Complexité

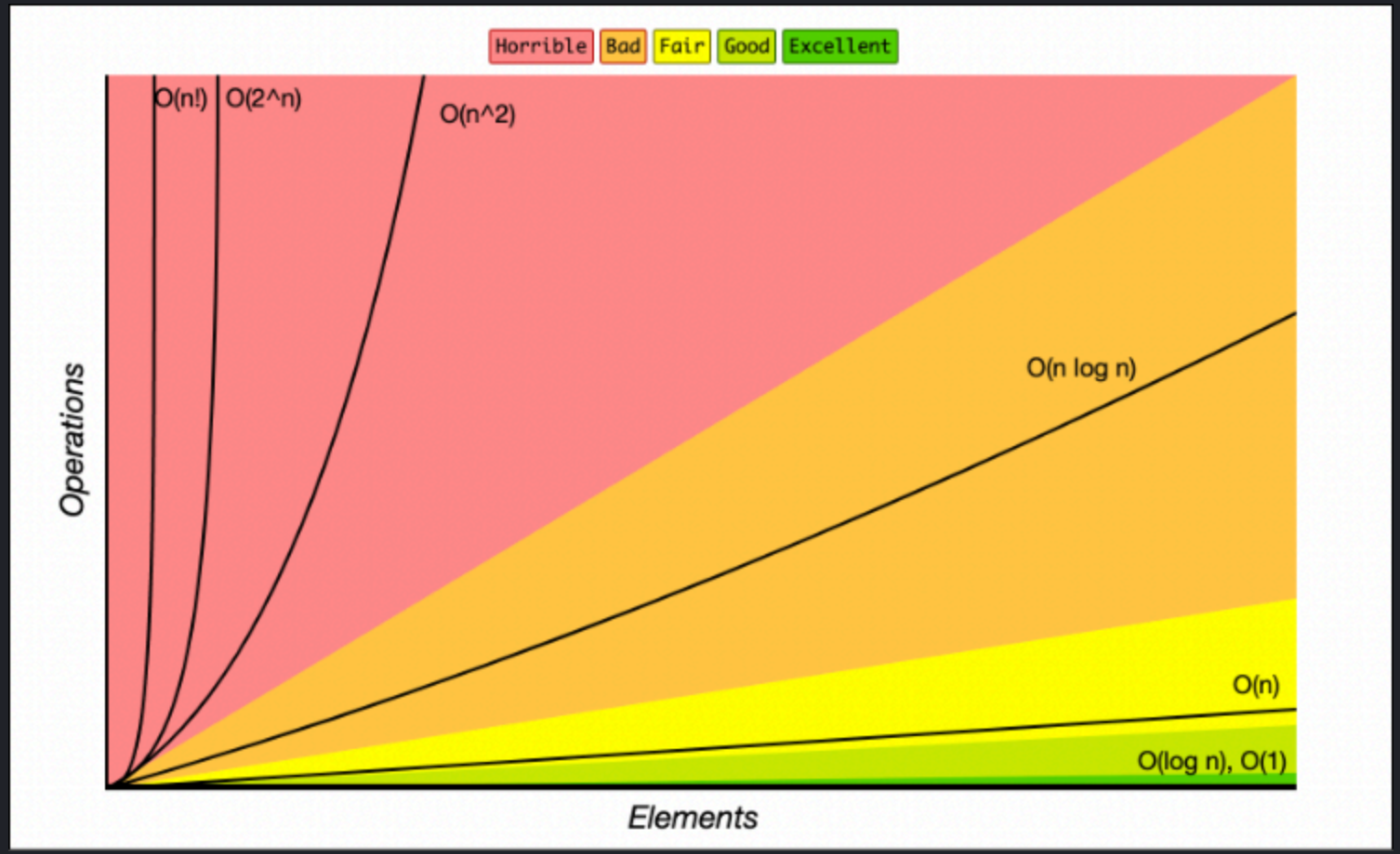
Complexité

- La complexité c'est l'évaluation de l'empreinte mémoire ou du temps d'exécution d'un algorithme.
- On utilise la notation "Big O" qui évalue par ordre de grandeur la complexité d'un algorithme.

