Initiation à la programmation C

Rachida Chakir

rachida.chakir@esiee.fr
rachida.chakir@univ-eiffel.fr

Bureau B130 Batîment bienvenue

3R-IN1A
2024-2025

ESIEE PARIS

Organisation du module

Répartition des enseignements

- 4 séances de 2h de cours magistral
- 3 séances de 2h de Travaux Dirigés (TD)
- 7 séances de 3h de Travaux Pratiques (TP)

Evaluation

- 20% contrôle continue sous forme de 3 d'interrogations écrites $\sim$ 15 min
  - semaine 41 : Interrogation n°1 en début de TP2 (partie 1 et 2 du cours)
  - semaine 48 : Interrogation n°2 en début de TP4 (partie 1,2, et 3 du cours)
  - semaine 50 : Interrogation n°3 en début de TP6 (totalité du cours)

  Note de contrôle continue = Moyenne des 3 notes d'interrogations écrites
- 80% Examen écrit, le 20/12/2024 de 13h à 15h

ESIEE PARIS

Programme du cours

- Partie 1 : Les bases de la programmation en C
- Partie 2 : Pointeurs et fonctions
- Partie 3 : Tableaux et allocation mémoire
- Partie 4 : Les Types composés : les structures.

Programme du cours

- Partie 1 : Les bases de la programmation en C
- Partie 2 : Pointeurs et fonctions
- Partie 3 : Tableaux et allocation mémoire
- Partie 4 : Les Types composés : les structures.

Les différents langages de programmation

Qu'est ce qu'un langage de programmation ?

C'est un langage *artificiel* conçu pour communiquer des instructions à une machine sous la forme d'un programme.

On utilise pas les mêmes langages pour faire un OS, un tableur, un site web ou une application pour smartphone.

Il existe différents *styles* de programmation. Parmi les plus utilisés, il y a :

- la programmation *impératif*, comparable à une recette de cuisine : les opérations sont des séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme.
  → Exemples : C, C++, Java, PHP, JavaSript, Python ...
- la programmation *orienté objet* : on définit des briques logicielles appelées objets (représentant un concept, une idée ou toute entité du monde physique) puis les différentes interactions associées.
  → Exemples : Java, Python, Rhuby, JavaScript, C++, PHP

ESIEE PARIS

Les différents langages de programmation

Quelque soit le langage utilisé, il faut le traduire en langage machine.
Le processeur ne peut exécuter que les instructions écrites dans un langage binaire appelé langage machine. Ce langage est propre à chaque famille de processeurs.

Certains langages sont compréhensibles plus facilement par l'ordinateur que d'autre. On parle de langage de *bas niveau* ou *haut niveau* selon sa proximité avec le matériel.

- Un langage de *haut niveau* est un langage indépendant de la machine. Ils sont faciles à écrire.
- Les *langages de bas niveau* sont plus lisible pour la machine, et difficile à lire ou écrire pour les humains mais ils nécessitent beaucoup moins d'espace mémoire.

→ Un langage de bas niveau ira plus vite qu'un langage de haut niveau

ESIEE PARIS

Les langages de programmation et le langage machine

Pour traduire un programme en langage machine, il existe des approches différentes :

- **L'assemblage** : Les combinaisons de bits du langage machine sont représentées par des symboles dits ≪ mnémoniques ≫.

  Un langage d'assemblage ou assembleur est le langage de plus bas niveau. Le langage assembleur est intimement lié au processeur, ce qui permet de développer des programmes performants.
- **La compilation** (C,C++,...) : Traduction de l'ensemble du programme (code source) en langage machine (code objet). Le programme en langage machine sera ensuite exécuté.
- **L'interprétation** (Javascript, Python, PHP,...) : Traduction au fur et à mesure de l'exécution. On nomme souvent ces programmes des scripts.
- **La compilation** & **Inteprétation** (Java) : Compilation en un code intermédiaire qui n'est pas du code machine mais un code pour une "machine virtuelle". Ce code est ensuite interprété par un interpréteur (la machine virtuelle)

ESIEE PARIS

Les différents langages de programmation

Tous les langages n'ont pas les mêmes performances.

- Le langage C est un langage à *bas niveau*. Une fois compilé, le programme est directement compris par la machine.
- Python est un langage à *haut niveau*. Il est portable, mais interprété ligne par ligne, il est donc plus lent que le langage C.

Pourquoi le langage C ?

- Le langage C est léger et s'exécute très rapidement.
- Le langage C fournit un accès direct au matériel.
- Il est encore à la base de la plupart des systèmes d'exploitation actuels.

Sans le langage C, Python serait inutilisable sur le Raspberry Pi et dans le développement de logiciels embarqués (IoT).
Python lui-même est extrêmement lent, mais il s'appuie fortement sur les bibliothèques C pour le calcul haute performance et l'accès au matériel.

ESIEE PARIS

Programmation et méthodologie

1. Analyse du problème à résoudre
   - Comprendre la nature du problème posé
   - Préciser les données en *entrée*
   - Préciser les résultats que l'on désire en *sortie*
2. Conception d'une solution
   - Déterminer le processus de transformation des données en résultats
   - Écrire l'algorithme
3. Mise en œuvre (ou implémentation) de la solution
   - Programmation (ou codage) de l'algorithme dans un langage choisi
4. Phase de test de la solution

   L'objectif d'un test est de détecter les *éventuelles* anomalies du code
   - Sélectionner un jeu de données *en entrée*
   - Définir le résultat attendu en *en sortie*
   - Vérifier le résultat obtenu avec le résultat attendu

ESIEE PARIS

Programmation et algorithme

Qu'est ce qu'un algorithme

Méthode permettant de résoudre un problème de manière systématique (c'est à dire sans demander aucune initiative à celui ou celle qui l'exécute).

Un algorithme est une suite d'opérations, qu'on appelle *instructions*, à *exécuter* pour un atteindre un objectif. Les algorithmes ne sont pas spécifique à l'informatique, on peut par exemple citer :

- Une recette de cuisine,
- Un mode d'emploi,
- Une notice de montage, ...

