Introduction au langage C[1]

Salem BENFERHAT

Centre de Recherche en Informatique de Lens (CRIL-CNRS)
email : benferhat@cril.fr

[1] Version préliminaire du cours. Tout retour sur la forme comme sur le fond est le bienvenu.

Informations pratiques

Informations pratiques

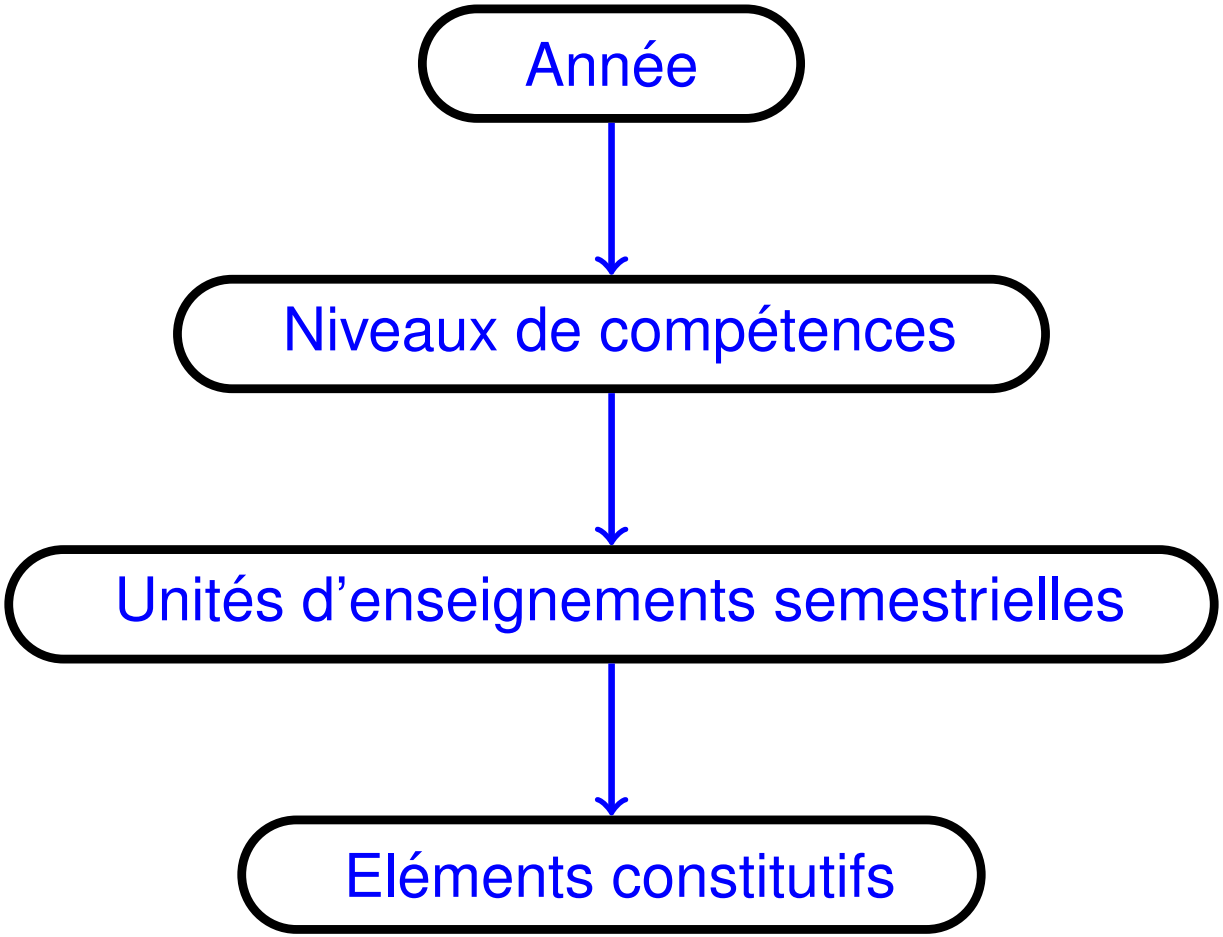
- Cours sur 12 semaines (12x1,5h).
- TD/TP sur 12 semaines (12x2h).
- Cours/TD/TP (Lundi et mardi).
  - **N'oubliez pas de récupérer vos identifiants pour les TP!**
- Un cours de "C avancé" sera donné en L3.

Informations pratiques

- Contrats pédagogiques
- Modalités de contrôle : contrôles continus
  - 3 notes : deux CC (chacun avec coefficient 1) et un CC terminal (coefficient 2)
- Merci de consulter régulièrement ADE car des changements d'emploi du temps sont probables.
- Contact : **benferhat@cril.fr**

M3C:
Modalités du Contrôle des Connaissances et des Compétences

M3C : organisation hiérarchique



Validation d'une année

Les 5 compétences en Licence Informatique :

C1 : Élaborer une modélisation numérique d'un problème et de ses données

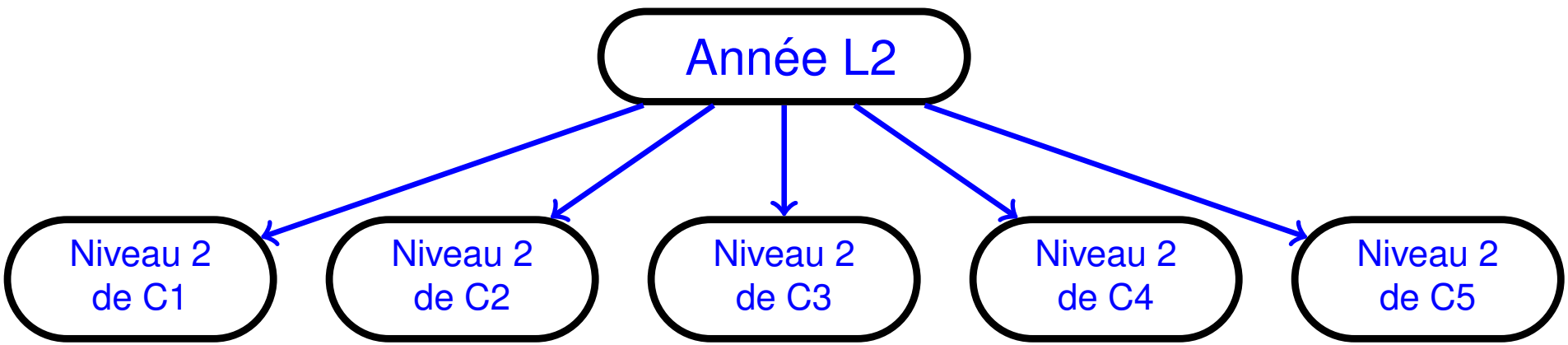
C2 : Développer des solutions informatiques

C3 : Gérer une solution informatique

C4 : Mettre en œuvre un projet informatique

C5 : Construire son projet professionnel.

Pour une année, on valide un niveau de compétence.



Validation d'une année

Calcul de la note d'une année

La note de l'année est égale à la moyenne pondérée des notes des niveaux des compétences.

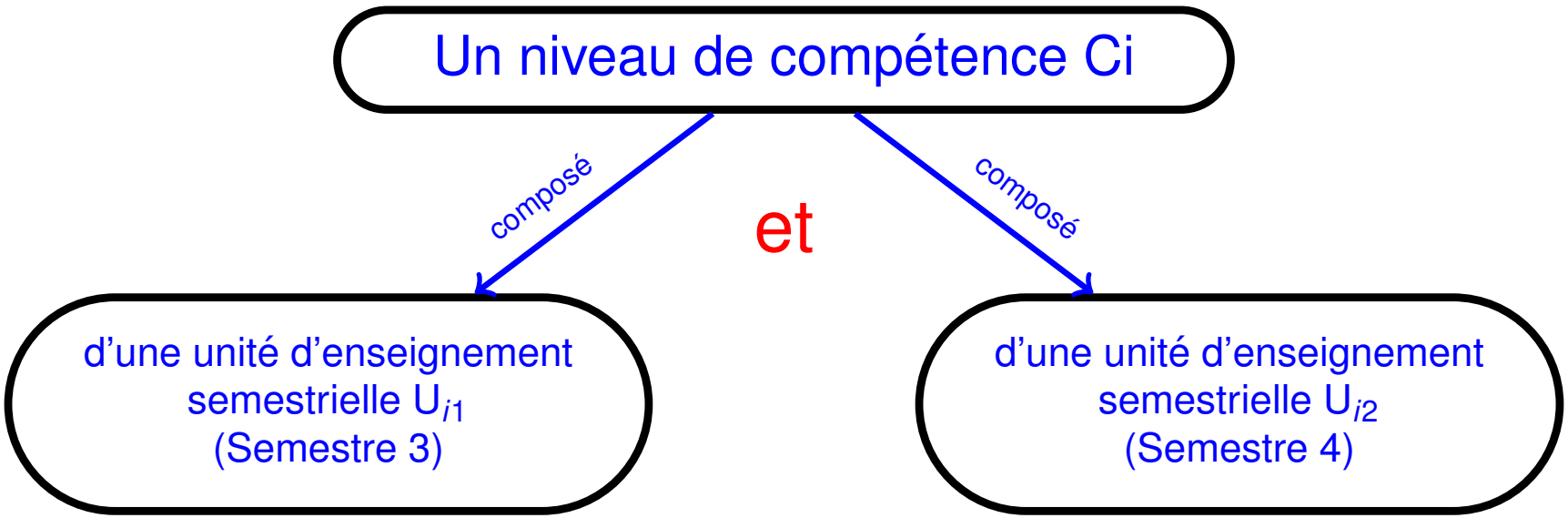
Validation d'une année

Une année est validée si :

