

Qu'est-ce qu'un langage ?

Un compromis entre la machine (qui ne comprend que des 0 et des 1, en fait des niveaux électriques) et l'utilisateur ou programmeur.

Un moyen de donner des instructions (ordres) à la machine.

Possède une syntaxe : une grammaire extrêmement pointilleuse : la machine ne fait pas d'efforts

Possède un lexique : orthographe des mots de référence, là encore pas d'efforts

Plus ou moins souple

De niveau plus ou moins élevé.

Plusieurs familles de langages.

Pourquoi le langage C ?

Langage impératif, déclaratif :

Très diffusé : ressources importantes.

Utilisé depuis longtemps : fiable, documenté, normalisé

Portable : nombreux compilateurs

Sert de base à : JAVA, C++, C#

Cohérent

Historique du langage C :

Inspiré par les langages :

BCPL (Basic Combined Programming Language) 1967
(M.Richards)

B (amélioration de BCPL) 1970 par K.Thompson 1970

le langage B était fruste et non portable

1971 : problème de B avec le PDP-11

1972 : première version de C écrite en assembleur
(B. Kernighan/D. Ritchie)

1973 : écriture d'un compilateur C portable (A. Snyder)

Historique du langage C :

1975 : écriture de PCC (Portable C Compiler)

1987 : début de normalisation par l'IEEE

1989 : norme ANSI X3-159 (d'où le nom C ANSI)

depuis : apparitions de langages basés sur le C, souple et puissant

Présentation du langage C

Langage de bas-niveau : accès à des données que manipulent les ordinateurs

Conçu pour écrire des programmes de système d'exploitation : 90% du noyau UNIX est écrit en C

Compilateur C écrit en C !

Suffisamment général pour des applications variées : scientifiques, accès aux bases de données

langage de haut-niveau : modules (fichiers séparés)

Le C : un langage compilé

L'ordinateur (le micro-processeur) ne comprend que le langage machine, constitué d'une suite de 0 et de 1.

Traduction des ordres donnés dans d'autres langages (PASCAL, C, Basic,...)

traduction au fur et à mesure : interprétation

traduction de tout un programme : compilation

compilateur : traduit un langage de programmation en langage machine

Principe de la compilation

Le programme en C est un fichier 'texte' (.c)

transformation en un fichier .exe (entête + code)

langage C modulaire : plusieurs fichiers .c → 1 fichier .exe

production de l'exécutable en 2 étapes :

