 16^0 + 1 \cdot 16^1 + 3 \cdot 16^2 = 15 + 16 + 259 = 799$;
- Écrits en base 8, avec le préfixe 0 : `010` est $0 \cdot 8^0 + 1 \cdot 8^1 = 8$; différent de `10` : en décimal, $0 \cdot 10^0 + 1 \cdot 10^1 = 10$.
- `0` est zéro.

Signe à gauche, optionnel : `+` ou `-`.

- `+42` et `42` sont deux façons d'écrire le même nombre entier, 42 en décimal—le signe `+` est optionnel ;
- `-42` (ou `-052`, ou `-0x2a`) est un nombre entier négatif.

Les littéraux entiers ne peuvent pas avoir un point (séparateur décimal) : `2.0` n'est pas un entier en C.

Lexique C : constantes entières (littéraux entiers)

Entiers :

- Écrits en base 10, si le premier chiffre n'est pas 0 : `42`, `7` ;
- Écrits en base 16, avec le préfixe `0x` : `0xf` est 15 ; `0x31f` est $15 \cdot 16^0 + 1 \cdot 16^1 + 3 \cdot 16^2 = 15 + 16 + 259 = 799$;
- Écrits en base 8, avec le préfixe 0 : `010` est $0 \cdot 8^0 + 1 \cdot 8^1 = 8$; différent de `10` : en décimal, $0 \cdot 10^0 + 1 \cdot 10^1 = 10$.
- `0` est zéro.

Signe à gauche, optionnel : `+` ou `-`.

- `+42` et `42` sont deux façons d'écrire le même nombre entier, 42 en décimal—le signe `+` est optionnel ;
- `-42` (ou `-052`, ou `-0x2a`) est un nombre entier négatif.

Les littéraux entiers ne peuvent pas avoir un point (séparateur décimal) : `2.0` n'est pas un entier en C.

Lexique C : constantes entières (littéraux entiers)

Entiers :

- Écrits en base 10, si le premier chiffre n'est pas 0 : `42`, `7` ;
- Écrits en base 16, avec le préfixe `0x` : `0xf` est 15 ; `0x31f` est $15 \cdot 16^0 + 1 \cdot 16^1 + 3 \cdot 16^2 = 15 + 16 + 259 = 799$;
- Écrits en base 8, avec le préfixe 0 : `010` est $0 \cdot 8^0 + 1 \cdot 8^1 = 8$; différent de `10` : en décimal, $0 \cdot 10^0 + 1 \cdot 10^1 = 10$.
- `0` est zéro.

Signe à gauche, optionnel : `+` ou `-`.

- `+42` et `42` sont deux façons d'écrire le même nombre entier, 42 en décimal—le signe `+` est optionnel ;
- `-42` (ou `-052`, ou `-0x2a`) est un nombre entier négatif.

Les littéraux entiers ne peuvent pas avoir un point (séparateur décimal) : `2.0` n'est pas un entier en C.

Lexique C : constantes flottantes (littéraux flottants)

Écrites en base 10, avec `.` (point) comme séparateur décimal :

- `2.5` : deux et mi ;
- `+2.5` : deux et mi ;
- `-2.5` : moins deux et mi ;
- `0.5` : zéro virgule cinq ;
- `5.0` : cinq virgule zéro : cinq en tant qu'un flottant ;
- `3.14159265358979` : une approximation de π.

On peut avoir un zéro "implicite" à gauche ou à droite du point :

- `.5` : zéro virgule cinq ;
- `5.` : cinq virgule zéro : cinq en tant qu'un flottant.

C supporte la notation scientifique aussi :

- `5.3e-7` est $5.3 \cdot 10^{-7}$.

Lexique C : constantes flottantes (littéraux flottants)

Écrites en base 10, avec `.` (point) comme séparateur décimal :

- `2.5` : deux et mi ;
- `+2.5` : deux et mi ;
- `-2.5` : moins deux et mi ;
- `0.5` : zéro virgule cinq ;
- `5.0` : cinq virgule zéro : cinq en tant qu'un flottant ;
- `3.14159265358979` : une approximation de π.