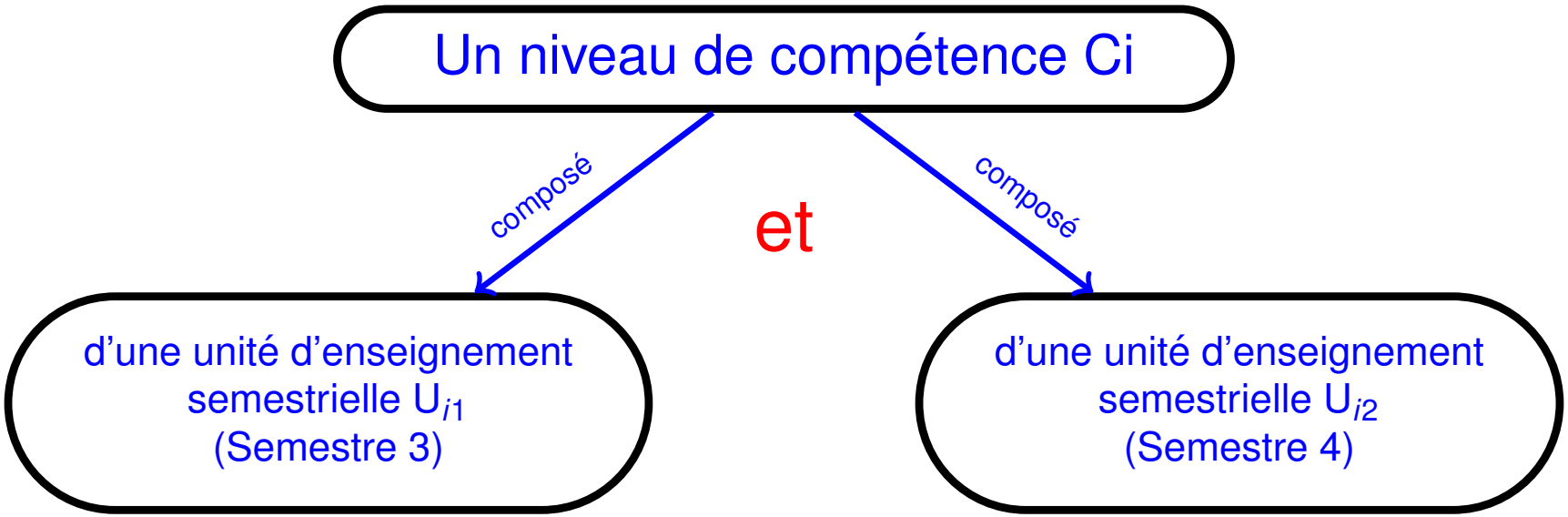
- la moyenne de l'année est supérieure ou égale à 10 **et**
- la note de chacun des niveaux de compétences est supérieure ou égale à 8.

‘

Validation d'un niveau de compétence



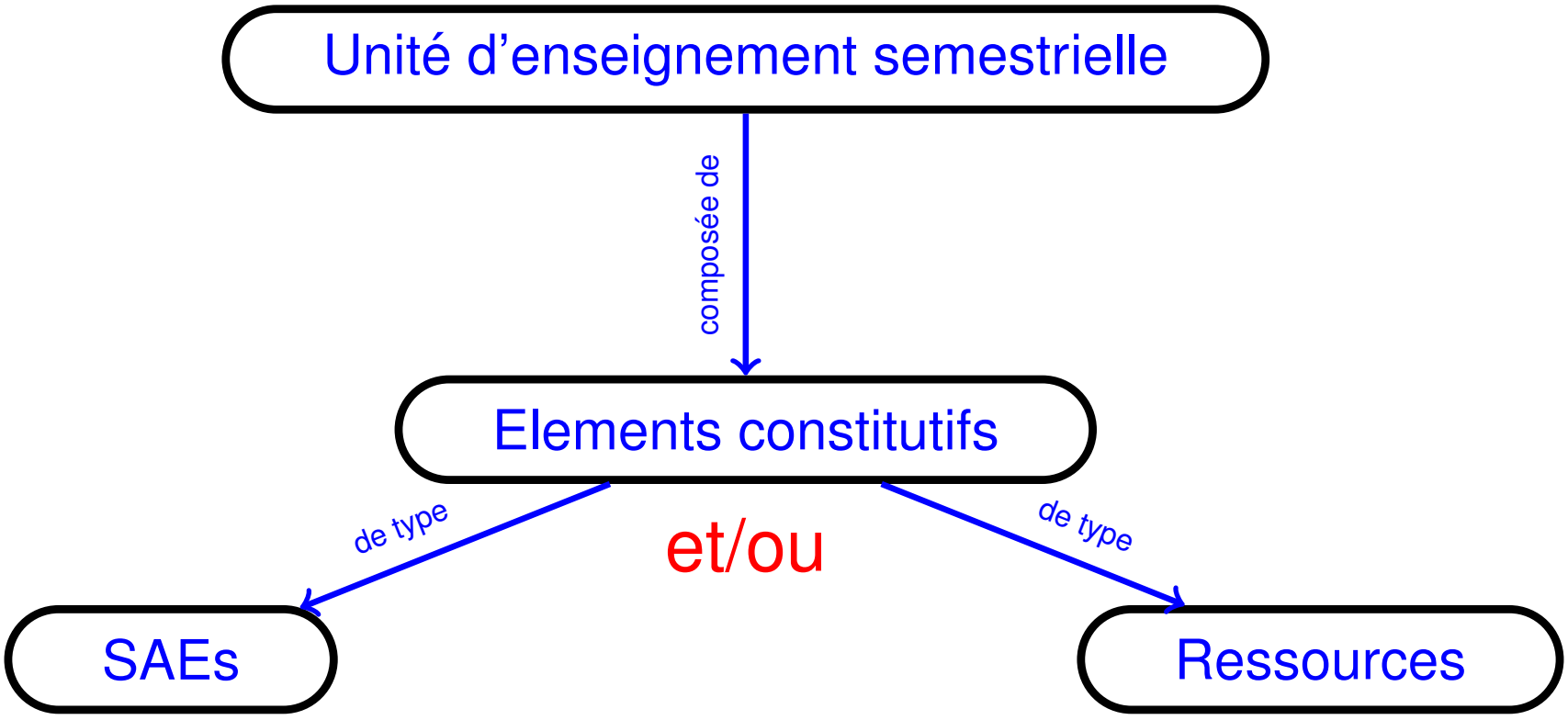
Validation d'un niveau de compétence



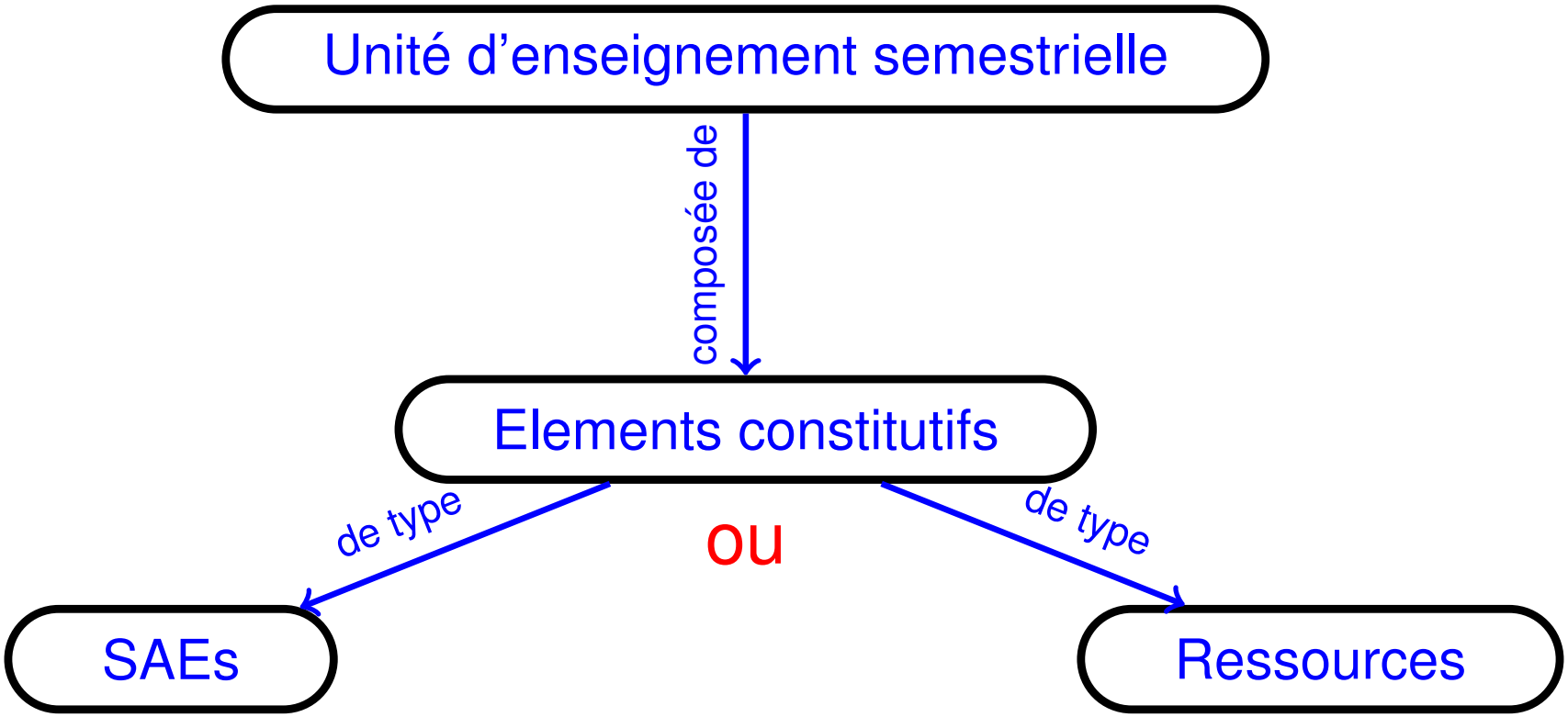
Validation d'un niveau de compétence

Un niveau de compétence est validé si la moyenne pondérée des deux unités d'enseignement semestrielles est supérieure ou égale à 10.