Pour un problème donné il peut y avoir plusieurs algorithmes ou aucun

- Pour définir un algorithme, on a besoin d'un **langage abstrait, non ambigu et indépendant du langage de programmation**

Pour concevoir un programme informatique, on utilise le plus souvent du **pseudo-code** ou des **diagrammes**.

ESIEE PARIS

Programmation et algorithme

Patron d'un algorithme

Algorithme *NomDeMonAlgo*
Début
... actions
Fin

On définit l'instruction *Écrire* pour afficher du texte, ainsi que l'instruction *ALaLigne* pour poursuivre l'affichage à la ligne suivante

▶ Programme pour afficher "Hello !"

l'algorithme

Début
  Écrire ("Hello !")
  ALaLigne
Fin

en C

```
# include <stdio.h>
int main ( ) {
  printf("Hello !\n");
  return 0;
}
```

en Python

```
print ('Hello !')
```

ESIEE PARIS

Un premier programme en C : le fichier source (1)

☐ Un programme écrit en langage C, comporte obligatoirement une fonction principale appelée `main()` renfermant les instructions qui doivent être exécutées.

Main.c

```
# include <stdio.h>
int main ( ) {
  printf("hello \n");
  return 0;
}
```

- Le type retourné par la fonction `main()` est `int`
- La fonction `printf()` produit une émission de caractères en séquence vers la sortie standard (par défaut il s'agit de l'écran).
- Il faut inclure un fichier nommé *stdio.h* qui définit l'usage de cette fonction `printf()`.

ESIEE PARIS

Un premier programme en C : le fichier source (2)

Main.c

```
# include <stdio.h>
int main ( ) {
  printf("hello \n");
  return 0;
}
```

- La manière d'écrire un code en langage C a son importance.
- Le langage C est par exemple sensible à la casse (en anglais case sensitive), cela signifie qu'un nom contenant des majuscules est différent du même nom écrit en minuscules.

→ La fonction principale doit être appelée `main()` et non `Main()` ou `MAIN()`.
De la même façon, on remarquera que la fonction `printf()` est écrite en minuscules.

ESIEE PARIS

Un premier programme en C : le fichier source (3)

Attention : **Le programme source Main.c ne peut pas être exécuté de manière immédiate par l'ordinateur** tel qu'il se présente à nos yeux.
Il faut le traduire en langage machine.
→ On utilise un programme destiné à le traduire : le compilateur C.

Main.c

```
# include <stdio.h>
int main ( ) {
  printf("hello \n");
  return 0;
}
```

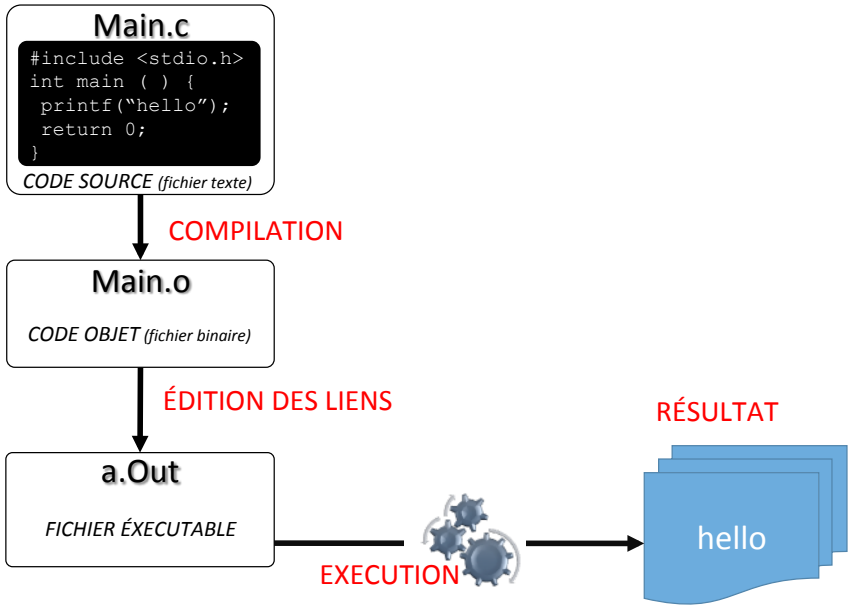
- L'instruction `printf()` se termine par un « ; », car en langage C, toute instruction se termine par un point virgule.
- Le compilateur détecte ainsi la fin d'une instruction.
- Le « ; » est ce que l'on appelle un délimiteur.

ESIEE PARIS

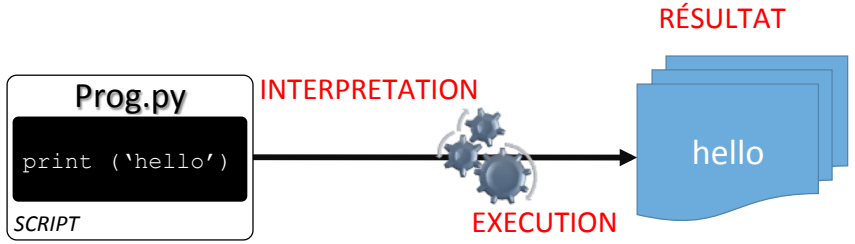
La compilation (1)

→ Le C est un langage *compilé* par opposition aux langages *interprétés* (comme le java, le python, HTML).