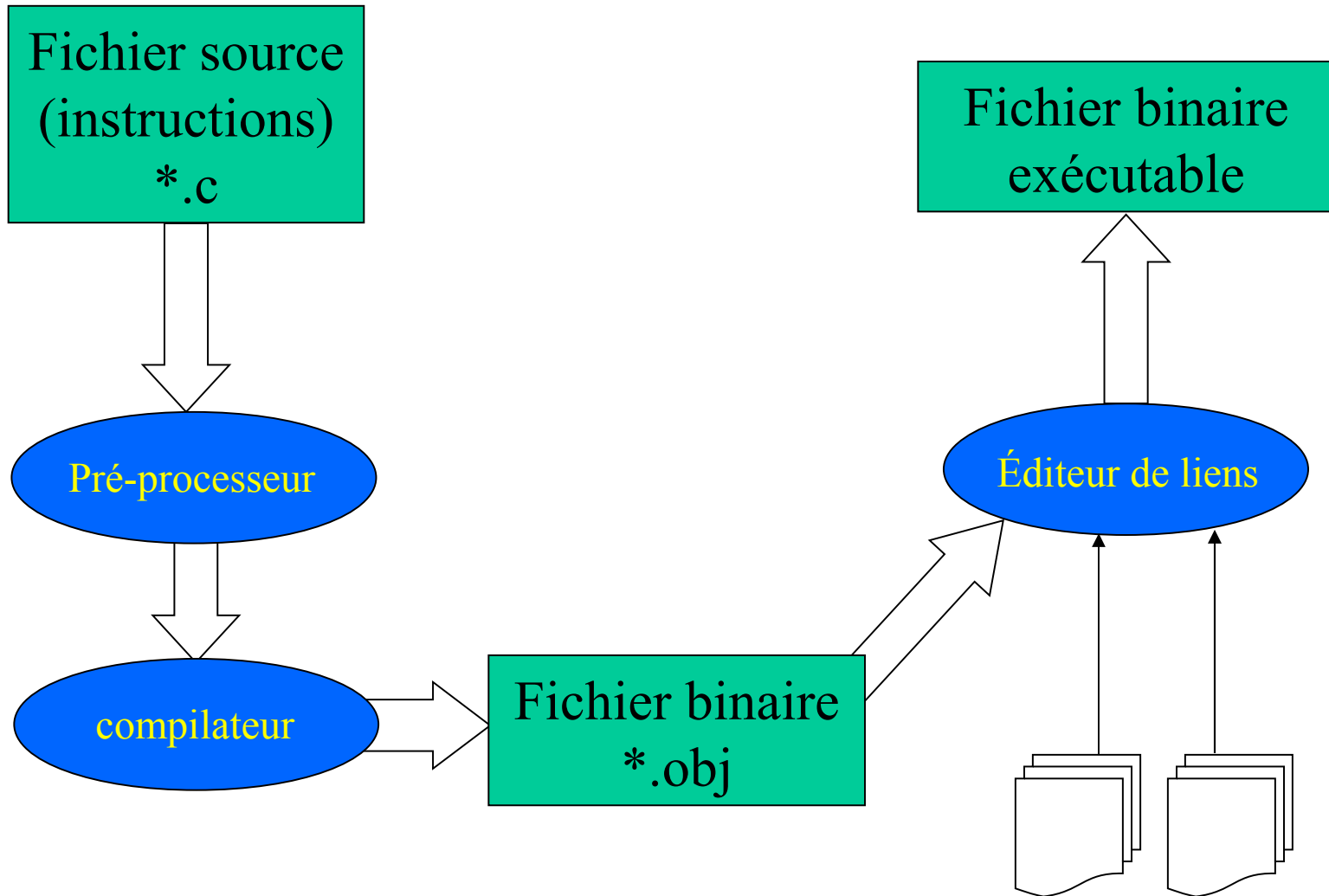
- la compilation proprement dite

génération de fichiers .o 'intermédiaires'

- l'édition des liens

réunion de fichiers .o en un exécutable .exe

Principe de la compilation



Principe de la compilation

Étapes à réaliser :

- écrire le fichier .c (environnement dédié) → **fichier .c**
- exécuter le compilateur → **fichier .o**
- si tout se passe bien, exécuter l'édition des liens → **fichier .exe**
- si tout se passe bien, on obtient un fichier exécutable

plusieurs sources d'erreur possibles :

compilation/édition des liens/exécution

L'ordinateur : machine numérique

Traite et produit des valeurs binaire : pas de sens

Représentations pour manipuler :

des nombres;

des caractères;

des données organisée.

Calcul entier exact mais limité

Calcul en nombres à virgule : valeurs approchées, précision limitée, arrondis.

Représentation et stockage

Les nombres entiers positifs : en base binaire

les nombres entiers signés : en binaire, représentation en complément à 2

les nombres à virgule : représentés selon une norme : la norme IEEE-754

les caractères : un numéro (entier) est associé à un caractère : c'est son code

Stockage de l'information

Unités de stockage :

le **bit** : vaut 0 ou 1 (transistor)

l'**octet** : regroupement de 8 bits

le **mot** : selon la machine, 2 ou 4 octets

mot court : 2 octets (16 bits)

mot long : 4 octets (32 bits)

attention aux traductions !

français

anglais

bit

bit

octet

byte

Rappel sur les bases

Utilisation d'une base pour les notations des nombres : base 10

d'autres bases sont possibles (a priori n'importe laquelle)

en base b , on dispose de b chiffres différents : de 0 à $b-1$

Stockage de l'information

Si le nombre p s'écrit, en base b , sous la forme suivante :