Validation d'une unité d'enseignement semestrielle



Validation d'une unité d'enseignement semestrielle

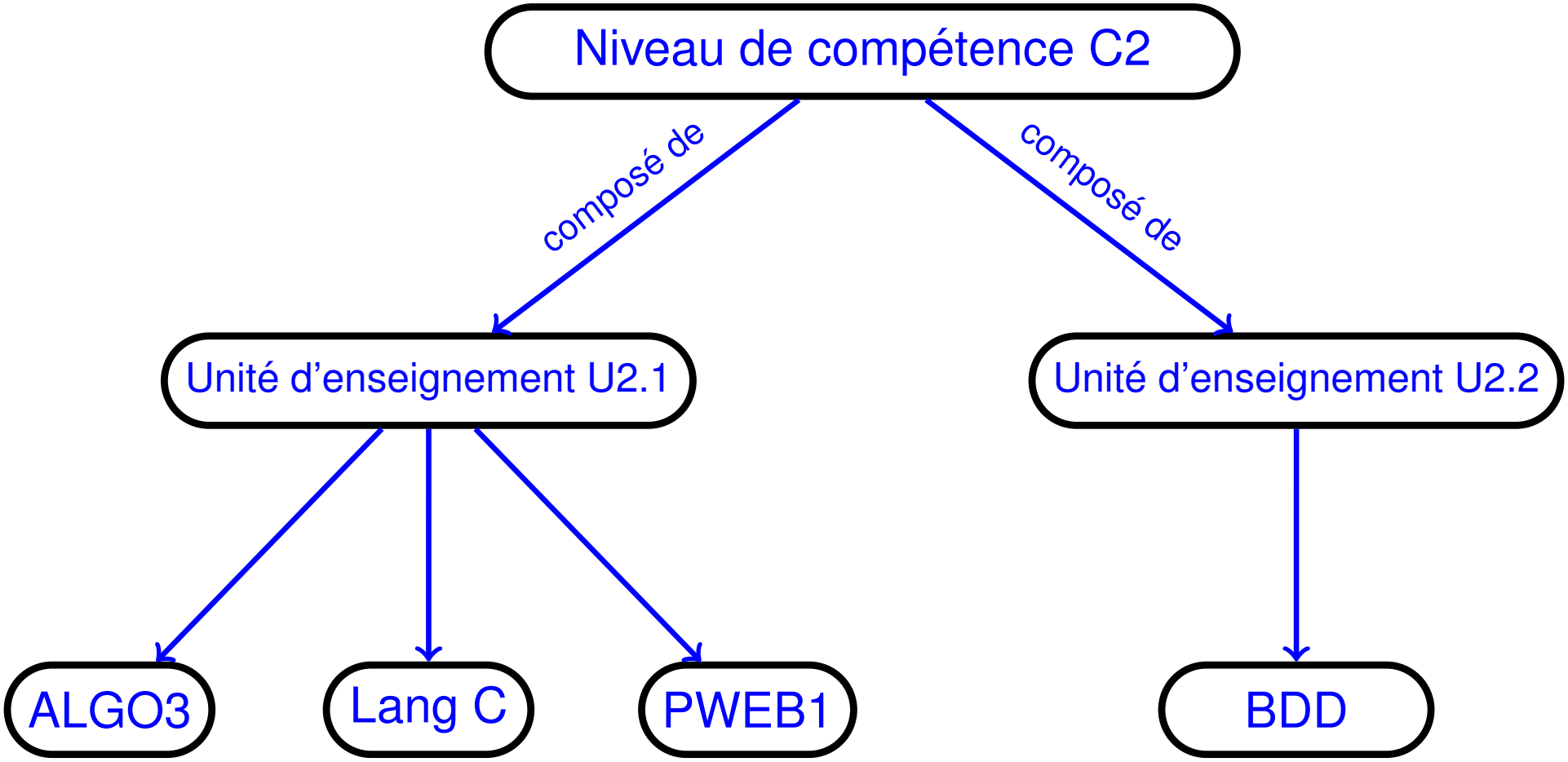


Validation d'une unité d'enseignement semestrielle

Une unité d'enseignement semestrielle est validée si la moyenne pondérée des éléments constitutifs est supérieure ou égale à 10.