Le C, un langage compilé

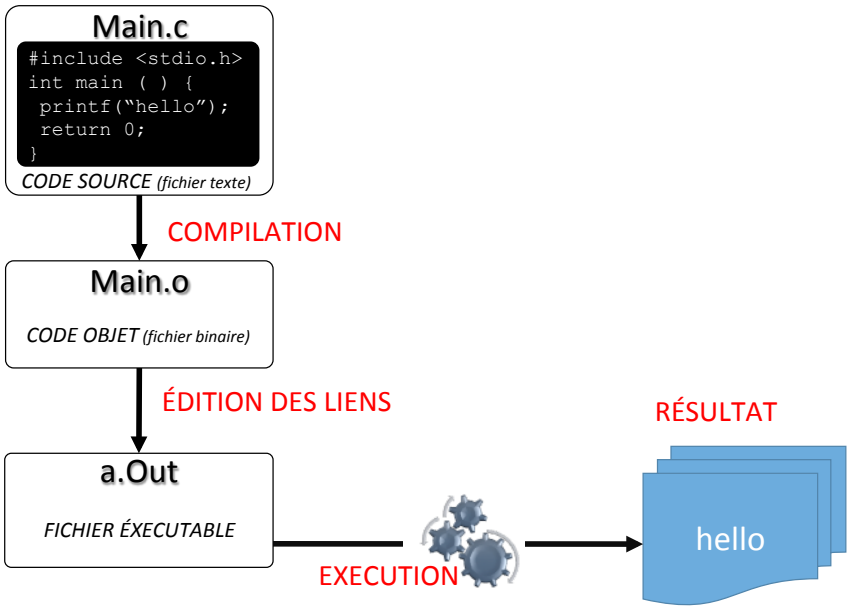


Le python, un langage interprété



La compilation (3)

Le C, un langage compilé



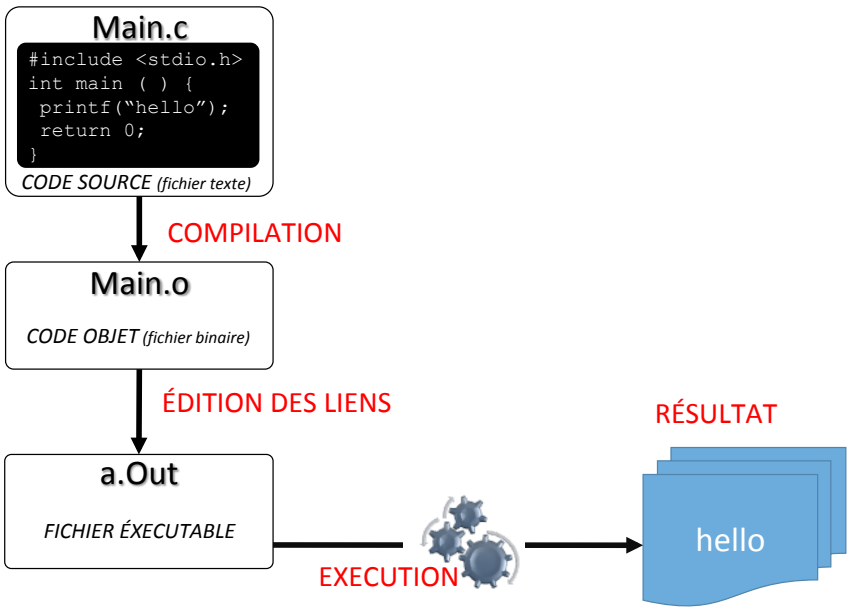

```
$ gcc Main.c -o a.Out
```

- La compilation d'un programme C consiste à traduire le code *source* (les fichiers *.c*) en langage machine.
- Le fichier produit est un code *objet* (les fichiers *.o*). Le fichier *objet* est un fichier binaire.
- L'édition des liens : Lien avec les bibliothèques standards ou entre les différents fichiers *.o* lorsque le programme est séparé en plusieurs fichiers sources.
- L'édition des liens produits alors un fichier *executable*.

ESIEE PARIS

La compilation (3)

Le C, un langage compilé




```
$ gcc Main.c -o a.Out
```

- Les étapes de compilation et d'édition des liens sont généralement automatiquement enchaînée. Les fichiers intermédiaires issus de la compilation sont automatiquement détruits.
- Il est cependant possible d'utiliser des options pour les effectuer séparément afin d'obtenir ces fichiers intermédiaires.

La compilation (4)

Main.c

```
#include <stdio.h>
int main ( ) {
 printf("hello");
 return 0;
}
```

CODE SOURCE (fichier texte)

Compilation et édition des liens en une étape

```
$ gcc Main.c -o a.Out
```

Compilation

```
$ gcc Main.c -c
```

Edition des liens

```
$ gcc Main.o -o a.Out
```

La compilation (4)

Main.c

```
#include <stdio.h>
int main ( ) {
 printf("hello");
 return 0;
}
```

CODE SOURCE (fichier texte)

Compilation et édition des liens en une étape

```
$ gcc Main.c -o a.Out
```

Compilation

```
$ gcc Main.c -c
```

Edition des liens

```
$ gcc Main.o -o a.Out
```

Quand le programmeur déclenche la compilation, le compilateur affiche éventuellement des messages. Deux types de messages existent :

- Error

  Ce qui signifie que la compilation a échoué en un point du programme. Le terme "error" est suivit d'un message qui est censé vous aider à trouver la nature de cette erreur.

- Warning

  C'est un avertissement. Le compilateur a réalisé le travail mais vous signale qu'il a détecté un problème potentiel quand vous exécuterez votre programme. A vous de trouver la cause de cet avertissement.

ESIEE PARIS

La compilation (5)

Main.c

```
#include <stdio.h>
int main ( ) {
 printf("hello");
 return 0;
}
```

CODE SOURCE (fichier texte)

Compilation

```
$ gcc Main.c -c
```

Edition des liens

```
$ gcc Main.o -o a.Out
```

Compilation et edition des liens en une étape

```
$ gcc Main.c -o a.Out
```

→ Un code C correctement écrit ne génère évidemment aucun message ≪ error ≫ et s'il est vraiment bien écrit aucun message ≪ warning ≫.

Options de compilation (obligatoire pour l'unité)

- `-ansi` indique que le *standard* du langage C utilisé est celui de la norme ANSI/ISO
- `-pedantic` garantit la portabilité du code sur tous les compilateurs qui respecte la norme ANSI/ISO
- `-Wall` affiche des avertissements (warning) supplémentaires

ESIEE PARIS

La compilation (5)

Erreur de syntaxe

Elles arrivent lorsque l'on ne respecte pas les règles de syntaxe du langage C.