$a_n a_{n-1} a_{n-2} \dots a_0, a_{-1} a_{-2} a_{-m}$, où les a_i sont des chiffres de la base b , alors

sa valeur est :
$$\sum_{i=-m}^n a_i \cdot b^i$$



Valeur exprimée de manière naturelle en base 10 MAIS indépendante de la base

Éviter les confusions : on note la base en faisant suivre l'écriture de : (b)

Exemples de notations et de valeurs

$$\mathbf{2003}_{(10)} : 2 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 3 \cdot 10^0 = 2003_{(10)}$$

$$\mathbf{1,732}_{(10)} : 1 \cdot 10^0 + 7 \cdot 10^{-1} + 3 \cdot 10^{-2} + 2 \cdot 10^{-3} = 1,732_{(10)}$$

$$\mathbf{54,102}_{(7)} : 5 \cdot 7^1 + 4 \cdot 7^0 + 1 \cdot 7^{-1} + 0 \cdot 7^{-2} + 2 \cdot 7^{-3} = 5 \cdot 7 + 4 + 1/7 + 2/7^3$$

$$(\approx 39,1487_{(10)})$$

$$\mathbf{10110}_{(2)} : 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 (= 22_{(10)})$$

$$\mathbf{CAFE}_{(16)} : C \cdot 16^3 + A \cdot 16^2 + F \cdot 16^1 + E \cdot 16^0 = 12 \cdot 4096 + 10 \cdot 256 + 15 \cdot 16 + 14$$

$$(= 51966_{(10)})$$

vocabulaire de la base 10 pour exprimer la valeur des nombres

Conversions standard

Bases utilisées en informatique :

binaire ("naturel à l'ordinateur")

hexadécimal ($b=16$) : conversion facile entre binaire et hexadécimal:

un chiffre en base 16 = 4 chiffres en base 2 : traduction immédiate

exemples : $E_{(16)} = 14_{(10)} = 1110_{(2)}$

$0011_{(2)} = 3_{(10)} = 3_{(16)}$

décimal : manipulé par le programmeur / utilisateur

Méthodes de conversion

Par calculatrice : très courant

bases 2 et 16 vers base 10 :

reprise de la formule donnant la valeur :

multiplier chaque chiffre par la puissance de la base correspondante

exemples avec virgule :

$$1001,011_{(2)} = 1.2^3 + 1.2^0 + 1.2^{-2} + 1.2^{-3} = 8 + 1 + 1/4 + 1/8 = 9,375_{(10)}$$

$$F5,0A_{(16)} = F.16^1 + 5.16^0 + A.16^{-2} = 245,00390625_{(10)}$$

Méthodes de conversion

Base 10 vers bases 2 et 16 : divisions successives par la base

lecture des restes de bas en haut : tableau de divisions

Nombre à diviser	base	quotient	reste
155	2	77	1
77	2	38	1
38	2	19	0
19	2	9	1
9	2	4	1
4	2	2	0
2	2	1	0
1	2	0	1

Arrêt quand quotient=0

Intervalles de représentation

Nombre de bits ou d'octets limités :

avec n bits : 2^n valeurs différentes

résultats ou valeurs **hors-limites** : pas forcément signalés comme erreurs

résultats parfois aberrants pour l'utilisateur mais logiques pour l'ordinateur !

exemple : $128+200 = 72$ (sous certaines conditions)

Toujours prendre des précautions pour interpréter les résultats !