Complexité : notation Big O

- Les valeurs remarquables :
  - $O(1)$
  - $O(\log n)$
  - $O(n)$
  - $O(n \log n)$
  - $O(n^2)$
  - $O(2^n)$
  - $O(n!)$

La complexité : notation "Big O"



Exemple

```
PROCEDURE print_until(UINT n) THEN
    FOR INT i = 0; i <= n; i++ THEN
        PRINT(i)
    ENDIF
ENDPROCEDURE
```

Gestion de la mémoire

- Le programme demande de stocker une donnée à l'OS qui lui réserve un emplacement mémoire.
- Une variable est en réalité le numéro de l'emplacement mémoire réservé.

- $\Rightarrow$ Quels sont les espaces mémoires dont disposent les programmes ?

Gestion de la mémoire : Stack et Heap

- Il y a deux espaces mémoires:
  - Le stack : il s'agit d'un espace mémoire exclusivement réservé pour le programme.
  - Le heap : il s'agit d'un espace mémoire commun aux différents programmes.

Gestion de la mémoire : Stack

- Le stack permet de stocker plus rapidement des valeurs.
- Le stack stocke les appels de fonctions et de procédures.
- Son espace est limité il faut donc faire attention.
- S'il n'y a plus de place (trop d'appels de fonctions par exemple), on appelle cela un "Stack overflow".

Gestion de la mémoire : Heap

- Le heap est plus lent pour stocker des valeurs mais il est plus grand.
- Ce sont généralement les valeurs volumineuses qui y sont stockées.

Créer nos propres types

Combiner les valeurs

- Il est possible de combiner des valeurs afin de créer de nouveaux types.
- Un objet ou une entité peut être décomposé en types atomiques.

Combiner les valeurs

- Il existe plusieurs facon, en fonction des langages, de combiner les types :
  - Les structures (langage C)
  - Les classes (programmation orientée objet)

Combiner les valeurs

- Nous utiliserons les structures.
- Le nom d'une structure est composé de lettres et du caractère '_'. Il respecte la convention de nommage "Pascal case" ou "Snake case".
- Les attributs d'une structure sont définis comme des paramètres de fonction.

Combiner les valeurs

```
struct person {
    int age;
    char[] name;
};
```

```
class Person {
    int age;
    string name;
}
```

Combiner les valeurs

- Pour instancier une structure il faut lister les valeurs entre accolades dans l'ordre des attributs.
- Pour accéder aux attributs d'une structure on utilise le caractère '.'.

Combiner les valeurs

```
STRUCTURE Person
    INT age
    STRING name
ENDSTRUCTURE

Person person = { 10, "Tom" }
PRINT()
```

Les énumérations

Les énumérations

- Une énumération est un moyen de lister différentes options.
- Chaque option possède une valeur entière différente.
- L'objectif est de faciliter la lecture du code.
- Le nom d'une énumération est composé de lettres et du symbole '_'. Il respecte la convention "Pascal case" ou "Snake case". Les options d'une énumération sont nommées avec la convention "Screaming snake case".

Les énumérations

```
PROCEDURE move(INT direction) THEN
    SWITCH move THEN
        CASE 0 THEN PRINT("Move north") BREAK
        CASE 1 THEN PRINT("Move east") BREAK
        CASE 2 THEN PRINT("Move south") BREAK
        CASE 3 THEN PRINT("Move west") BREAK
    ENDSWITCH
ENDPROCEDURE

// Pas très explicite
move(0)
```

Les énumérations

```
ENUM Direction
    NORTH
    EAST
    SOUTH
    WEST
ENDENUM

PROCEDURE move(Direction direction) THEN
    SWITCH move THEN
        CASE NORTH THEN PRINT("Move north") BREAK
        CASE EAST THEN PRINT("Move east") BREAK
        CASE SOUTH THEN PRINT("Move south") BREAK
        CASE WEST THEN PRINT("Move west") BREAK
    ENDSWITCH
ENDPROCEDURE

move(NORTH)
```

Une variable de plusieurs types ?