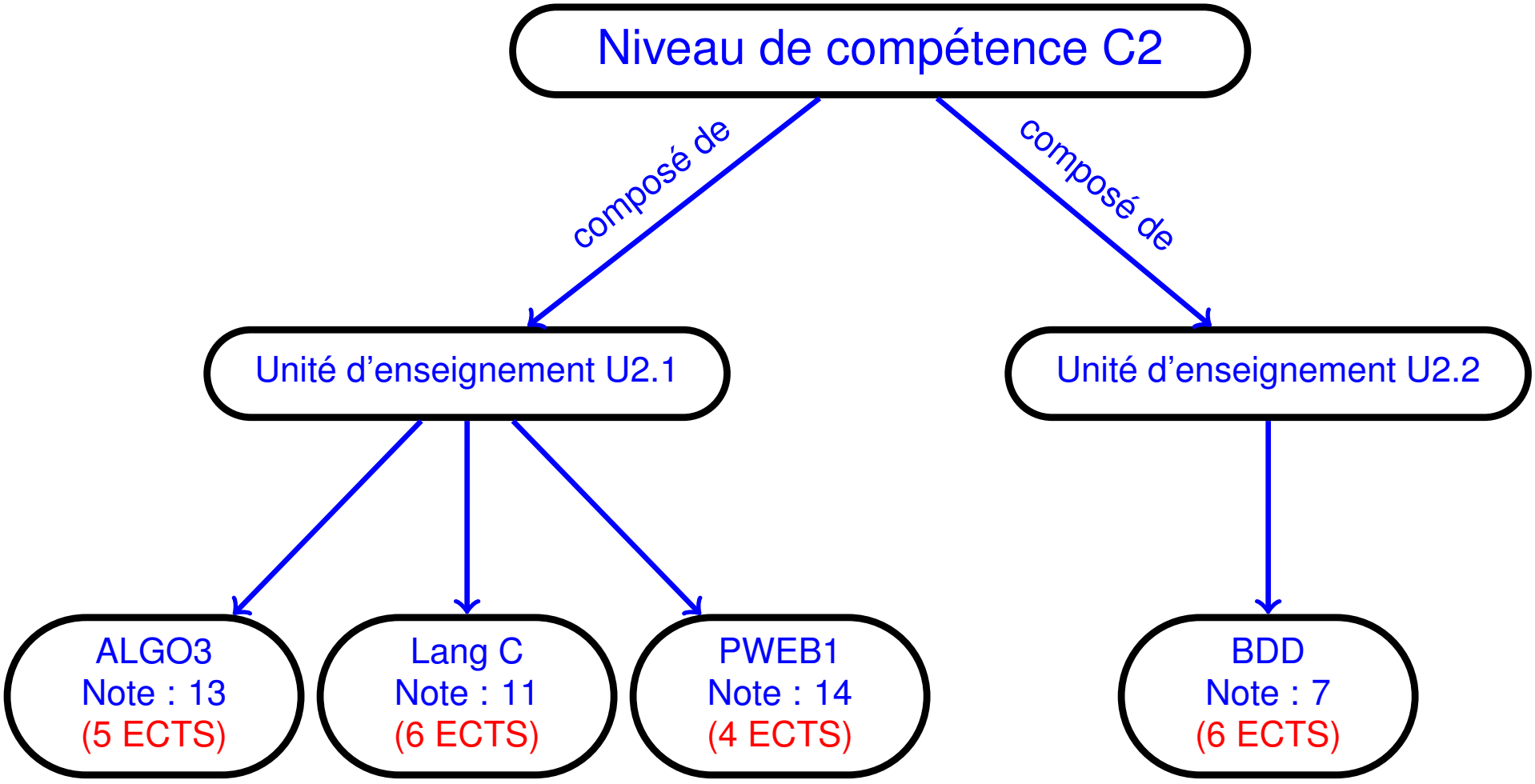
Exemples

Exemple : validation d'un niveau de compétence C2



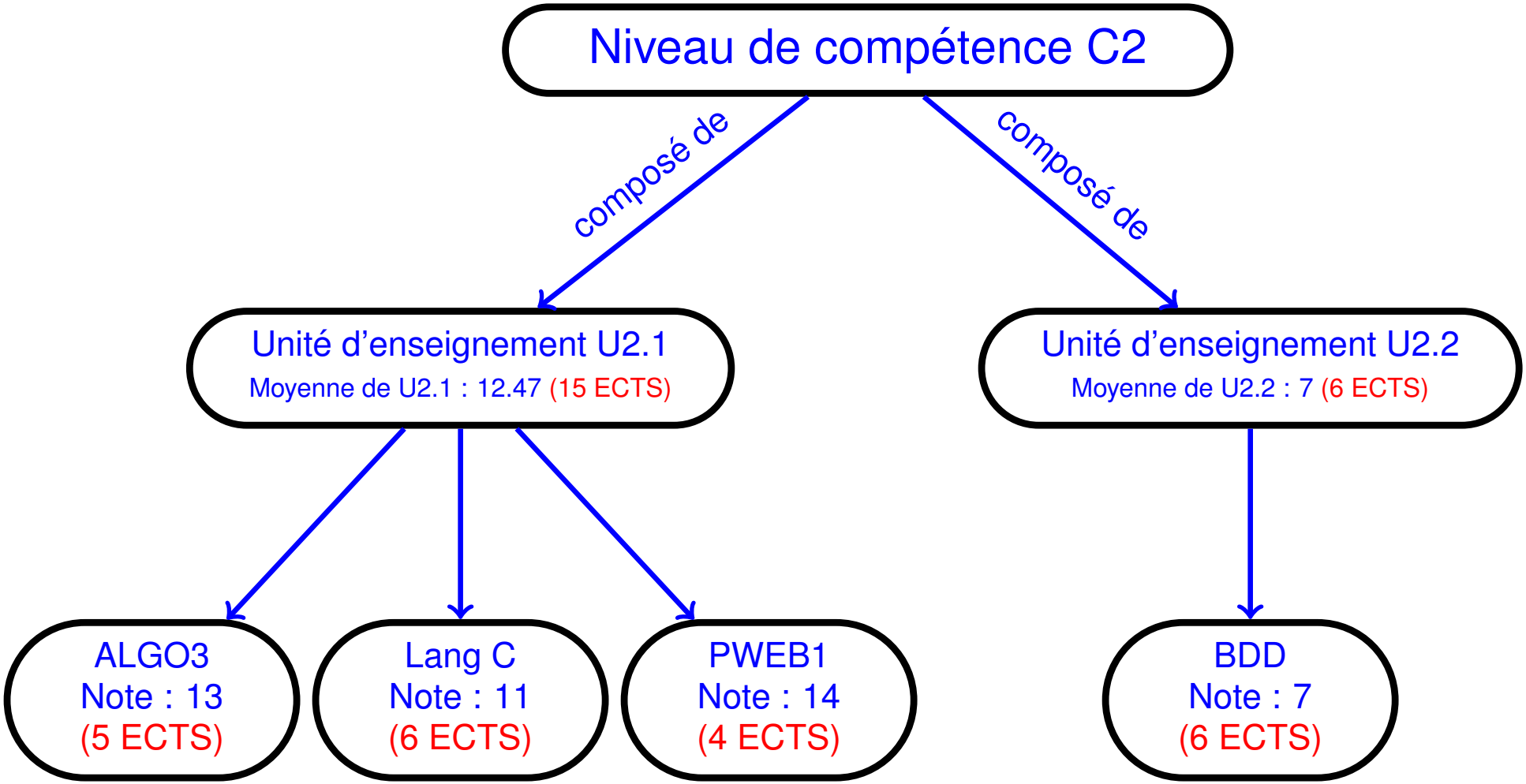
Exemple : validation d'un niveau de compétence C2

Seules les SAE et les ressources sont évaluées!



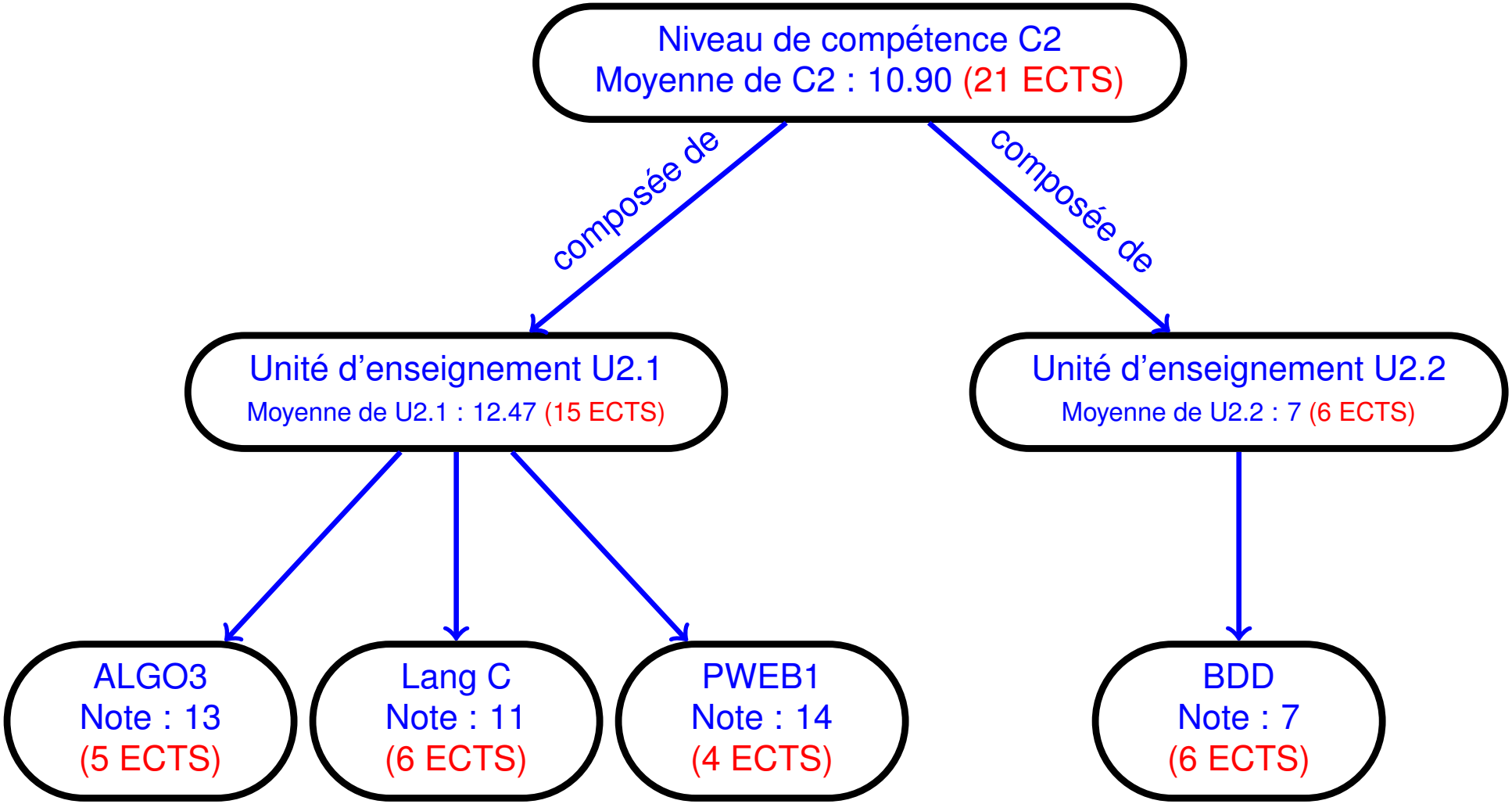
Exemple : validation d'un niveau de compétence C2

Propagation des ECTS (ou des pondérations) des feuilles vers la racine!



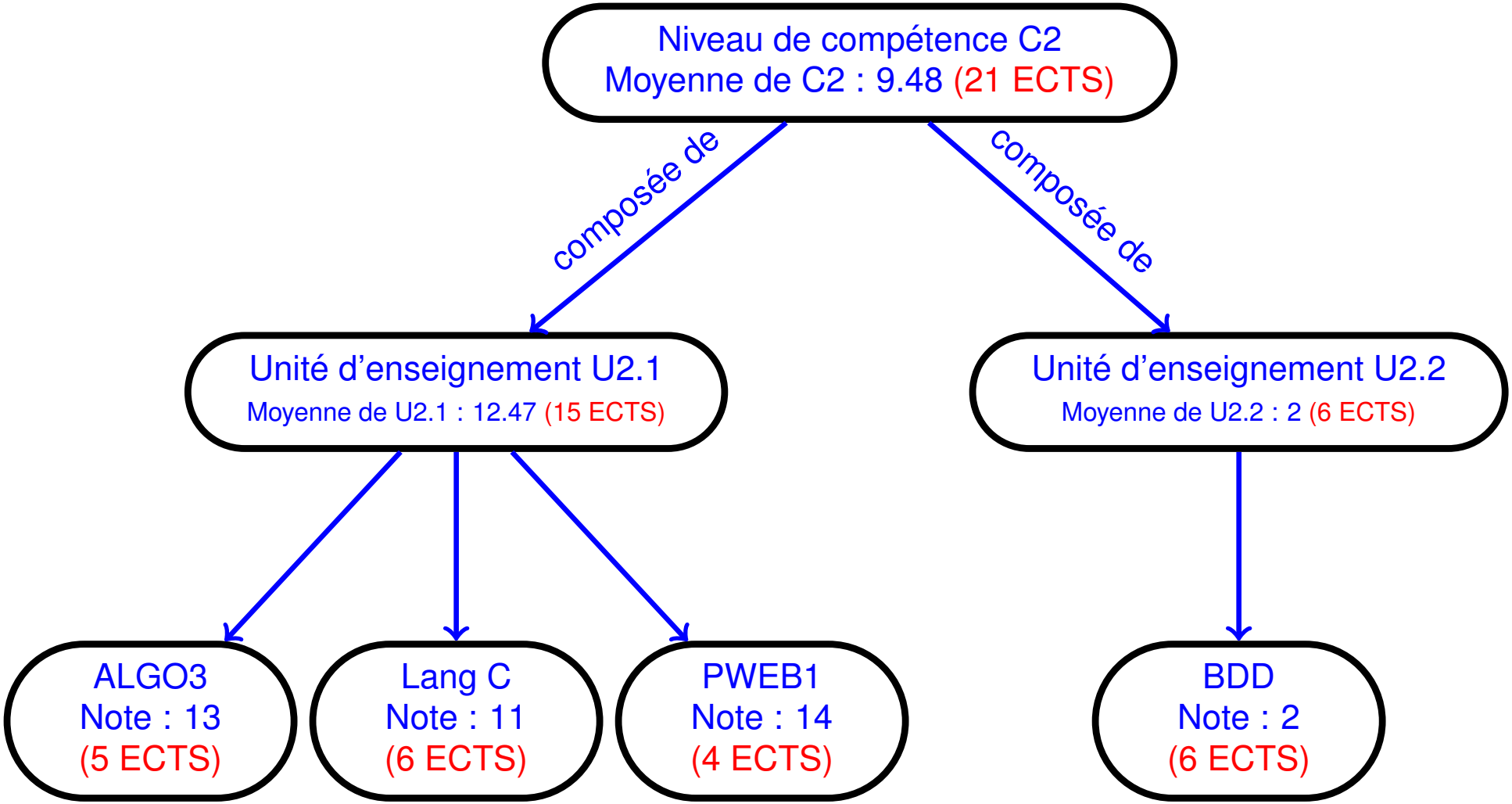
Calcul des notes des deux unités d'enseignement semestrielles