Les types

Un langage manipule des données :

le langage C a des données dites typées :

les valeurs manipulées sont physiquement des 0 et des 1, mais elles sont regroupées : significations différentes

- nombres entiers
- nombres à virgule
- caractères

chaque valeur peut avoir un traitement différent : procédés de calcul, affichage, saisie

Les types de base

Le type **char** :

codé sur 1 octet, 255 valeurs différentes

intervalle : de -128 à +127 ou de 0 à 255

utilisé pour les caractères ou pour les 'petits' nombres entiers

le type **int** :

codage variable en taille, généralement 2 octets (65536 valeurs différentes)

intervalle : de -32768 à +32767, ou de 0 à +65535

utilisé pour les nombres entiers

Les types de base

Le type **float** : représentation de nombres en virgule flottante (dits nombres réels mais limitations !)

codé sur 4 octets selon la norme IEEE-754

forme : mantisse, exposant (notation scientifique)

intervalle : de -10^{38} à -10^{-38} et de 10^{-38} à 10^{38}

le type **double** : extension du type **float**, codé sur 8 octets

intervalle : de -10^{308} à -10^{-308} et de 10^{-308} à 10^{308}

pour ces types, le signe est compris dans la représentation

Les types de base

Les modificateurs : à mettre devant le type

signed : types *int* et *char* : indique que les valeurs peuvent être positives ou négatives : cas par défaut

unsigned : types *int* et *char* : valeurs positives seulement

short : type *int* : force le stockage sur 2 octets

long : type *int* : force le stockage sur 4 octets

type *double* : stockage sur 10 octets (précision étendue)

peuvent être cumulés

Les types entiers : résumé

nom du type	taille en octets	intervalle
(signed) char	1	[-128;+127]
unsigned char	1	[0;255]
(signed) short int	2	[-32768;+32767]
unsigned short int	2	[0;65535]
(signed) int	2 ou 4, selon la machine	
unsigned int	2 ou 4, selon la machine	
(signed) long int	4	[$\approx -2.10^6$; $\approx 2.10^6$]
unsigned long int	4	[0; $\approx +4.10^6$]

Les types flottants : résumé

Nom du type	taille en octets	intervalle
float	4	$[-10^{38}; +10^{38}]$
double	8	$[-10^{308}; +10^{308}]$
long double	10	$[-10^{4932}; +10^{4932}]$

Remarque sur le type char

Codage d'un caractère par un nombre entier : référence à une table pour savoir ce qui doit être affiché : code ASCII

	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
<i>0</i>	NUL	DLC	SP	0	@	P	"	p
<i>1</i>	SOH	DLC1	!	1	A	Q	a	q
<i>2</i>	STX	DLC2	"	2	B	R	b	r
<i>3</i>	ETX	DLC3	#	3	C	S	c	s
<i>4</i>	EOT	DLC4	\$	4	D	T	d	t
<i>5</i>	ENQ	NAK	%	5	E	U	e	u
<i>6</i>	ACK	SYN	&	6	F	V	f	v
<i>7</i>	BEL	ETB	'	7	G	W	g	w
<i>8</i>	BS	CAN	(8	H	X	h	x
<i>9</i>	HT	EM)	9	I	Y	i	y
<i>a</i>	LF	SUB	*	:	J	Z	j	z
<i>b</i>	VT	ESC	+	;	K	[k	{
<i>c</i>	FF	FS	,	<	L	\	l	
<i>d</i>	CR	GS	-	=	M]	m	}
<i>e</i>	SO	RS	.	>	N	^	n	~
<i>f</i>	SI	US	/	?	O	_	o	DEL

Les types de base

Le type **void** : type particulier, signifiant 'rien'.

Utile pour écrire les premiers programmes

vu en détail plus tard

Syntaxe et premier programme

L'ordinateur ne traduit pas des intentions, mais des ordres :

domaine de l'utilisateur \neq domaine de l'ordinateur

capacités de l'utilisateur \neq capacités de l'ordinateur

➔ savoir comment lui donner des ordres pour qu'il soit efficace

langage C : compromis entre les utilisateur et ordinateur

nécessite une **syntaxe très stricte** (pas d'ambiguïtés)

Syntaxe et premier programme

Mots réservés : ont une signification pour le langage

- certaines instructions
- noms et extensions de type (déjà abordés)

autres noms utilisés :

- identificateurs (noms des variables et fonctions, vus plus tard)
- constantes

ne pas utiliser les mots réservés pour les identificateurs !