Une variable de plusieurs types ?

- Pour simplifier et rendre générique le code d'une application on peut définir une valeur ayant différent types.
- L'idée est dire qu'une variable peut représenter différentes choses et qu'il faut bien distinguer les différents cas.

Une variable de plusieurs types ?

- On peut y parvenir de différentes façons en fonction des langages de programmation :
  - Les unions (langage C).
  - L'héritage, les classes abstraites (programmation orientée objet).
  - Les types énumérés (programmation fonctionnelle).
  - Les traits, interfaces et templates (programmation fonctionnelle).

Une variable de plusieurs types ?

- Nous utiliserons les unions.
- Le nom d'une union est composé de lettres et du symbole '_'. Il respecte la convention de nommage "Pascal case" ou "Snake case".
- Les variantes d'une union sont définies comme les attributs d'une structure.
- Pour instancier une union on place la valeur entre accolades.

Une variable de plusieurs types ?

- Pour accéder à une variante on utilise le symbole '.'.
- Pour savoir quelle variante est contenue dans l'union on utilise la structure `SWITCH`.

Une variable de plusieurs types ?

```
UNION MyUnion
    INT an_int;
    BOOL a_boolean;
    CHAR a_char;
ENDUNION

MyUnion my_union = { 10 }

SWITCH my_union THEN
    CASE INT int_value THEN PRINT("Its an integer : " + int_value.to_string()) BREAK
    CASE BOOL bool_value THEN PRINT("Its a boolean : " + bool_value. to_string()) BREAK
    CASE CHAR char_value THEN PRINT("Its a char : " + char_value.to_string()) BREAK
ENDSWITCH
```

Une variable de plusieurs types ?

- En C++

```
union MyUnion {
    int an_integer;
    bool a_boolean;
    char a_char;
}
```

Une variable de plusieurs types ?

- En C#

```
abstract class Value {
    abstract string to_string();
}

class Integer : Value {
    int val;
    string to_string() { return val.to_string(); }
}

class Char : Value {
    char val;
    string to_string() { return val.to_string(); }
}
```

Une variable de plusieurs types ?

- En OCaml

```
type MyValue =
| an_integer of int
| a_boolean of bool
| a_char of char
```

Une variable de plusieurs types ?

- En Rust 1/2

```
enum MyValue {
    AnInteger(i64),
    ABoolean(bool),
    AChar(char)
}
```

Une variable de plusieurs types ?

- En Rust 2/2

```
trait ToString {
    pub fn (&self) -> String;
}

struct AnInt(i64);
struct ABool(bool);
struct AChar(char);

impl ToString for AnInt { ... };
impl ToString for ABool { ... };
impl ToString for AChar { ... };
```

Les pointeurs

Les pointeurs

- Les pointeurs permettent de stocker une addresse dans la mémoire.
- Ils sont utilisés pour ne pas avoir à réserver d'espaces mémoires continus.
- Ils sont également utilisés pour modifier le contenu des structures et des unions.

Structures de données

Structure de données : définition

- Une structure de données permet d'organiser des informations.
- Elles permettent de représenter des relations, des priorités, etc…

Structure de données : Les listes

- C'est un tableau qui peut changer de taille.

Structure de données : Les files

- Même principe qu'une liste.
- Enregistre les éléments par ordre d'arrivée (FIFO).

Structure de données : Les piles

- Même principe qu'une liste.
- Enregistre les éléments par ordre inverse d'arrivée (FILO).

Structure de données : Les ensembles

- Il s'agit d'une liste où les valeurs n'apparaissent qu'une seule fois.
- On ne peut pas accéder aux éléments via leurs index.