Exemple : validation d'un niveau de compétence C2



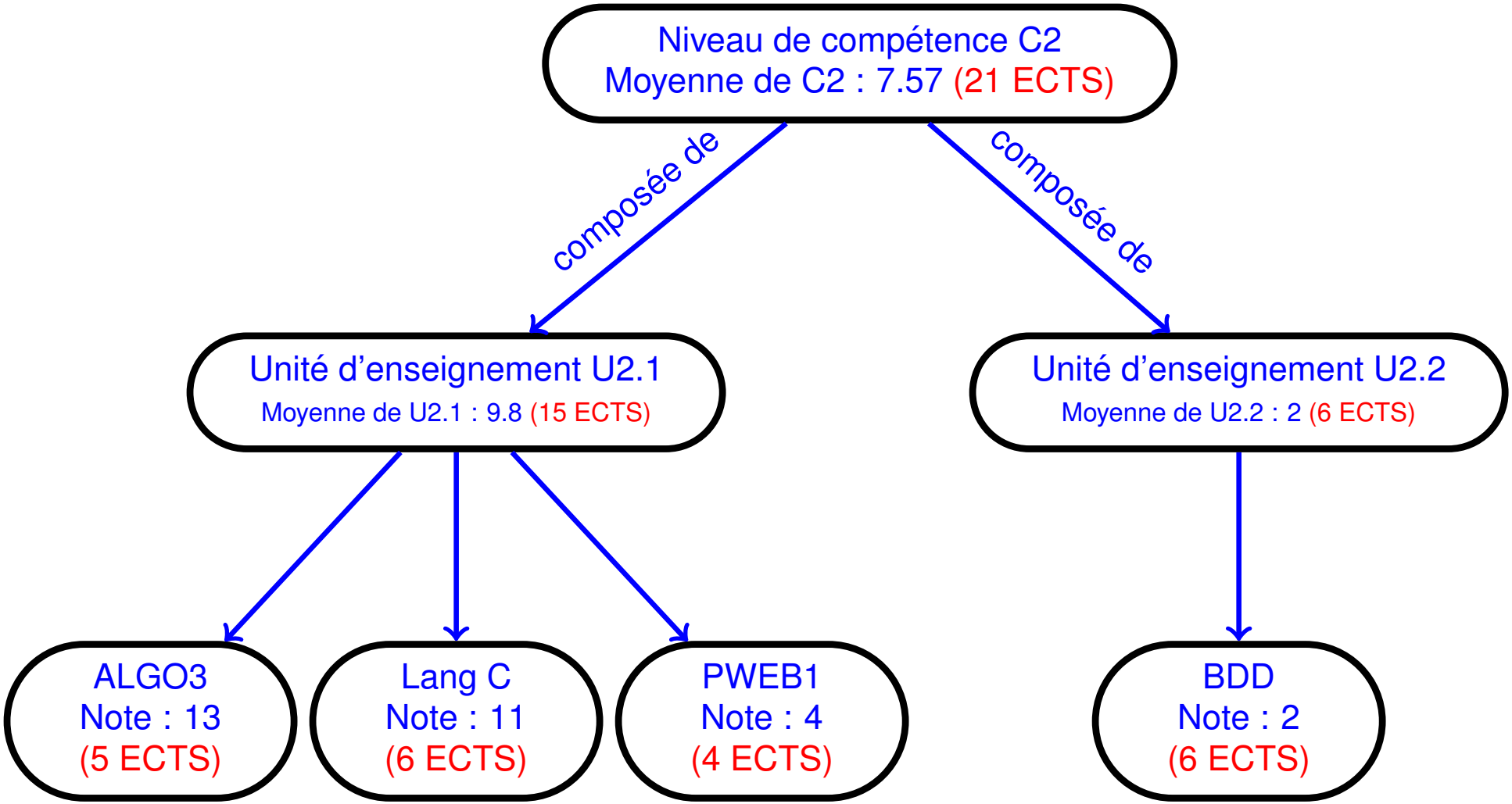
Calcul de la note du niveau de la compétence C2 : Validée !

Un autre exemple



Calcul de la note du niveau de la compétence C2 : Non validée.
Peut être compensée avec les notes des autres compétences.

Un dernier exemple



Calcul de la note du niveau de la compétence C2 : Non validée.
Ne peut pas être compensée (session 2).

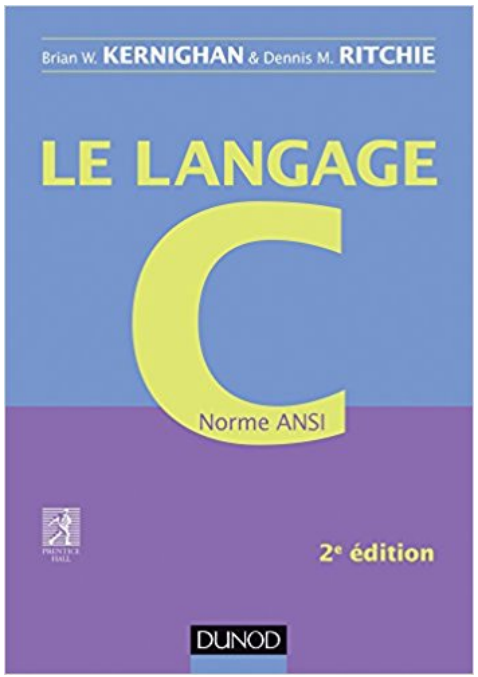
Pour plus de détails ...

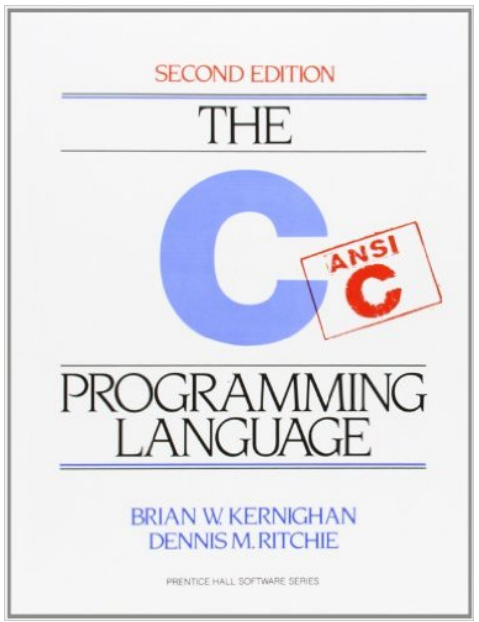
Bientôt sur Moodle ...

Introduction

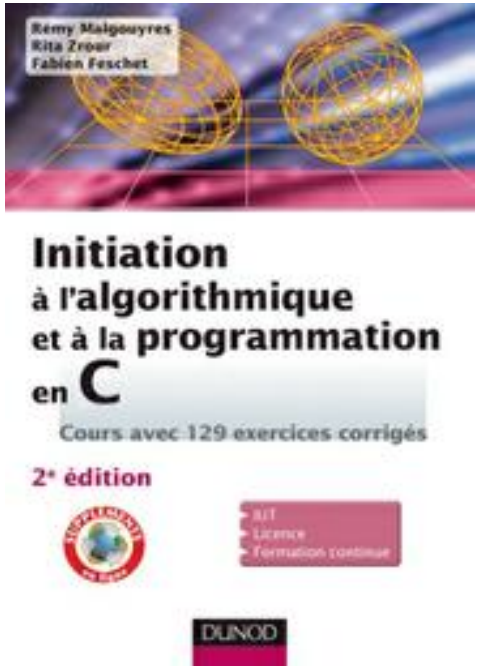
- Le langage C est écrit en 1972 par Dennis Ritchie (Bell Labs)
- Le langage C fait suite aux langages BCPL and B.
- Le développement du Langage C est étroitement lié à celui de Unix.
- Publication du livre ”The C Programming Language” en 1978 par Kernighan et Ritchie.
- En 1988, la norme ”ANSI C” (American National Standards Institute) a été proposée.

Des références bibliographiques





Des références bibliographiques



- Disponibles à la bibliothèque.

La version qui sera utilisée dans ce cours

- Nous utiliserons *principalement* la version du C ANSI standard.
- Cette version est également appelée : C89 ou C90.
- Une des motivations principales : la portabilité
  - Tous les compilateurs C acceptent la version du C ANSI standard.

Présentation informelle à travers deux exemples

Examples

Commençons par deux exemples de programmes écrits en C.

Commençons
par
un exemple très simple

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- L'instruction #include <stdio.h> (Standard Input/Output Header) permet d'inclure une bibliothèque qui contient les éléments nécessaires (comme la définition des fonctions) pour les opérations d'entrée/sortie.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- L'instruction #include <stdio.h> (Standard Input/Output Header) permet d'inclure une bibliothèque qui contient les éléments nécessaires (comme la définition des fonctions) pour les opérations d'entrée/sortie.
- Cette instruction est présente dans la quasi-totalité des programmes C.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- L'instruction #include <stdio.h> (Standard Input/Output Header) permet d'inclure une bibliothèque qui contient les éléments nécessaires (comme la définition des fonctions) pour les opérations d'entrée/sortie.
- Cette instruction est présente dans la quasi-totalité des programmes C.
- L'instruction #include <biblio.h> est équivalente à l'instruction "import" de Python.