On peut avoir un zéro "implicite" à gauche ou à droite du point :

- `.5` : zéro virgule cinq ;
- `5.` : cinq virgule zéro : cinq en tant qu'un flottant.

C supporte la notation scientifique aussi :

- `5.3e-7` est $5.3 \cdot 10^{-7}$.

Lexique C : constantes flottantes (littéraux flottants)

Écrites en base 10, avec `.` (point) comme séparateur décimal :

- `2.5` : deux et mi ;
- `+2.5` : deux et mi ;
- `-2.5` : moins deux et mi ;
- `0.5` : zéro virgule cinq ;
- `5.0` : cinq virgule zéro : cinq en tant qu'un flottant ;
- `3.14159265358979` : une approximation de π.

On peut avoir un zéro "implicite" à gauche ou à droite du point :

- `.5` : zéro virgule cinq ;
- `5.` : cinq virgule zéro : cinq en tant qu'un flottant.

C supporte la notation scientifique aussi :

- `5.3e-7` est $5.3 \cdot 10^{-7}$.

Lexique C : caractères littéraux

Caractères :

- un seul caractère, entre deux apostrophes (`'`) : par exemple `'a'`, `'b'`, `'B'`, `' '` (le caractère « espace »).

Utilisez l'*apostrophe symétrique* ! (touche 4 sur le clavier français)

Certains caractères ont une syntaxe spéciale (escaping) :

- retour à la ligne `'\n'` ("*newline*" en anglais) ;
- apostrophe `'\''` ;
- backslash `'\\'` ;
- ...

Lexique C : chaînes de caractères littérales

Comme les caractères, mais délimitées par " à gauche et à droite à la place de '. Bien sûr vous pouvez avoir *plusieurs caractères* entre " et ".

Guillemets symétriques : touche 3 sur le clavier français.

```
"je suis une chaîne de caractères"
```

Nouvelle séquence d'escaping : écrivez les deux caractères `\"` pour avoir *un seul* caractère " dans une chaîne. Vous avez les autres séquences que vous connaissez déjà aussi, comme `\n`.

[Simplification ! Les chaînes ne sont pas un « vrai » type en C]

Lexique C : identifiants

(Les *identifiants* sont des noms (de variables, de fonctions), . . .)

Une séquence arbitraire de lettres (ASCII), chiffres et _ (“*underscore*”), ne commençant pas par une chiffre.

Exemples : `foo`, `bar`, `x`, `i42`, `a_b_c`.

Lexique C : identifiants

(Les *identifiants* sont des noms (de variables, de fonctions), . . .)

Une séquence arbitraire de lettres (ASCII), chiffres et _ (“*underscore*”), ne commençant pas par une chiffre.

Exemples : `foo`, `bar`, `x`, `i42`, `a_b_c`.

Le squelette d'un programme est encore « magique » . . .

Vous allez comprendre tout détail plus tard.

Maintenant utilisez un des exemples sur la page web du cours, et modifiez juste le texte entre { et }.

Expressions

Une expression est une phrase du langage calculant un résultat.
Une expression peut être élémentaire, ou composée par des autres sous-expressions.

Exemple : la somme de 4 et de 5 : `4 + 5`.

Commandes

Une commande est un ordre que vous donnez à la machine : une action à exécuter.

Exemple : change la valeur de la variable a, en la mettant à 7 :

```
a = 7;
```

Les commandes sont terminés par `;`, ou sont des blocs.

Commandes

Déclarations de variables.
Les variables sont déclarées dans un bloc, précédées par leur type.
...

Demo

[Demo]

Bibliography I


Saiu, L. (2018). La page web de mes cours.
http://ageinghacker.net/teaching
La page web officielle du cours contient des pointeurs à des ressources web, et une copie des mes transparents.


Stallman, R. and Rothwell, T. (2019). *GNU C Intro and Reference*. Free Software Foundation.
(Manuscrit non encore officiellement publié. Demandez-moi un exemplaire si vous êtes intéressés.).