Structure de données : Les dictionnaires

- Associent des clés et des valeurs.
- Il ne peut pas y avoir deux fois la même clé.
- Il peut y avoir deux fois la même valeur.

Structure de données : Les tampons

- Ils servent à temporairement stocker des valeurs.
- Ils servent de liaison entre deux fonctionnalités.

Structure de données : Les graphes

- Représentent des relations entre des objets.

Structure de données : Les arbres

- C'est un type de graphe.
- Il y a un lien de père à fils entre les données.

Structure de données

Structure	Accès	Recherche	Insertion	Supr.
Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Queue	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Pile	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Liste C.	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Liste 2C.	$O(n)$	$O(n)$	$O(1)$	$O(1)$
Dict.	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Structure de données

Structure	Accès	Recherche	Insertion	Supr.
Arbre B.R.	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Arbre B.E.R.	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

Le langage C

Mon premier programme

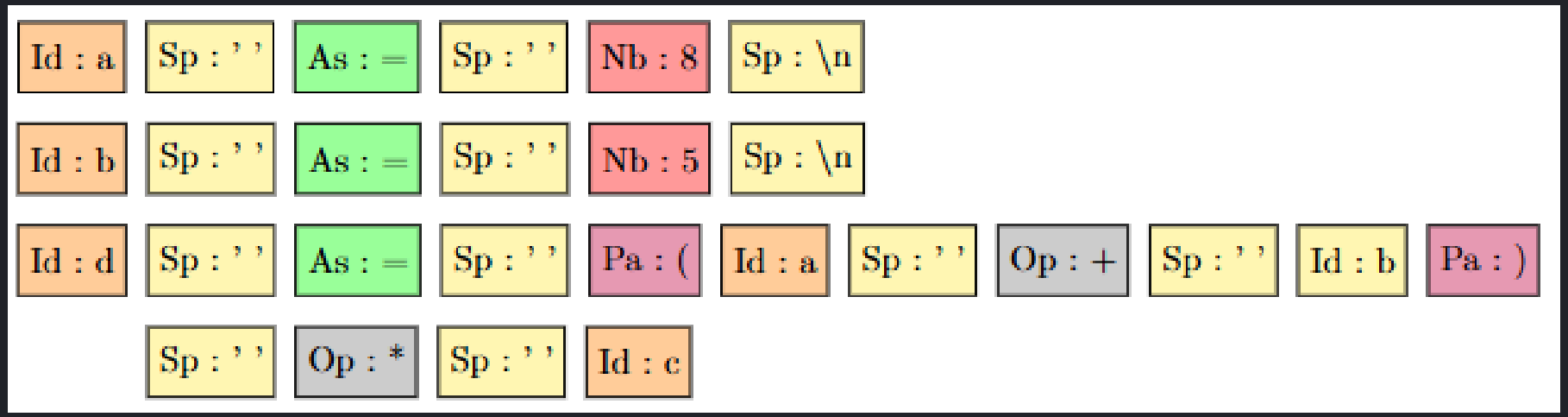
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    printf("Hello, world");
    return EXIT_SUCCESS;
}
```

La compilation

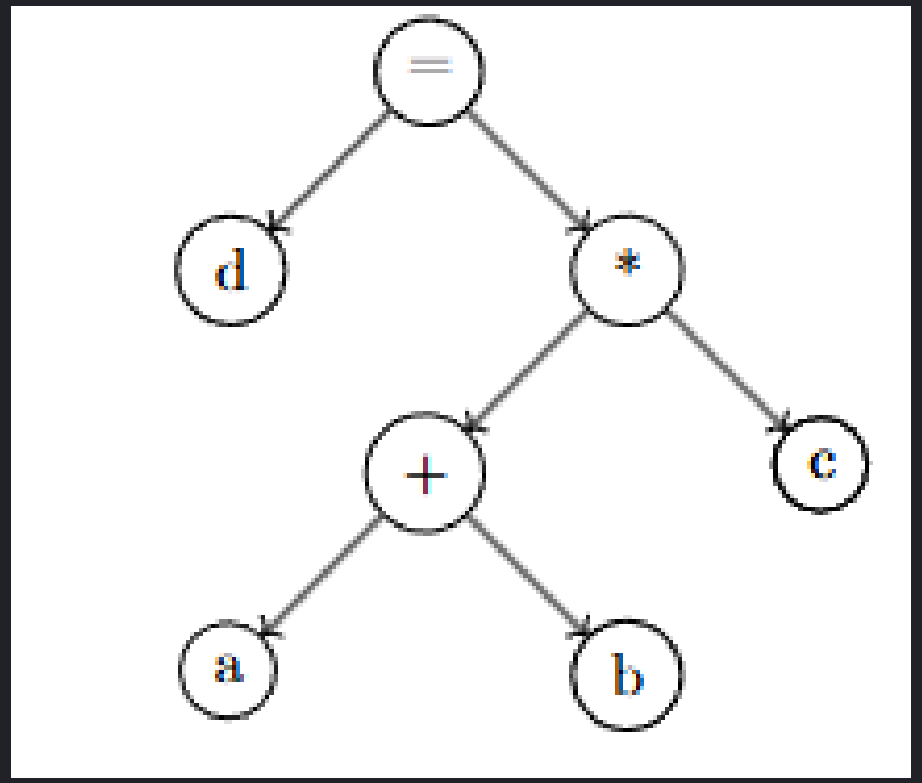
- La compilation transforme le code en fichier exécutable. Il y a plusieurs étapes:
  - La traduction du code:
    - L’analyse lexicale
    - L’analyse syntaxique
    - L’analyse s ́emantique
  - L’assemblage du programme

La compilation : l'analyse lexicale