Les plus courantes sont :

- Oublie d'une accolade "{" ou "}", une parenthèse non fermé, ...
- Oublie du point virgule " ;" en fin d'instruction
- Vouloir utiliser une variable que l'on n'a pas déclaré

Les variables

- Un programme ne fera que **traiter que des données**
- Pour manipuler des données, on utilise des **variables**
- la valeur stocké dans une variable peut être modifié durant l'exécution, contrairement aux constantes.
  Par exemple si on devait programmer la fonction

$$f(x, y) = 5x + 3y$$

  x et y serait des variables, tandis que 5 et 3 serait des constantes.
- **Une variable a**
  - un nom, qu'on appelle **identifiant** et qui renvoie à une position dans la mémoire (**une adresse**)
  - un **type**, le type définit la façon dont la variable est représenté en mémoire, il spécifie aussi sa taille (la longueur de la séquence de bit) soit habituellement 8, 32 ou 64 bits.
  - une **valeur**, c'est la séquence de bits elle même.

ESIEE PARIS

Les variables dans les algorithmes

Pour créer une variable, il faut la **définir** en lui donnant **un nom** (éloquent et précis) et la **déclarer** en lui précisant un **type**.

Les variables dans les langages de programmation

Dans les langages compilés, **les variables doivent être déclarées en précisant leur type**.

Dans les langages interprétés, seules les valeurs ont un type ce qui n'oblige pas toujours à déclarer les variables.

En C

```c
/* Une variable entière */
int i = 0;
/* Une variable réelle */
float j = 2.5;
```

En Python

```python
# une variable entière
i = 0;
# une variable réelle
j = 2.5;
```

ESIEE PARIS

Les variables

Le C est un langage typé

- **les variables doivent être déclarées en précisant leur type**.
- Les types de base en C concernent les caractères, les entiers et les flottants (nombres réels).
- Ils sont désignés par les mots-clefs suivants :
  char *int* *float* *double* *short* *long* *unsigned*
  - *char* : petit entier, convient pour les caractères
  - *int* : pour les valeurs numériques entières
  - *float* : nombres à virgule flottante en simple précision
  - *double* : nombres à virgule flottante en double précision

  On peut ajouter des mots-clés particuliers pour modifier la plage de valeurs (*short*, *long*) et/ou le fait que la valeur soit signée ou non (*signed*, *unsigned*).

ESIEE PARIS

Les types prédéfinis et variables

Le type *char*

- Le type *char* permet de stocker des nombres compris entre 0 et 256. Mais il faut savoir qu'en C on l'utilise rarement pour ça.
  - Il est en fait prévu pour stocker un caractère
- En effet, la plupart des caractères « de base » sont codés entre les nombres 0 et 127. Une table fait la conversion entre les nombres et les lettres : la table ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

  - Le langage C permet de faire très facilement la traduction lettre <=> nombre correspondant.

ESIEE PARIS

Les types prédéfinis et variables

Les noms de variables

- l'identificateur (le nom de la variable) commence forcement par une lettre [a - z] [A - Z] ou par le caractère _
  l'identificateur peut aussi contenir des chiffres.
- les mots clés du langage sont réservés.

Les 32 mots clés du C

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Les variables

Déclarations de variables

`type nom_variable;`

- La déclaration de variables se fait uniquement en début de bloc { }
- Initialisation possible (et fortement conseillé) :

```
int i = 3;
```

- Déclarations multiples :

```
int i = 7, j = 6, k = -5;
```

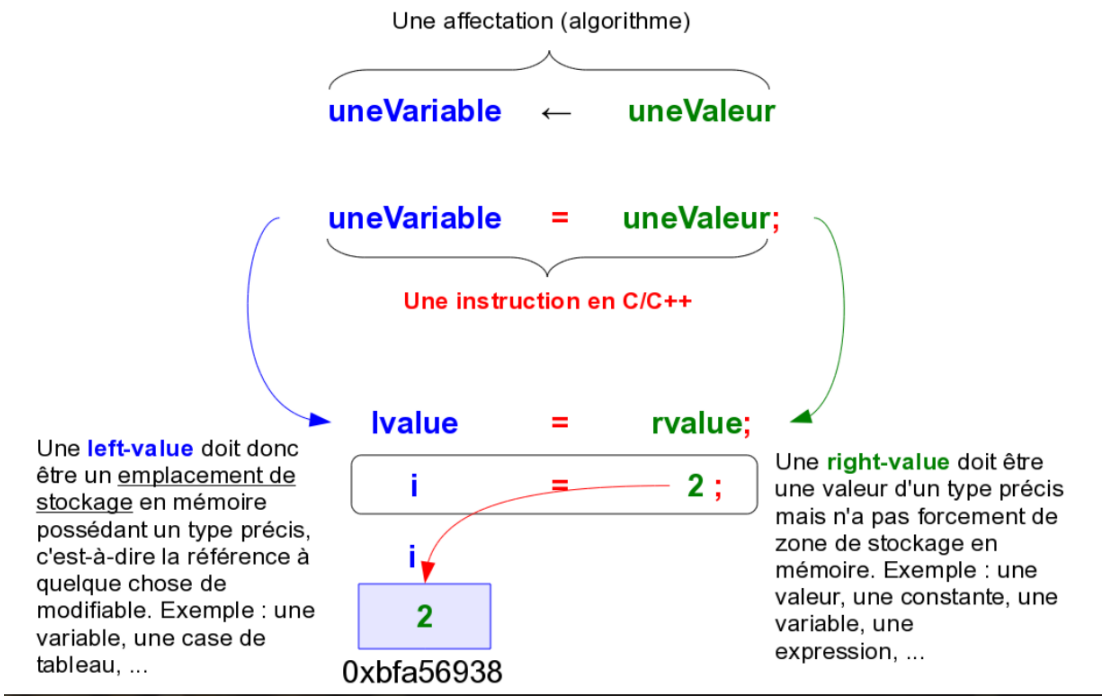
Les instructions dans un programme

→ Un programme comporte deux types d'instructions

- **Les instructions de base**
  - Elles permettent de manipuler les variables :
    - affectation
    - lecture / écriture
  - Elles permettent de faire des opérations : addition, soustraction, multiplication, division
- **Les instructions de structuration**
  - Elles servent à préciser comment doivent s'enchaîner chronologiquement les instructions de bases. Il existent deux types.
    - les instructions conditionnelles (if/else, switch)
    - les structures itératives (les boucles)

ESIEE PARIS

La première des instructions est la possibilité d'*affecter une valeur à une variable*.