Syntaxe et premier programme

Qu'est-ce qu'une instruction simple ?

Ordre donné à l'ordinateur et qui provoque une action : affichage, saisie ou calcul

une instruction peut être un mot réservé (mais pas forcément)

au niveau syntaxique :

une ligne ne doit comporter qu'une instruction (lisibilité)

une instruction doit toujours être suivie d'un point-virgule ;

non-respect de cette règle : ERREUR DE SYNTAXE

Exemples d'instructions simples

Voici des instructions simples et leur rôle

<code>printf("coucou");</code>	affiche le mot "coucou"
<code>x=y+2;</code>	calcule $y+2$ (y a une valeur) et range le résultat de ce calcul dans x
<code>exit();</code>	provoque la fin du programme

Syntaxe et premier programme

Qu'est-ce qu'une instruction composée ?

C'est un **ensemble de plusieurs instructions** (chacune d'elles est simple ou composée)

au niveau syntaxique :

une instruction composée est encadrée par des accolades {}

chacune des instructions qui la composent doit être correcte au niveau syntaxique

on appelle aussi une instruction composée un **bloc d'instructions**

Syntaxe et premier programme

Le premier programme : présentation de la syntaxe

```
#include <stdio.h>

void main ()

{

    printf ("Bonjour\n");

}
```

Le rôle de ce programme est d'afficher le mot "Bonjour" à l'écran

premier programme :explications

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

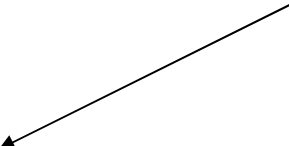
```
void main ()
```

```
{
```

```
printf ("Bonjour\n");
```

```
}
```

Ceci est une instruction, qui décrit ce que le programme doit faire



premier programme :explications

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

```
void main ()
```

```
{
```

```
    printf ("Bonjour\n");
```

```
}
```

← Directive permettant d'utiliser printf. Ne fait aucune action en elle-même.

premier programme :explications

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

```
void main ()
```

```
{
```

```
printf ("Bonjour\n");
```

```
}
```

Éléments indiquant le début et la fin du programme. Ne font pas d'action.

résumé

**Pour l'instant, un
programme aura
toujours cette forme**

```
#include <stdio.h>
void main ()
{
```

description des actions à mener pour réaliser
le programme

```
}
```

Les commentaires

Lignes ignorées par le compilateur

utiles pour le programmeur : noter ce qui est fait

2 syntaxes :

C: encadrer le commentaire par /* (début) et */ (fin du commentaire)

C++ : faire débiter la ligne par //

la majorité des compilateurs comprend les 2 syntaxes

il faut en mettre le plus possible sans surcharger

Les variables

Boîtes ou cellules contenant une **valeur**

accessibles par leur **nom**

contiennent un certain **type** de valeur

physiquement situées dans la mémoire vive (RAM) de l'ordinateur, à une certaine **adresse**

on peut les : lire (consulter)

écrire (modifier)

Les variables

Toute valeur susceptible d'être modifiée (de varier) doit être stockée dans une variable.

Pour manipuler une variable, on est obligé de préciser ce qu'elle est pour que le langage puisse la traiter correctement.

C'est le rôle d'une **déclaration de variable** :

présentation des éléments indispensables au compilateur :

- le nom de la variable
- le type de la variable

Les variables : déclaration

Le nom : c'est un identificateur à choisir. Moyen facile de repérer la variable.

Au niveau syntaxique :

- un nom de variable doit commencer par une lettre ou '_' (underscore)
- il y a 31 caractères significatifs
- différence minuscules/majuscules
- il peut contenir des chiffres, des lettres, et '_'

Les variables : déclaration

Exemples de noms de variables et leur validité

une_variable correct

toto correct

x9 correct

6060n incorrect : commence par un chiffre

une-variable incorrect : contient le caractère '-'

ceciEstUnIdentificateurDeVariableA43Lettres correct

ceciEstUnIdentificateurDeVariableAvec46Lettres correct

ATTENTION : pas de différence entre les deux derniers exemples !