```
a = 8
b = 5
d = (a + b) * c
```

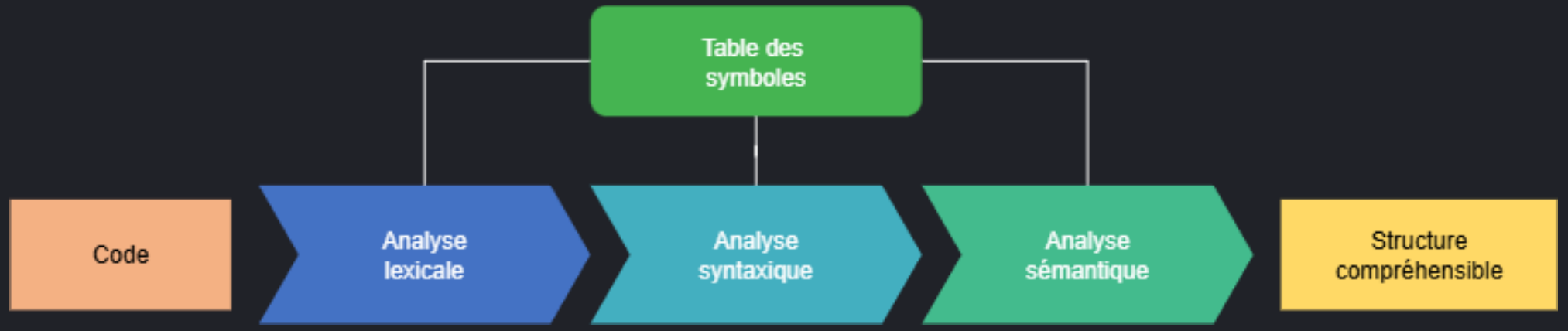


La compilation : l'analyse syntaxique

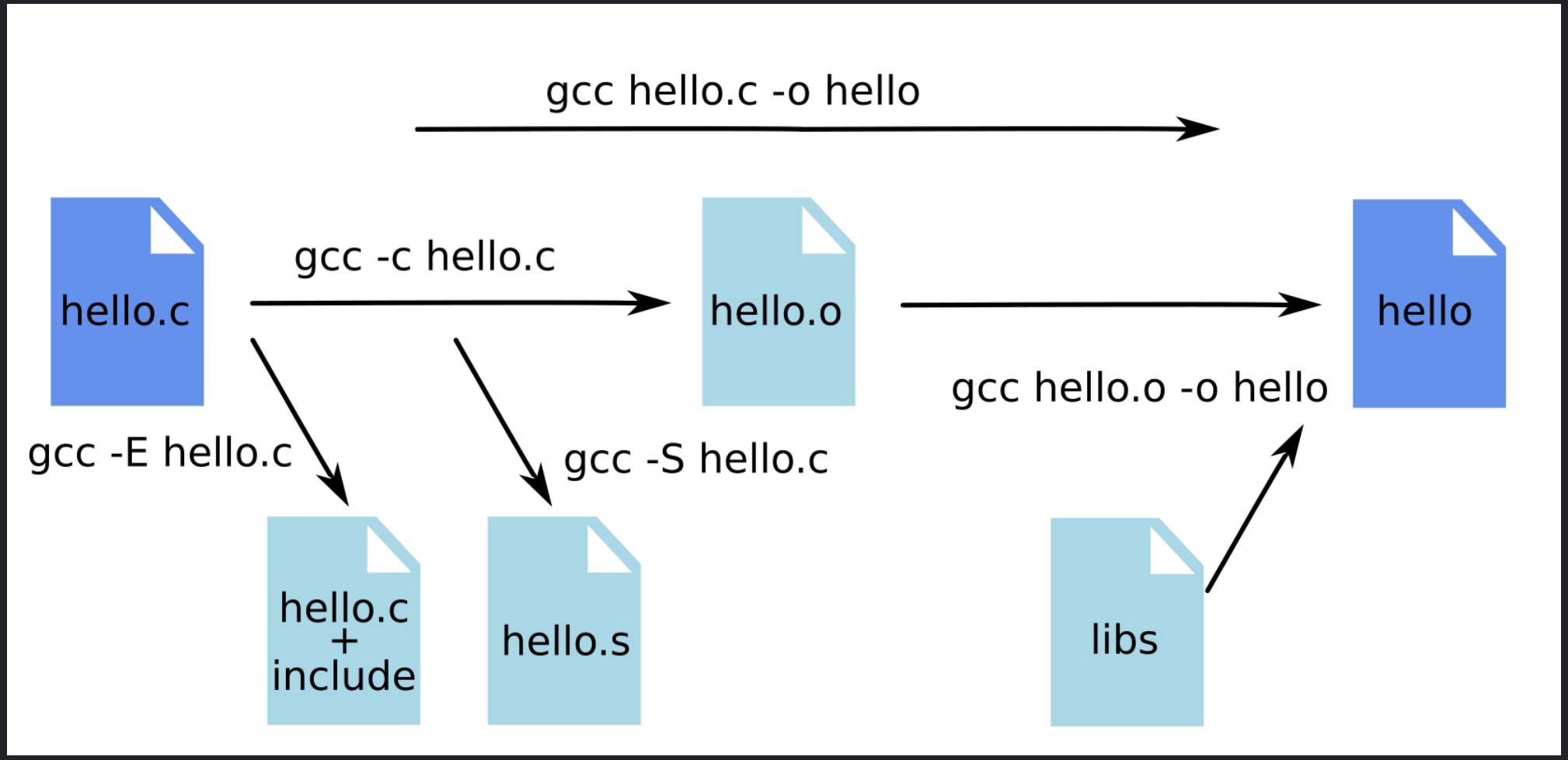


La compilation : l'analyse syntaxique

- Analyse du sens



La compilation : l'assemblage



Introduction à GCC

- `-std=c99` : version de C
- `-pedantic` : Norme ISO du C
- `-Wall` : Affiche les warnings
- `-Wextra` : Affiche encore plus de warnings
- `-Werror` : transforme les warnings en erreurs
- `-Wvla` : Interdit les tableaux à tailles variables
- `-fstack-protector` : active la protection de la pile contre les d ́ebordements

Introduction à GCC : exemple de compilation

Makefile

Langage C

Langage C : les commentaires