Opérateur *affectation* en C

'=' est l'opérateur d'affectation

```
i = 2;
```

On affecte la valeur 2 à la variable `i`,

Algorithme vs Programmation
L'affectation

l'algorithme

variable i : entier
i $\leftarrow$ 0
variable j : réel
j $\leftarrow$ 2.5

en langage C

```
int i = 0;
float j ;
j = 2.5;
```

en Python

```
i = 0;
j = 2.5
```

Les entrées/sorties

printf - fonction d'affichage sur la sortie standard

- Les variables sont indiquées par des caractères spéciaux, qu'on appelle *format de sortie* :
  %d : nombre entier, %c : caractère, %s : chaine de caractère, %f : nombre réel (écrite décimale), %e : nombre réel (écriture exponentielle)
- %.5f réel avec 5 chiffres après la virgule
- \n : saut de ligne ; \t : tabulation

Example

```
char cara = 'a';
printf(" cara = %d \n", cara);
printf(" cara = %c \n", cara);
```

ESIEE PARIS

Les entrées/sorties

printf - fonction d'affichage sur la sortie standard

- Les variables sont indiquées par des caractères spéciaux, qu'on appelle *format de sortie* :
  %d : nombre entier, %c : caractère, %s : chaine de caractère, %f : nombre réel (écrite décimale), %e : nombre réel (écriture exponentielle)
- %.5f réel avec 5 chiffres après la virgule
- \n : saut de ligne ; \t : tabulation

Example

```
float b = 3, c = 7;
float d = b/c;
printf(" b = %f et c = %f \n", b,c);
printf(" d = %.5f \n", d);
```

Les entrées/sorties

scanf - fonction de saisie au clavier

On peut lire des nombres *entiers* (int) ou *réels* (float), des *caractères* (char) ou des *mots* (chaine de caractère)

Syntaxe

scanf ("% *format d'entrée*", adresse de la variable)

Example

```
int c;
scanf("%d, &c);
```

- %d est le format d'entrée pour une variable de type int
- & c'est pour indiquer la valeur saisie sera enregistré dans la variable c
- L'operateur '&' devant les noms de variables

ESIEE PARIS

Les entrées/sorties

Le type *char*

- Le langage C permet de faire très facilement la traduction lettre <=> nombre correspondant.

  Testons le programme suivant

```
int main( ) {
{
      char lettreMaj= 'A';
      char lettreMin= 'a';
      printf( " A => %d et a => %d\n" , lettreMaj,lettreMin);
  return 0;
}
```

  A l'exécution il renvoie

```
A => 65 et a => 97
```

  - En effet, la lettre 'a' n'est pas identique à la lettre 'A', l'ordinateur faisant la différence entre les majuscules et les minuscules (on dit qu'il « respecte la casse »).

Les entrées/sorties

Le type *char*

❑ Entrée d'un caractère unique

- En ligne de commandes, un programme demande souvent un caractère à l'utilisateur.

Testons le programme suivant

```
int main( ) {
      char m;
      printf( " Entrer un caractere ? " );
      scanf( "%c" , &m);
      printf("caractere = %c \n", m);
      printf("(code ascii=%d)\n", m);
      return 0;
}
```

- %c : format d'entrée-sortie pour les *char*
- %d : représentation *int* du *char*
- code ASCII d'un *char* : nombre entier de 0 à 255

A l'exécution il renvoie

```
> ./Prog.exe
Entrer un caractere ? b
caractere = b
(code ascii=98)
>
```

L'utilisateur a tapé sur la touche ***b***

puis la touche **Entrée.**

Les entrées/sorties

Le type *char*

❑ Entrée de 2 caractères

Testons le programme suivant

```
int main( ) {
        char m, n;
        printf("Entrer un caractere  ? ");
        scanf("%c", &m);
        printf("caractere 1 = %c (code ascii=%d)\n", m,m);
        printf("Entrer un deuxieme caractere  ? ");
        scanf("%c", &n);
        printf("caractere 2 = %c  (code ascii=%d)\n", n,n);
        return 0;
}
```

A l'exécution il renvoie

```
> ./Prog.exe
Entrer un caractere  ? a
caractere 1 = a (code ascii=97)
Entrer un deuxieme caractere  ? caractere 2 =
  (code ascii=10)
>
```

Ce programme ne marche pas.

Mais pourquoi ?

Les entrées/sorties

Le type *char*

❑ Entrée de 2 caractères

Testons le programme suivant

```
int main( ) {
        char m, n;
        printf("Entrer un caractere  ? ");
        scanf("%c", &m);
        printf("caractere 1 = %c (code ascii=%d)\n", m,m);
        printf("Entrer un deuxieme caractere  ? ");
        scanf("%c", &n);
        printf("caractere 2 = %c  (code ascii=%d)\n", n,n);
        return 0;
}
```

A l'exécution il renvoie

```
> ./Prog.exe
Entrer un caractere  ? a
caractere 1 = a (code ascii=97)
Entrer un deuxieme caractere  ? caractere 2 =
  (code ascii=10)
>
```

- Le scanf avec %c lit **tous les caractères** tapés au clavier.