Afficher un message de bienvenue

```c
#include <stdio.h>

/*
 Ce programme affiche un message
 */

int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Les commentaires sont délimités par ”/*” et ”*/”.
- Les commentaires peuvent-être placés n’importe où dans le programme.
- Certains compilateurs acceptent l’utilisation de ”//” pour exprimer des commentaires au niveau d’une ligne seulement.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Tout programme C contient nécessairement le mot clef ”main” (fonction principale).
- C’est le point de départ de l’exécution du programme C.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Tout programme C contient nécessairement le mot clef ”main” (fonction principale).
- C’est le point de départ de l’exécution du programme C.
- Le mot clef ”void” indique que la fonction main n’a pas d’argument.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Le mot clef ”int” indique que la valeur retournée par la fonction est de type entier.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Le mot clef ”int” indique que la valeur retournée par la fonction est de type entier.
- Contrairement à Python, en Langage C le type retourné par les fonctions doit être spécifié (”void” si la fonction ne retourne pas de valeurs).

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Le mot clef ”int” indique que la valeur retournée par la fonction est de type entier.
- Contrairement à Python, en Langage C le type retourné par les fonctions doivent être spécifiées (”void” si la fonction ne retourne pas de valeurs).
- On reviendra sur ces points plus tard.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- L'instruction "printf" (et non print comme en Python) permet d'afficher sur écran le message délimité par les guillemets.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- L'instruction "printf" permet d'afficher sur écran le message délimité par les guillemets.
- Le ";" (point virgule) indique la fin d'une instruction C.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- L'instruction "printf" permet d'afficher sur écran le message délimité par les guillemets.
- Le ";" (point virgule) indique la fin d'une instruction C.
- Le caractère d'échappement "\n" (new line) permet de revenir à la ligne.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- L'instruction "printf" permet d'afficher sur écran le message délimité par les guillemets.
- Le ";" (point virgule) indique la fin d'une instruction C.
- Le caractère d'échappement "\n" (new line) permet de revenir à la ligne.
- La définition de la fonction printf se trouve dans la bibliothèque "stdio.h" (d'où l'importance de l'inclure!).

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- L'instruction "printf" permet d'afficher sur écran le message délimité par les guillemets.
- Le ";" (point virgule) indique la fin d'une instruction C.
- Le caractère d'échappement "\n" (new line) permet de revenir à la ligne.
- La définition de la fonction printf se trouve dans la bibliothèque "stdio.h" (d'où l'importance de l'inclure!).
- Nous reviendrons en détails sur les opération d'entrée/sortie dans le prochain cours.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Les accolades permettent, de manière générale, de délimiter des blocs d'instructions.

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Les accolades permettent, de manière générale, de délimiter des blocs d'instructions.
- Les tabulations/indentations et les espaces n'ont pas d'importance au niveau de la compilation (contrairement à Python).

Afficher un message de bienvenue

```c
#include <stdio.h>
/*
 Ce programme affiche un message
 */
int main (void)
{
 printf ("Bienvenue dans ce cours du langage C! \n");
 return(0);
}
```

- Les accolades permettent, de manière générale, de délimiter des blocs d'instructions.
- Les tabulations/indentations et les espaces n'ont pas d'importance au niveau de la compilation (contrairement à Python).
- Mais ils sont importants pour la lisibilité des programmes.

Regardons un autre exemple

Un autre exemple

```c
#include <stdio.h>
/*  Ce programme fait : (à compléter) */
int main (void)
{
 int nombre, somme, i;
 somme=0;
 printf ("Merci d\'introduire un nombre entier positif : ");
 scanf("%d", &nombre);
 for (i=0; nombre!=0; i++)
 {
  somme = somme + (nombre % 10);
  nombre = nombre / 10;
 }
 printf ("xxxx (à deviner) est : %d\n", somme);
 return(0);
}
```

A votre avis : que fait ce programme ?

La somme des chiffres d'un nombre

```c
#include <stdio.h>
 /* Ce programme :
 - lit un entier (un nombre positif),
 - calcule la somme de ces chiffres, et
 - imprime le résultat  */
int main (void)
{
 int nombre, somme, i;
 somme=0;
 printf ("Merci d\'introduire un nombre entier positif : ");
 scanf("%d", &nombre);
 for (i=0; nombre!=0; i++)
 {
  somme = somme + (nombre % 10);
  nombre = nombre / 10;
 }
 printf ("La somme des chiffres de ce nombre est : %d\n", somme);
 return(0);
}
```

La somme des chiffres d'un nombre

- Nous avons déjà expliqué : l'inclusion (stdio.h), les commentaires, la forme simple de printf, le main, etc.
- Ce programme contient d'autres formes d'instructions.
- L'instruction :

```
int nombre, somme, i;
```

La somme des chiffres d'un nombre

- Nous avons déjà expliqué : l'inclusion (stdio.h), les commentaires, la forme simple de printf, le main, etc.
- Ce programme contient d'autres formes d'instructions.
- L'instruction :

```
int nombre, somme, i;
```

  - permet de déclarer trois variables appelées "nombre, somme, i", et

La somme des chiffres d'un nombre

- Ce programme contient d'autres formes d'instructions.
- L'instruction :

```
int nombre, somme, i;
```

  - permet de déclarer trois variables appelées "nombre, somme, i", et
  - d'associer un type (ici type entier) à ces variables grâce au mot clef "int" (integer).

La somme des chiffres d'un nombre

- Ce programme contient d'autres formes d'instructions.
- L'instruction :

```
int nombre, somme, i;
```

  - permet de déclarer trois variables "nombre, somme, i", et
  - d'associer un type (ici type entier) à ces variables grâce au mot clef "int" (integer).
  - Les variables en C sont déclarées et leur type doit-être précisé.
  - Nous reviendrons sur les types de bases utilisés dans quelques transparents.

La somme des chiffres d'un nombre

- L'instruction :

```
scanf("%d", &nombre);
```

  - permet de lire à partir d'un clavier une valeur entière et la mettre dans la variable "nombre".
  - La fonction "scanf" a deux paramètres :
    - le premier donne le type de données à lire (ici type entier indiqué par "%d") et
    - le deuxième indique la variable qui contiendra la donnée (ici "nombre").

La somme des chiffres d'un nombre

- L'instruction :

```
scanf("%d", &nombre);
```

  - permet de lire à partir d'un clavier une valeur entière et la mettre dans la variable "nombre".
  - La fonction "scanf" a deux paramètres :
    - le premier donne le type de données à lire (ici type entier indiqué par "%d") et
    - le deuxième indique la variable qui contiendra la donnée (ici "nombre").
  - Nous reviendrons sur le caractère & qui donne l'adresse où se trouve la variable "nombre".

Structure d'un programme C

Très schématiquement, un programme C est composée de trois parties :

- Une première partie qui contient des instructions (appelées directives) à l'intention du pré-processeur. Par exemple,
  - les instructions d'inclusion de librairies, comme "#include <stdio.h>"
  - ou encore les déclarations de constantes ou de macros "#define".
- Une deuxième partie qui contient la déclaration des fonctions et des variables globales.
- La dernière partie contient le programme principal qui est la fonction main.

Langage C :
Représentation
des données et des variables

Langage C : représentation des données et des variables

- Une présentation brève et informelle sur la représentation des données (en binaire) en mémoire.
  - Cette présentation est importante pour bien comprendre la gestion des variables par le langage C.
- Variables : descriptions et représentations.

Parlons d'abord des données

En Informatique :
on ne travaille
qu'avec
des "0" " " et des "1"

Pour être précis :
la mémoire
est une ”très” longue suite
de ”0” ” et de ”1”

Il est important
pour un langage
d'établir des conventions
pour interpréter chaque suite
de "0" " " et de "1"

Question

Que représente la séquence binaire suivante (composée de ”0” et de ”1”) :

01000001

Question

Que représente la séquence binaire suivante (composée de ”0” et de ”1”) :

01000001

Réponse majoritaire !

65

Question

En effet, le nombre décimal (base 10) associé au nombre binaire (base 2):

$$01000001$$

est :

$$1 * 2^0 + ... + 1 * 2^6 = 65.$$

Remarques

- Les applications à programmer contiennent des données de natures différentes.
- De ce fait, en mémoire nous ne représenterons pas que des entiers positifs !
- Par exemple, nous avons également besoin de représenter des caractères, des nombres négatifs, des réels, etc.
- Gardons en tête, les différents types de données doivent-être tous codés en mémoire que par des ”0” et des ”1”.

Remarques

- Comment représenter les caractères en binaire (base 2) ?
  - Un caractère est souvent représenté par un code ASCII.
  - Par exemple, le caractère ’A’ a le nombre 65 comme code ASCII.
  - De ce fait, le caractère ’A’ sera également codé en binaire par :

    01000001

- Il est important que le compilateur ait une convention pour dire si la suite 01000001 code un :

  entier positif ou un caractère.

Remarques

De même, la séquence binaire suivante (composée de ”0” et de ”1”) :

10000001

Peut représenter :

- un nombre positif (129), ou bien
- un nombre négatif (-127) en complément à 2.

Notion de type pour lever les ambiguités

Le langage C utilise la notion de **type** pour distinguer les différentes catégories de données à représenter.

Remarques

Que représente la longue séquence binaire suivante :

111110001111010101000010 111000000101001001010101
000011000011110111010001 101111101111110100000010
100011010101000011101100 100110001111111101010001
010100001000111

Remarques

Que représente la longue séquence binaire suivante :

111110001111010101000010 111000000101001001010101
000011000011110111010001 101111101111110100000010
100011010101000011101100 100110001111111101010001
0101000001000111

Pas de réponse unique

- Il faut d'abord préciser combien de données sont représentées par la séquence ?
- Pour chaque donnée, il faut spécifier :
  - son type, qui permet de connaître la longueur (ou la taille) de la données; et
  - l'indice début de la donnée.

Organisation des données en mémoire

- Les données sont regroupées en blocs :
  - Typiquement, chaque bloc est composé de 8 bits.
  - Chaque bloc est appelé, par abus de langage, un mot mémoire (byte).
  - Par abus de langage, on parle aussi d'octet.
  - Le mot mémoire peut-être vu l'unité utilisée pour quantifier la taille d'une donnée ou d'une mémoire.
- Les mots mémoires :
  - sont indexés de 0 jusqu'à (Taille_de_la_mémoire -1).
  - Ces indices sont appelés des adresses.
  - Ils sont écrits en héxadécimal.
    - avec le préfixe par 0x (une simple convention).

Organisation des données en mémoire

Par exemple, la longue séquence de bits, donnée précédemment, sera représentée en mémoire (en supposant que le premier mot est stocké à l'adresse "0x 0") par :

0x **0**	1	1	1	1	1	0	0	0
0x **1**	1	1	1	1	0	1	0	1
0x **2**	0	1	0	0	0	0	1	0
0x **3**	1	1	1	0	0	0	0	0
0x **4**	0	1	0	1	0	0	1	0
0x **5**	0	1	0	1	0	1	0	1
0x **6**	0	0	0	0	1	1	0	0
0x **7**	0	0	1	1	1	1	0	1
0x **8**	1	1	0	1	0	0	0	1
0x **9**	1	0	1	1	1	1	1	0
0x **A**	1	1	1	1	1	1	0	1
0x **B**	0	0	0	0	0	0	1	0
0x **C**	1	0	0	0	1	1	0	1
0x **D**	0	1	0	1	0	0	0	0
0x **E**	1	1	1	0	1	1	0	0
0x **F**	1	0	0	1	1	0	0	0
0x **10**	1	1	1	1	1	1	1	1
0x **11**	0	1	0	1	0	0	0	1
0x **12**	0	1	0	1	0	0	0	0
0x **13**	0	1	0	0	0	1	1	1
0x **14**	-	-	-	-	-	-	-	-
0x **15**	-	-	-	-	-	-	-	-
0x **16**	-	-	-	-	-	-	-	-
0x **17**	-	-	-	-	-	-	-	-
0x **18**	etc							

Notion
de variables

Les variables
et
leur type

Les variables

Une variable possède :

- un identificateur (nom),
- un type,
- une donnée,
- une taille mémoire
  - nombre de mots mémoire, de ”bytes”, d'octets, nécessaires pour stocker les données d'une variable.
  - voir la commande ”sizeof” en TP.
- une adresse
  - L'emplacement mémoire de la variable.
  - L'opérateur ”&” permet de donner l'adresse d'une variable.

Déclaration de variables

- Toute variable utilisée dans le programme C doit être déclarée (contrairement à Python).
- La déclaration suit la syntaxe suivante :

type nom_de_la_variable ;

- Par exemple :