Les variables : déclaration

Le type : permet au langage C de savoir comment coder les valeurs de la variable et comment les traiter.

Rappel : tailles différentes !

Au niveau syntaxique : on utilise les noms de types déjà vus : char, int, float, double, void et les modificateurs adéquats.

La déclaration de variable se fait de la manière suivante :

type de la variable **nom de la variable;**

ne pas oublier le ;

Les variables : déclaration

Exemples et significations :

int	unNombre;	unNombre est une variable de type int
char	x;	x est une variable de type char
float	valeur_numero_4;	valeur_numero_4 est une variable de type float
short int	toto;	toto est une variable de type short int
void	riendutout;	

Les variables : déclaration

Dans un programme, on ne peut utiliser que des variables préalablement déclarées.(erreur sinon)

Rôle et placement dans le programme :

une **déclaration de variable n'est pas une instruction**, elle n'effectue pas d'action visible ou sensible.

Ce que fait le compilateur : il utilise dans la mémoire une boîte pour stocker la valeur de la variable, et repère son adresse (là où elle se situe). Le nom est associé à cette adresse pour faciliter l'écriture.

Les variables : déclaration

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

```
void main ()
```

```
{
```

```
    déclarations de toutes les variables
```

```
    instructions du programme
```

```
}
```

Les variables : déclaration

En pratique :

on ne connaît pas toujours toutes les variables au début du programme : on peut les y ajouter au fur et à mesure si on en a besoin.

Choix des noms de variables : 1 seule lettre ? Plusieurs ?

Problème de clarté : choisir des noms significatifs : facilite la relecture, ne change rien au programme.

Exemple : programme avec déclaration

```
#include <stdio.h>
void main()
{
    int valeur1;
    float x_1;
    float x_2;
}
```

Syntaxiquement correct (le compilateur accepte), ne fait rien.

Déclarations abrégées

Écriture concise pour déclarer plusieurs variables de même type :
au lieu d'une variable : liste de variables séparées par des virgules,
terminée par un point-virgule.

Exemple :

au lieu de :

```
int a;
```

```
int b;
```

```
int c;
```

écrire :

```
int a,b,c;
```

Opérations de base

Un peu de pratique avec des exemples :

Affichages à l'écran

Saisies

Manipulations et opérateurs de base

Quelques subtilités

Opérations de base

Les affichages : utilisation de *printf*

qu'est-ce que printf ?

Ce n'est pas un mot réservé : dépend du système

C'est une fonction (vu plus tard) qui permet de réaliser un affichage à l'écran.

Capable d'afficher les valeurs des variables (pas leur nom !)

syntaxe générale :

```
printf(texte_formaté, liste_de_variables...);
```

Opérations de base

Qu'est-ce que le texte formaté ?

C'est un texte contenant des mots ou caractères à afficher :
caractères normaux (lettres) et spéciaux (mise en page)

basé sur le codage ASCII : codes pour quelques caractères spéciaux

`\n` : passage à la ligne

`\a` : bip

`\t` : tabulation horizontale

`\v` : tabulation verticale

un texte doit se trouver entre double guillemets " "

Opérations de base

Exemples : textes simples

```
printf("Bonjour");
```

affiche le texte `Bonjour` à l'écran

```
printf("\tSalut");
```

affiche une tabulation horizontale puis le texte `Salut`

```
printf("\a\a\nhello \n");
```

'affiche' deux bips, un passage à la ligne, le texte `hello`, puis un autre passage à la ligne

Opérations de base

Exemples : textes formatés, affichage de variables

pour afficher une variable : ne pas donner son nom dans le texte

exemple de programme pour afficher une variable

```
#include <stdio.h>
void main()
{
    int uneValeur;          /* declaration de valeur */

    uneValeur = 5;         /* affectation de la variable */
    printf("uneValeur");  /* on veut afficher '5' */
}
```

Opérations de base

Le programme précédent affiche le texte `uneValeur`.