- Ici l'utilisateur a tapé sur la touche a puis sur la touche Entrée.

La touche Entrée correspond au caractère « saut de ligne » '\n' de code ascii = 10

Les entrées/sorties

Le type *char*

```c
int main ( ){
  char m, n, temp;
  printf("Entrer un caractere?");
  scanf("%c", &m);
  scanf("%c", &temp);
  printf("caractere 1 = %c (code ascii = %d) \n" , m,m);
  printf("Entrer un deuxième caractere?");
  scanf ("%c", &n);
  printf("caractere 2 = %c (code ascii = %d) \n" , n,n);
  return 0;
}
```

A l'exécution, il renvoie

```
> ./Prog.exe
Entrer un caractere?a
caractere 1 = a (code ascii = 97)
Entrer un deuxième caractere?b
caractere 2 = b (code ascii = 98)
```

ESIEE PARIS

Algorithme vs Programme - Saisie au clavier

l'algorithme

```
variable x,y,z : entier
Début
  Ecrire "Saisir deux valeurs entieres "
  Lire x
  Lire y
  z ← x + y
  Ecrire "Résulat = " z
  Saut de ligne Fin
```

Langage C

```
int main ()
  int x,y,z;
  printf("Saisir deux valeurs entiere");
  scanf("%d",&x);
  scanf("%d",&y);
  z = x + y;
  printf("Resultat = %d \n",z);
return 0;
}
```

ESIEE PARIS

Les opérateurs

Opérations arithmétiques

- `+, -, *` : Addition, soustraction, multiplication
- `/` : Division *réelle* pour les variables de type `float` et `double`
- `/` : Quotient de la division *entière* pour les variables de type `int`
- `%` : Reste de la division entière pour les variables de type `int`

ESIEE PARIS

Les opérateurs

Opérateurs d'incrémentation

- `++`, `--` : incrémentation, décrémentation

`a++;` ou `++a;` $\Longleftrightarrow$ `a = a+1;`

`a--;` ou `--a;` $\Longleftrightarrow$ `a = a-1;`

```
expression(a++); ⟺ expression(a); a=a+1;
expression(++a); ⟺ a=a+1;expression(a);
```

Les opérateurs

Opérateurs d'incrémentation

- `++`, `--` : incrémentation, décrémentation

```
a++;  ou  ++a;  ⟺  a = a+1;
a--;  ou  --a;  ⟺  a = a-1;
```

```
expression(a++); ⟺ expression(a); a=a+1;
expression(++a); ⟺ a=a+1;expression(a);
```

Opérations et affectations simultanées

- `+=` ,`-=`, `*=`, `/=` : opérateurs fusionnés

```
a += b;  ⟺  a = a + b;
a /= 2;  ⟺  a = a/2;
```

ESIEE PARIS

Les opérateurs

```
# include <stdio.h>
int main() {
   int a = 5,b,c;
   b = a++;
   c = ++a;
   printf("a = %d \n,a");
   printf("b = %d \n,b");
   printf("c = %d \n,c");
   return 0;
}
```

```
$ a = ...
$ b = ...
$ c = ...
```

Les opérateurs

Opérateurs de logiques

- ! : NON logique (négation)
- && : ET logique

  (*v1* && *v2*) est FAUX si *v1* ou *v2* est FAUX.
  est VRAI seulement si *v1* et *v2* sont VRAI
- || : OU logique

  (*v1* || *v2*) est VRAI si *v1* ou *v2* est VRAI.
  est FAUX seulement si *v1* et *v2* sont FAUX

Les opérateurs

Opérateurs de logiques

- `!` : NON logique (négation)
- `&&` : ET logique

  (*v1* `&&` *v2*) est FAUX si *v1* ou *v2* est FAUX.
  est VRAI seulement si *v1* et *v2* sont VRAI
- `||` : OU logique

  (*v1* `||` *v2*) est VRAI si *v1* ou *v2* est VRAI.
  est FAUX seulement si *v1* et *v2* sont FAUX

Opérateurs de comparaison

- `==`, `!=` : Égalité, Différence
- `<`, `>` : inférieur strict, supérieur strict
- `<=`, `>=` : inférieur ou égal, supérieur ou égal

ESIEE PARIS

Autres opérateurs

Opérateur de conversion de type (cast)

- Permet de modifier explicitement le type d'un objet

`(type) objet`

```
int i = 3, j = 2;
printf("%d \n", i/j);
printf("%f \n", (float) i/j);
```

Autres opérateurs

Opérateur de conversion de type (cast)

- Permet de modifier explicitement le type d'un objet

`(type) objet`

```
int i = 3, j = 2;
printf("%d \n", i/j);
printf("%f \n", (float) i/j);
```

Opérateur d'adresse &

- L'opérateur & appliqué à une variable retourne l'adresse mémoire de cette variable.

Les instructions conditionnelles (if, else, switch)

- La condition est une expression logique (VRAI/FAUX)
  ATTENTION : Il n'existe pas de type *booléen* en C
  Par convention :
  - 0 signifie FAUX
  - 1 ou toute autre valeur entière signifie VRAI
- On peut combiner plusieurs test avec des ET (&&), OU (||) ou utiliser la NEGATION (!)
- La partie SINON est facultative
- On peut imbriquer plusieurs instructions conditionnelles

Algorithme

```
Si (condition) Alors
    instruction(s)
Sinon
    Si (condition) Alors
        instruction(s)
    FinSi
FinSi
```

ESIEE PARIS

Les instructions conditionnelles (if, else, switch)

L'instruction IF en langage C

`if` (*condition*) *instruction* ;
Si la condition est vrai alors l'instruction est exécutée.

- Si plusieurs instructions doivent être exécutées à la suite, elles doivent être dans un bloc { }.

```
if ( condition ) {
    instruction numéro 1 ;
    instruction numéro 2 ;
}
```

ESIEE PARIS

Les instructions conditionnelles (if, else, switch)

L'instruction IF/ELSE

`if` (*condition*) *instruction1* ; `else` *instruction2* ;
Si la condition est vrai alors l'instruction 1 est exécutée sinon c'est l'instruction 2.