```
int i;
float moyenne;
char c;
```

Déclaration de variables

- Toute variable utilisée dans le programme C doit être déclarée (contrairement à Python).
- La déclaration suit la syntaxe suivante :

type nom_de_la_variable ;

- Par exemple :

```
int i;
float moyenne;
char c;
```

- Plusieurs variables de même type peuvent être déclarées en les séparant par une virgule. Par exemple :

```
int i, somme;
double moyenne, ecart_type;
```

Identificateurs de variables

Alphabet autorisé

Un identificateur de variables est composé d'une suite :

- de lettres majuscules ('A' ... 'Z') ou minuscules ('a'...'z'),
- du caractère '_' (blanc souligné, tiret bas, ou encore underscore) et
- de chiffres ('0'–'9').

Remarques

- Un identificateur de variable ne doit pas commencer par un chiffre.
- Le langage C est sensible à la casse des lettres. Les deux identificateurs "Somme" et "somme" seront considérés comme deux objets différents.
- Il n'est pas permis d'utiliser des mots clefs du langage C (par exemple int, return) comme des identificateurs de variables.

Exercice

Questions

Parmi les identificateurs suivants, indiquer ceux qui sont valides en langage C :

- Date_13
- 13_date
- date_13
- Date#13

Réponse

- Date_13 ✓
- 13_date ✗
- date_13 ✓
- Date#13 ✗

Les types de base

Parmi les entiers ...

il y a des petits et des grands

mais aussi

des positif et des négatifs

$\Rightarrow$ **chacun aura son type**

Différent types de variables entières

(en fonction de la taille/signe des nombres)

Le langage C distingue plusieurs types de variables entières :

- char
- int (”integer”) **Si vous êtes perdu, utiliser int par défaut**
- short (ou short int)
- long (ou long int)

Remarques

- Ces types diffèrent :
  - par la taille des variables avec :
    La taille ”char” $<$ La taille ”short” $\leq$ la taille ”int” $\leq$ la taille ”long”.
  - De ce fait, en fonction du domaine des valeurs que peut prendre votre variable, vous choisirez le type associé.

Le type Char:

double usage

Le type char: première utilisation pour contenir des caractères

- Une variable de type ”char” est souvent utilisée pour stocker un seul caractère.
- Les constantes caractères sont contenues entre deux simples quotes. Par exemple, ’A’, ’$’, etc.
- Une variable de type ”char” est codée sur un seul mot mémoire (typiquement 8 bits).

Remarque sur les quotes et double quotes

Contrairement à Python, il existe une différence entre :

- Une simple quote (ou apostrophe) : réservé pour désigner **un seul** caractère
- Une double quote (ou guillemet) : réservé pour désigner **une chaine** de caractère (que nous verrons plus tard)

Le type char: deuxième utilisation pour représenter des petits entiers

- Le type ”char” peut-être utilisé pour représenter des tout petits nombres entiers.

```
#include <stdio.h>
int main (void)
{
 char a, b;
 a=1;
 b=a+4;
 return(0);
}
```

Le type char: remarque

- Les constantes caractères sont représentées par des des petits entiers (ne dépassant pas 255 ou 127).
- De ce fait, on peut utiliser tous les opérations arithmétiques (+,-,* etc.) sur ces variables. Par exemple, les instructions suivantes sont valides :

```
#include <stdio.h>
int main (void)
{
 char ch1, ch2;
 ch1='A';
 ch2=ch1+3;
 return(0);
}
```

- Question : que contient la variable ch2 ?

Le type char: remarque

- Les constantes caractères sont représentées par des des petits entiers (ne dépassant pas 255 ou 127).
- De ce fait, on peut utiliser tous les opérations arithmétiques (+,-,* etc.) sur ces variables. Par exemple, les instructions suivantes sont valides :

```
#include <stdio.h>
int main (void)
{
 char ch1, ch2;
 ch1='A';
 ch2=ch1+3;
 return(0);
}
```

- Ici ch2 code le caractère 'D'.

Le type char

- Un autre exemple

```
#include <stdio.h>
int main (void)
{
 char A, B;
 A=12;
 B=A+13;
 return(0);
}
```

- La valeur de B est bien sûr égale à 25.
- Une valeur de type char peut même contenir des petits nombres négatifs.

Le type char

Pour résumer

- Une variable de type char peut-être aussi bien utilisé :
  - pour représenter un seul caractère,
  - que pour représenter des petits nombres entiers.
- Au niveau de l'impression, la commande printf permet aussi bien d'afficher :
  - une variable char sous forme d'un caractère (grâce à %c)
  - que sous la forme de l'entier associé à ce caractère (grâce à %d).

Signed vs Unsigned

- Chaque type de variables entières (char, short, int, long) peut-être associé d'un préfixe : signed ou unsigned.
- Le préfixe "signed" précise que la variable peut contenir des nombres négatifs.
- Alors que le préfixe "unsigned" précise que la variable contient uniquement des nombres positifs.
- Le mot "signed" est considéré par défaut par la majorité des compilateurs C.

L'intérêt d'utiliser unsigned

- Si on a que des nombres positifs mieux vaut utiliser le préfixe unsigned.
- Ceci permettra de représenter plus de nombres positifs.
- Par exemple, dans le programme ci-dessous :

```
#include <stdio.h>
int main (void)
{
 unsigned char A;
 char B;
 ...
}
```

- La variable A est déclarée en "unsigned char" alors que B est déclarée en "char" (par défaut signed char).
- Le plus grand nombre positif que peut contenir A est 255 alors que le plus grand nombre positif que peut contenir B est 127 (voir limits.h).

Domaines de valeurs

- A chaque variable déclarée, un espace mémoire lui réservé.
- La taille de l'espace mémoire réservée dépend du type de la variable et est exprimée en "bytes" (généralement synonymes de "octets=8bits").
- Cette taille donne donc le domaine de valeurs que peut prendre la variable.
- Supposons que la taille d'une variable est égale à N bits (nécessairement un multiple de bytes), alors le domaine de valeurs des nombres **positifs** est de :
  - $[0, 2^N - 1]$ si le préfixe unsigned est utilisé,
  - $[0, 2^{N-1} - 1]$ sinon.

Domaines de valeurs des nombres négatifs

Remarques préliminaires

Il existe différentes façons de représenter des nombres négatifs. Le bit de poids le plus fort est réservé pour le signe (0 pour positif et 1 pour négatif).

- Complément à 1 :
  - Consiste tout simplement à prendre le complément de chaque bit.
  - Par exemple, le complément à 1 du nombre ”00001101” (13 en décimal) est 11110010.
- Complément à 2 :
  - Consiste d’abord à prendre le complément de chaque bit,
  - puis rajouter 1.
  - Par exemple, le complément à 2 du nombre ”00001101” (13 en décimal) est 11110011.

Domaines de valeurs des nombres négatifs

- Le complément à 2 est largement utilisé dans les architectures des ordinateurs modernes.
- Avec le complément à 2, le domaine de valeurs des nombres négatifs est : $[-2^{N-1}, 0]$.

Domaine de signed : résumé

Si une variable A est de type signed et est représentée sur N bits alors le domaine de valeurs de A est :

$$[-2^{N-1}, \quad 2^{N-1} - 1]$$

Problèmes du débordement de valeurs

- Que se passe-t-il si on utilise une valeur qui n'est pas admise dans le domaine de valeur d'une variable.

- Pour les variables de types unsigned, avec domaine de valeurs $[0, 2^N - 1]$, le module 2^N est appliqué.

Problèmes du débordement de valeurs

Exemple : considérons le programme suivant :