Utiliser un code de format dans la chaîne pour préciser que l'on veut afficher une valeur, le type de la valeur est précisé.

Code	type de la valeur à afficher
<code>%d</code>	entier signé
<code>%c</code>	caractère
<code>%u</code>	entier non signé
<code>%e</code>	nombre à virgule - exposant
<code>%f</code>	nombre à virgule float
<code>%lf</code>	nombre à virgule double
<code>%g</code>	nombre à virgule : meilleure
<code>%s</code>	chaîne de caractères

Opérations de base

La chaîne contient le texte et les formats, la liste des valeurs à afficher est précisée à la suite de la chaîne.

Exemple : reprise du programme précédent

uneValeur est un entier signé : format %d mis dans la chaîne

```
#include <stdio.h>
void main()
{
    int uneValeur;          /* declaration de valeur */

    uneValeur = 5;         /* affectation de la variable */
    printf("%d",uneValeur); /* on veut afficher '5' */
}
```


Opérations de base

`printf("%d", uneValeur);` se lit alors :

le format de la valeur à afficher est `%d`

ou

affichage d'un entier signé dont la valeur est donnée par `uneValeur`.

On peut mélanger texte et formats :

On peut afficher plusieurs valeurs avec un seul `printf`.

On peut faire des affichages complexes avec un seul `printf` en utilisant les différents codes de mise en page : pas forcément la meilleure solution.

Opérations de base

Texte et formats : exemple de programme

```
#include <stdio.h>
void main()
{
    double nbDouble;

    nbDouble=5.14E+8;

    printf("le nombre vaut %lf\n",nbDouble);
}
```

résultat :

le nombre vaut 514000000.00000

Opérations de base

Texte et plusieurs formats : exemple de programme

```
#include <stdio.h>
void main
{
    char carac;
    double valD;

    carac='H';
    valD = 3.14159;

    printf("deux valeurs : %c et %lf\n",carac,valD);
}
```

résultat:

deux valeurs : H et 3.141590000

Opérations de base

Texte et plusieurs formats : exemple de programme

les valeurs doivent être listées dans le même ordre que les formats correspondants.

Opérations de base

Affichage complexe : exemple

```
#include <stdio.h>
void main
{
    char carac;
    double valD;
    int    toto;

    carac='x';
    valD = 3.14159;
    toto = -6;

    printf("\ttrois valeurs\n\tun entier : %d\n\tun char :
%c\n\tet enfin...\v\tun double : %lf\n",toto,carac,valD);
}
```

attention à \v, ne marche pas avec tous les compilateurs !

Ligne obtenue difficile à lire... pas une bonne chose !

Opérations de base

Saisie de valeurs : emploi de *scanf*

comme printf, scanf est une fonction qui permet de faire des saisies, en utilisant aussi des formats pour interpréter ce qui est saisi.

Emploi légèrement différent : pas de texte, seulement une chaîne avec des formats.

En fait, on n'utilise qu'un format par scanf

saisie d'une valeur \Leftrightarrow un scanf

possibilité de saisir plusieurs valeurs dans un scanf : emploi pas évident, source de confusion.

Opérations de base

Saisie de valeurs : emploi de *scanf*

scanf s'emploie de la manière suivante :

```
scanf(chaine_de_format, adresse_de_la_variable);
```

différent de printf : variables, ici adresses.

Faire précéder la variable dont on veut saisir la valeur de l'opérateur **&**.

Les codes pour les formats sont les mêmes que pour printf (%d, %u, %f,...)