Exemple : Fonction qui renvoie le maximum entre deux entiers

```
int calcul_max (int a, int b){
   int maxi ;
   if (a > b) maxi = a ;
   else maxi = b ;
   return maxi ;
}
```

ESIEE PARIS

Les instructions conditionnelles (if, else, switch)

→ Lorsque que l'on souhaite conditionner l'exécution de plusieurs ensembles d'instructions par la valeur que prend une variable, plutôt que d'utiliser des structures conditionnelles imbriquées, on peut utiliser un *Selon* (un `switch` en langage C)

algortithme

Selon (identificateur)

valeur 1 : instructions
valeur 2 : instructions
...
valeur n : instructions
[autres : instructions]

FinSelon

L'instruction SWITCH

```
switch (variable){
    case valeur1 :
          liste instructions1;
          break;
    case valeur2 :
          liste instructions2;
          break;
    default:
          liste instructions3;
}
```

Les instructions conditionnelles (if, else, switch)

Une façon plus courte de faire un test

(condition) ? instruction si vrai : instruction si faux

Remarques :

- La condition doit être entre des parenthèses
- Lorsque la condition est vraie, l'instruction de gauche est exécutée
- Lorsque la condition est fausse, l'instruction de droite est exécutée

Example

```
(moyenne >=10) ? printf (" Admis ") : printf (" Ajourne ") ;
```

ESIEE PARIS

Les instructions conditionnelles (if, else, switch)

Erreurs de débutants

1. Il ne faut pas mettre de un point-virgule après un if

Example

```
int a = 0
if (a != 0);
   printf("bug : a n'est pas nul et pourtant a= %d !! \ n", a);
```

2. Il ne faut pas confondre l'opérateur $=$ (d'affectation) avec l'opérateur $==$ (comparaison d'égalité).

Example

```
int a = 0;
if (a = 0)
  printf("a = %d : a est égal à 0 \ n", a);
else
   printf("bug : a n'est pas égal à 0 et pourtant a = %d !\ n",
a);
```

Les boucles

→ Les structures répétitives permettent d'**itérer une instruction ou une suite d'instructions**. En programmation on parle de *boucle*.

Il existe 3 types de boucle en C : *while*, *do while*, *for*

1. Les instructions sont répétées **0 à n fois** (*n* **est un nombre de fois indéterminé** qui dépend uniquement de la condition de sortie de la boucle.

   TantQue `(condition)`
   Faire `instruction(s))`
   FinTantQue

→ Si la condition est fausse , alors on ne rentre pas dans la boucle

En C il s'agit de la boucle *while*

```
while (condition ) {
   instructions;
}
```

Les boucles

2. Les instructions sont répétées **1 à n fois** (*n* **est un nombre de fois indéterminé** qui dépend uniquement de la condition de sortie de la boucle)

   Répéter `instruction(s)`
   TantQue (condition)

→ On passe toujours au moins une fois dans la boucle

En C il s'agit de la boucle *do while*

```
do
{
    instructions;
}
while (condition) ;
```

ESIEE PARIS

Les boucles

3. Les instructions sont répétées **n fois** ((*n* **est un nombre de fois déterminé**)

   Pour `variable` de ... à ...
   Faire (instruction(s))
   FinPour

En C il s'agit des boucles *for*

```
for ( expression1;expression2;expression3 )
{
   instructions;
}
```

- *expression1* est évalué une fois avant d'entrée dans la boucle.
- *expression2* conditionne la poursuite de la boucle. Elle est évalué avant chaque parcours.
- *expression3* est évalué à la fin de chaque parcours.

ESIEE PARIS

Les boucles

Instruction *break*

- Elle sert à interrompre le déroulement de la boucle et en sortir prématurément.

Instruction *continue*

- Elle permet de passer prématurément au tour de boucle suivant

```
int i;
for (i = 0; i<5; i++){
    if (i==3)
        continue;
    printf("i = %d \n", i);
}
```

Un exemple - programmer la fonction factorielle

- Écrire l'algorithme pour calculer les valeurs de la fonction f

$$f(n) = 1 \times 2 \times \cdots \times n-1 \times n$$

avec

  - une boucle while
  - une boucle do while
  - une boucle for

- Puis programmer les algos en langage C

ESIEE PARIS

Un exemple - programmer la fonction factorielle

while

```
int n = 5, m = 1;
int fn = 1;
while (m < n)
{
  m = m + 1;
  fn = fn * m;
}
```

do while

```
int n = 5, m = 1;
int fn = 1;
do
{
  m = m + 1;
  fn = fn * m;
} while (m < n);
```

for

```
int n = 5, m;
int fn = 1;
for (m = 2; m < n; m++)
{
  fn = fn * m;
}
```

La structure d'un programme

- Décomposer un problème en plusieurs sous-problèmes
  - Ceci conduit souvent à diminuer la compléxité d'un problème et permet de le résoudre plus facilement
- Éviter de répéter plusieurs fois les mêmes lignes de code
  - Ceci facilite la résolutions des bugs mais aussi le processus de maintenance
- Généraliser certaines parties de programme
  - La décomposition en module permet de constituer des sous-programmes réutilisables dans d'autres contextes

La structure d'un programme

→ Dans le cas d'une approche fonctionnelle, un programme n'est plus une simple séquence d'instructions mais est constitué de :

- un ensemble de **sous-programmes** et
- un **programme principal** : UNIQUE ET OBLIGATOIRE

L'exécution du programme commence par l'exécution du programme principal.

- L'appel à un sous programme permet de déclencher son exécution, en interrompant le déroulement séquentiel des instructions du programme principal
- Le déroulement des instructions du programme reprend dès que le sous programme est terminer, à l'instruction qui suit l'appel.