```
#include <stdio.h>
/*
 Débordement de valeurs avec warning
 */
int main (void)
{
 unsigned char  i;
 i=256;
 printf("La valeur de i est %d \n", i);
 return(0);
}
```

Exemple de débordement

- Au niveau de compilation : Le ”warning” suivant
  ”implicit conversion from ’int’ to ’unsigned char’ changes value from 256 to 0 [-Wconstant-conversion]”
  est généré pour l’instruction :

```
i=256;
```

- Ce warning dit :
  - Comme *i* déclaré en ”char” et que les constantes entières sont considérées comme de type ”int” alors une conversion de ”int” vers ”char” est effectuée.
  - Le résultat de cette conversion est : 256 modulo 256 = 0.
- Au niveau de l’exécution : sans surprise, la valeur ”0” est affichée.

Exemple de débordement

- Sur cet exemple, le ”warning” est instructif.
- Il alerte le programmeur sur la présence d'un débordement de valeur.
- Ceci n'est pas le cas de programmes où des débordements de valeurs peuvent avoir lieu sans qu'aucun warning sera générée.
- Un exemple simple sera donné en TP

Problèmes débordement: variables de type signed

- Soit *var* une variable de type Signed codé sur N bits. Les valeurs dans le domaine $[-2^{N-1}, 2^{N-1} - 1]$.
- Toute valeur affectée à *var* doit appartenir au final à cet intervalle.
  - Si une valeur est inférieur -2^{N-1} alors on lui rajoute 2^N.
  - Si une valeur est supérieur $2^{N-1} - 1$ alors on lui enlève 2^N.
  - On répète ces opérations jusqu'à ce que la nouvelle valeur soit comprise dans l'intervalle $[-2^{N-1}, 2^{N-1} - 1]$.

Débordements de valeurs

Remarques

- Le débordement de valeurs peut mener à des situations indésirables.
- Détails au niveau des TD/TP.

Représentation des nombres réels

Les nombres réels

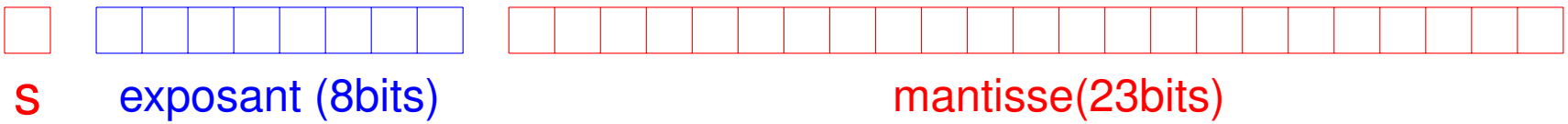
- Pour rappel, un nombre binaire avec virgule, s'écrit comme une somme de puissances (positives ou négatives) de 2.
- Par exemple, le nombre binaire avec virgule $11110,0111$ s'écrit de manière équivalente à :

  $$1*2^4+1*2^3+1*2^2+1*2^1+0*2^0+0*2^{-1}+1*2^{-2}+1*2^{-3}+1*2^{-4}$$

  qui est égal au nombre $30,4375$ en base 10.
- Il existe différentes façons de représenter les nombres binaires avec virgules en machines.
- La norme IEEE 754 est utilisée en C.
  - Précision simple.
  - Précision double.

La norme IEEE 754

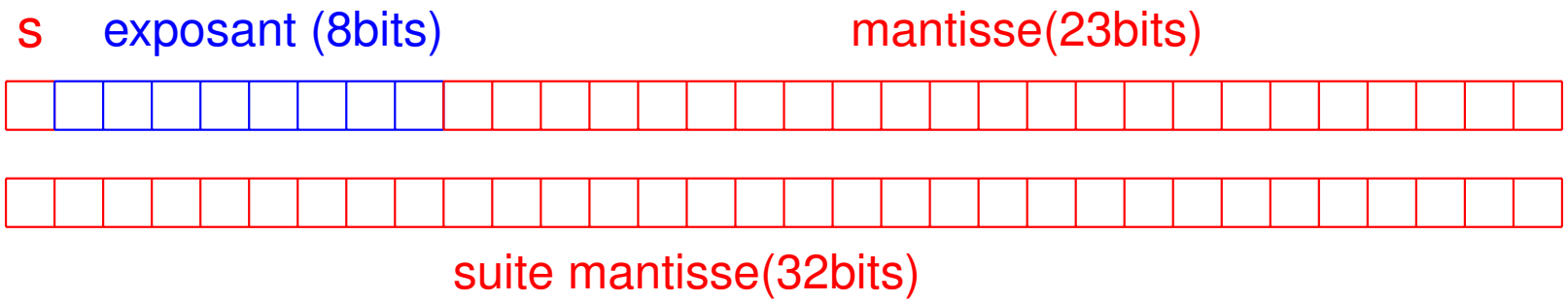
- Pour la précision simple, la norme IEEE 754 propose de représenter des nombres flottants sur 32 bits répartis sur trois champs :



- Le premier champs (un seul bit) est réservé pour représenter le signe du nombre.
  - ”0” pour les nombres positifs.
  - ”1” pour les nombres négatifs.
  - Pas de complément à 1 ni de complément à 2.
- Le deuxième champs (sur 8) et le troisième champs (sur 23bits) représentent respectivement l’exposant et la mantisse.

La norme IEEE 754

- Pour la précision double, la norme IEEE 754 propose de représenter des nombres flottants sur 64 bits :



- La seule différence (mais importante) avec la précision simple et la taille de la mantisse (55 bits au lieu de 23 bits).

Les nombres flottants en Langage C

- Les nombres réels sont représentés en utilisant deux types :
  - float (sur 4 octets) et
  - double (sur 8 octets) .

Attention!!!!!!

- Un nombre réel peut ne pas avoir son équivalent en binaire. Par exemple, le nombre binaire associé au nombre décimal ”0,1” est :

  0.0001100110011001100110011001100110011001100110011...

- De ce fait, sur ordinateurs, les nombres réels n’ont souvent qu’une représentation approchée.

Pas de type booléen

- Dans le langage C, il n'existe pas de type booléen proprement dit.
  - La valeur entière 0 (zéro) est interprétée comme "faux"
  - Toute valeur différente de 0 (zéro) est interprétée comme "vrai"
- De ce fait, le type int (ou char) peut-être utilisé pour coder des variables booléennes
- Presque comme en python :-)

Récapitulatifs sur les types de données

- Pour représenter des caractères, on utilise le type char
- Pour représenter des entiers, on utilise le type char, short, int, long. Chacun de ces types, peut-être préfixé de ”unsigned” ou ”signed”.
- Pour représenter (de manière approximative) des réels, on utilise le type float ou double en fonction de la précision souhaitée.

Représentation des données ou des constantes

Représentation des données entières

- Pour exprimer des données entières on peut utiliser :
  - une représentation décimale : par exemple 125, -345244, ou
  - une représentation octale (précédé d'un zéro): par exemple 0125 (équivalent à 85 en décimal), ou encore
  - une représentation hexadécimale (précédée d'un 0x ou 0X): par exemple 0x12 (équivalent à 18 en décimal).
- Soyez vigilants donc : en C, les deux données entières 0125 et 125 ne sont pas équivalentes!
- Par défaut, une constante entière est de type "int" sauf si on utilise les suffixes u (pour unsigned) ou l (pour long).

Représentation des données réelles

- Pour les données réelles on peut utiliser une représentation avec partie entière et décimale séparées par un ”.” (qui joue le rôle de la virgule) : par exemple 125.12, -344.14, ou
- Soyez vigilants donc : en C, les deux données 125 et 125. ne sont pas représentées de la même manière.

Déclaration de constantes

- Une constante est un objet dont la valeur ne change pas lors de l'exécution du programme.
- Une façon de la déclarer est d'utiliser la macro "#define" du pré-processseur C, avec la syntaxe :

#define nom_de_la_constante valeur

Déclaration de constantes

- Une constante est un objet dont la valeur ne change pas lors de l'exécution du programme.
- Une façon de la déclarer est d'utiliser la macro "#define" du pré-processseur C, avec la syntaxe :

#define nom_de_la_constante valeur

- Il s'agit d'une "méta-instruction" de substitution qui consiste à remplacer les occurrences de nom_de_la_constante par valeur.

Déclaration de constantes

- Une constante est un objet dont la valeur ne change pas lors de l'exécution du programme.
- Une façon de la déclarer est d'utiliser la macro "#define" du pré-processseur C, avec la syntaxe :

#define nom_de_la_constante valeur

- Il s'agit d'une "méta-instruction" de substitution qui consiste à remplacer les occurrences de nom_de_la_constante par valeur.
- Un exemple :

```
#define Vrai 1
#define Faux 0
#define nombre_de_joueurs 5
```

Déclaration de constantes

- Un autre façon de le faire est d'utiliser la même déclaration des variables, mais :
  - en rajoutant, en préfixe, le mot "const", et
  - en précisant la valeur de la constante.

Déclaration de constantes

- Un autre façon de le faire est d'utiliser la même déclaration des variables, mais :
  - en rajoutant, en préfixe, le mot "const", et
  - en précisant la valeur de la constante.
- C'est-à-dire, la déclaration d'une constante suit la syntaxe suivante :

```
const type nom_de_la_variable =valeur ;
```

Déclaration de constantes

- Un autre façon de le faire est d'utiliser la même déclaration des variables, mais :
  - en rajoutant, en préfixe, le mot "const", et
  - en précisant la valeur de la constante.
- C'est-à-dire, la déclaration d'une constante suit la syntaxe suivante :

  const type nom_de_la_variable =valeur ;
- Un exemple :

```
const int nombre_de_joueurs=5;
```

Récapilutons quelques points de syntaxe à retenir

- Toute instruction doit se terminer par ”;”
- Les commentaires sont compris entre ”/*” et ”*/”
- Les accolades, les parenthèses et les guillemets fonctionnent par paires.
- Toute variable utilisée doit être correctement déclarée.