```
// Un commentaire sur une ligne

/*
Un commentaire
sur plusieurs
lignes
*/
```

Langage C : les types

Nom	Taille (en bits)
char / unsigned char	8
short / unsigned short	16
int / unsigned int	32
long / unsigned long	32 ou 64
float	32
double	64
long double	128

Langace C : les variables

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {
    int my_int = 10;
    char my_char;
    long double my_double = 1.12;
    return EXIT_SUCCESS;
}
```

Langace C : les constantes

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {
    const int MY_CONST = 10;
    return EXIT_SUCCESS;
}
```

Langace C : alias de type

```c
#include <stdlib.h>
#include <stdlio.h>

typedef unsigned char color_param_t;

int main(int argc, char **argv) {
    unsigned char color_red = 255;
    color_param color_blue = 255;
    return EXIT_SUCCESS;
}
```

Langace C : les tableaux

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {
    int my_array[100];
    printf("La premiere case est : %d\n", my_array[0]);

    int my_array2[] = {25, 50, 75, 100};
    printf("La premiere case est : %d\n", my_array2[0]);

    return EXIT_SUCCESS;
}
```

Langace C : les `if` - `else`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {
    int a = 11;
    int b = 32;

    if (a > b) {
        printf("A is greater than b");
    } else if (a < n) {
        printf("A is lower than b");
    } else {
        printf("A is equal to b");
    }
    return EXIT_SUCCESS;
}
```

Langace C : les `if` - `else`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {
    int a = 11;
    int b = 32;

    if (a > b)
        printf("A is greater than b");
    else if (a < n)
        printf("A is lower than b");
    else
        printf("A is equal to b");

return EXIT_SUCCESS;
}
```

Langage C : les `switch`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char** argv) {
int dice = 6;

switch (dice) {
case 1:
    printf("The dice roll is 1");
    break;
case 2:
    printf("The dice roll is 2");
    break;
case 3:
    printf("The dice roll is 3");
    break;
case 4:
    printf("The dice roll is 4");
    break;
case 5:
    printf("The dice roll is 5");
    break;
case 6:
    printf("The dice roll is 6");
    break;
default:
    printf("This is not a valid dice roll");
    break;
}
return EXIT_SUCCESS;
}
```

Langage C : boucle `while`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {
    int n = 0;
    while (n < 10) {
        if (n == 3)
            continue;
        if (n == 8)
            break;
        n++;
    }
    return EXIT_SUCCESS;
}
```

Langage C : boucle `while`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {
    int n = 0;

    while (n < 10)
        n++;

    return EXIT_SUCCESS;
}
```

Langage C : boucle `do while`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {
    int n = 0;
    do {
        if (n == 3) {
            continue;
        }
        if (n == 8) {
            break;
        }
        n++;
    } while (n < 10);

    return EXIT_SUCCESS;
}
```

Langage C : boucle `for`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char **argv) {

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

    return EXIT_SUCCESS;
}
```

Langage C : boucle `for`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char** argv) {

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

    return EXIT_SUCCESS;
}
```

Langage C : boucle `for`

```c
#include <stdlib.h>
#include <stdlio.h>

int main(int argc, char** argv) {

    for (int _ = 0; _ < 10; _++)
        printf("Loop");

    return EXIT_SUCCESS;
}
```

Langage C : fonctions et procédures

```c
#include <stdlio.h>

void say_hello() {
    printf("Hello\n");
}

void say_hello_n_times(int n) {
    for (int _ = 0; _ < 10; _++)
        printf("Hello\n");
}

int max(int a, int b) {
    if (a < b)
        return b;
    return a;
}
```

Langage C : fonction printf

- Elle utilise des indicateurs pour formatter le texte.
- Elle accepte un nombre variable de paramètres.

Langage C : la fonction main

- Retourne un entier qui indique l'état de fin du programme.
- Accepte des arguments via la ligne de commande.