ESIEE PARIS

La structure d'un programme

→ On distingue deux types de sous-programmes :

- Les **fonctions**
  - Sous - programme qui retourne **une et une seule valeur** : permet de récupérer un résultat.
  - Par convention, ce type de sous-programme ne devrait pas interagir avec l'environnement (écran, utilisateur)
- Les **procédures**
  - Sous-programme qui permet de récupérer de **0 à n résultats**
  - Par convention, ce type de sous-programme peut interagir avec l'environnement (écran, utilisateur)

ATTENTION : la distinction dans la syntaxe ne se retrouve pas dans tous les langages.
Par exemple, le C n'admet que le concept de fonction qui serviront à la fois pour les fonctions et les procédures.

ESIEE PARIS

La structure d'un programme C - Fichiers *sources* et *d'en-tête*

→ Le programme est découpé en un *programme principal* et des *sous programmes* qui sont appelés *fonctions* ou *procédures*

Programme principal (ou *main*)

[Directives au préprocesseur]
[Définition des Sous-programmes]

```
int main() {
    Déclarations de variables internes
    Instructions
    return 0;
}
```

'Fonctions' (renvoie un résultat)

```
type ma_fonction ( arguments ) {
    Déclarations de variables internes
    Instructions
    return ...;
}
```

ESIEE PARIS

La structure d'un programme C - Fichiers *sources* et *d'en-tête*

→ Le programme est découpé en un *programme principal* et des *sous programmes* qui sont appelés *fonctions* ou *procédures*

Programme principal (ou *main*)

[Directives au préprocesseur]
[Définition des Sous-programmes]

```
int main() {
    Déclarations de variables internes
    Instructions
    return 0;
}
```

'Fonctions' (renvoie un résultat)

```
type ma_fonction ( arguments ) {
    Déclarations de variables internes
    Instructions
    return ...;
}
```

'Procédures' (ne renvoie rien)

```
void ma_procedure ( arguments ) {
    Déclarations de variables internes
    Instructions
}
```

ESIEE PARIS

La structure d'un programme C - Fichiers *sources* et *d'en-tête*

Un programme C se présente sous la forme d'un ou plusieurs fichiers *sources* (avec l'extension *.c*) et fichiers *d'en-tête* (avec l'extension *.h*).

Main.c

```
# include <stdio.h>
# include "somme.h"
# define PI 3.14
int main ( ) {
  int a, b, c, d ;
  a = PI;
  b = 5;
  c = somme(a,b);
  printf("%d +%d = %d",a,b,c);
  return 0;
}
```

somme.c

```
int somme (int a, int b) {
   return a + b ;
}
```

somme.h

```
int somme (int a, int b);
```

- *.c* → code *source*
  On n'inclut jamais les *.c* !
- *.h* → fichier *d'en-tête*
  On ne compile jamais les *.h* !

ESIEE PARIS

La structure d'un programme C - Fichiers *sources* et *d'en-tête*

`# ...` → directives au pré-processeur (un programme qui procède à la transformation du code avant la compilation)

Main.c

```
# include <stdio.h>
# include "somme.h"
# define PI 3.14
int main ( ) {
  int a, b, c, d ;
  a = PI;
  b = 5;
  c = somme(a,b);
  printf("%d +%d = %d",a,b,c);
}
```

- `# include <stdio.h>`
  Lien vers le fichier d'en-tête de la bibliothèque standard *stdio.h*, utilisé pour les opérations d'entrée/sortie.
- `# include "somme.h"`
  Lien vers les fichiers d'en-tête du code *source* 'somme.c'
- `# define PI 3.14`
  Déclaration d'une MACRO sans paramètre (définition d'une constante PI)

ESIEE PARIS

La structure d'un programme C - Fichiers *sources* et *d'en-tête*

Fichiers *.h* = fichiers *d'en-tête*

On ne compile jamais les fichiers .h

somme.h

```
int somme (int a, int b);
```

- définitions des fonctions

somme.h

```
# ifndef SOMME_H
# define SOMME_H
int somme (int a, int b);
# endif
```

- Pour éviter les problèmes à l'édition des liens si plusieurs fichiers *.c* contiennent le même *en-tête* (fichier *.h*)

ESIEE PARIS

Les variables dans un programme

Main.c

```
# include <stdio.h>

void affiche_nombre (float a) {
  printf("%f \n");
}

float cte_pi = 3.14;

int main ( ) {
  float a = 10.5 ;
  affiche_nombre(cte_pi);
  affiche_nombre(a);
}
```

Variables locales vs globales

- Les variables locales ne sont définies que l'intérieur du bloc où elles ont été déclarés.
- Les variables globales (à éviter autant que possible) sont déclarées à l'extérieur d'une fonction.
  Elles sont accessibles dans toutes les fonctions.

ESIEE PARIS

A retenir

Les instructions conditionnelles

- La valeur *VRAI* peut être assimilée à la valeur numérique 1 ou à toute valeur *non nulle*
- La valeur *FAUX* peut être assimilée à la valeur numérique 0
- Ne pas oublier les parenthèse lorsqu'il y a un **if**

A retenir

Les boucles

- Si le bloc d'instruction ne doit pas être exécuté si la condition est fausse, alors on utilisera une boucle *while* ou *for*.
- Si le bloc d'instructions doit être exécuté au moins une fois, alors on utilisera la boule *do - while*
- Si le nombre d'exécutions du bloc d'instructions est connu à l'avance alors on utilisera la boucle *for*
- Si le bloc d'instructions doit être exécuté aussi longtemps qu'une condition extérieure est vraie alors on utilisera la boucle *while*.

Le choix entre les boucles *for* et les boucles *while* n'est souvent qu'une question de préférence ou d'habitudes.

Questions de révision

Identificateurs

Lesquels de identificateurs ne sont pas acceptés par le langage C pour appeler des variable et pourquoi ?

âge	var1	_Moyenne_du_bac_	N°tel
lim_sup	vitesse-max	3nombres	Age
note info	heure	prix.TTC	double