Opérations de base

Exemples de saisies avec scanf : différents types

```
#include <stdio.h>

void main()
{
    unsigned int value_1;
    char unCaractere;
    double nbVirgule;
    int x;

    /* 4 saisies séparées pour les 4 valeurs */

    scanf("%u", &value_1);
    scanf("%c", &unCaractere);
    scanf("%lf", &nbVirgule);
    scanf("%d", &x);

    printf("%u %c %lf %d\n", value_1, unCaractere,
    nbVirgule, x);
}
```


Opérations de base

Effet de scanf : arrêt du programme en cours, attente d'une saisie : entrer la valeur souhaitée et valider avec entrée.

scanf n'affiche pas de message : le faire précéder d'un printf pour que l'utilisateur sache ce qu'il doit faire (pas d'écran noir sans instructions, recommandé, pas obligatoire)

ne pas inclure de codes de mise en page dans le scanf, notamment pas de `\n` !!!

Nécessitera d'appuyer 2 fois sur entrée : une fois pour terminer le format du scanf, une fois pour valider.

Opérations de base

Saisie propre : en fait, les saisies au clavier ne sont pas transmises directement au programme : existence d'un buffer (tampon), qui stocke des valeurs et les délivre de temps en temps. Avant saisie, il faut vider ce buffer pour éviter les comportements 'bizarres' des saisies : utiliser l'instruction `fflush(stdin);`

Les expressions

Une expression est une valeur qui peut être calculée par l'ordinateur, qui contient des termes et des opérateurs reliés par une syntaxe précise.

Plus simplement : une suite de symboles qui est comprise par le langage et qui donne un résultat.

Déjà connu : les expressions mathématiques qui donnent une valeur

$1+2\times 4-(5/6)$ est une expression mathématique valide et calculable, qui vaut : $49/6$

$!21+ \times \times 2$ n'est pas une expression, bien que constituée de symboles tous connus : ne respecte pas une syntaxe donnée.

Les expressions

En C : même principe : règles de syntaxe et de priorités pour calculer la valeur d'une expression.

De plus, une expression a un type : sa valeur calculée sera utilisée par le langage, donc doit avoir un type.

Termes possibles dans une expression :

les variables peuvent être utilisées (puisqu'elles ont un type et une valeur)

des constantes numériques peuvent être utilisées

Opérateurs : nous en verrons tout un ensemble.

Constantes numériques

Constantes entières : nombre sans virgule, précédé ou non du signe

-

exemples : 3 0 824 -12000

constantes caractère : les caractères sont des valeurs entières, on peut leur attribuer une valeur comme pour les entiers ci-dessus, mais aussi directement un caractère en l'encadrant de simples guillemets : la valeur sera alors le code ASCII du caractère

exemples : 'a' 23 'H' ':'

constantes à virgule : un nombre écrit sous la forme classique ou scientifique (avec exposant) : partie entière.ppartie décimaleEexposant

exemples : 3.14 0.0001 1.27E+32 65634.23476432E-5

Constantes numériques

Pour un nombre à virgule n'ayant pas de partie décimale : noter juste le .

Exemple : 1. -8. Le point indique que ce n'est pas un entier.

Opérateurs

Arité (cardinalité) des opérateurs : nombre de termes (ou opérandes)

Opérateurs mathématiques et priorités :

opérations classiques 2-aires :

addition : + soustraction : - multiplication : * division : /

* et / prioritaires sur + et -

modulo 2-aire : %

$a \% b$ donne le reste de la division entière de a par b .

ajout de parenthèses possible pour gérer les priorités : à employer sans modération : ne change pas le programme, facile à lire !

Opérateur d'affectation

Opérateur =

cet opérateur a un rôle particulier.

À sa **gauche**, on doit trouver **une variable obligatoirement** (et pas une expression complexe)

À sa **droite**, on trouve une **expression**.

Rôle : calcule l'expression à droite de l'opérateur = et donne cette valeur à la variable à gauche de cet opérateur.

Peut sembler évident, mais attention !

Opérateur d'affectation

```
void main()  
{  
    int    a,b;  
    double d,e;  
  
    /* toujours la même chose : calcul de l'expression */  
    /* à droite, affectation à la variable de gauche */  
  
    a=4;  
    b=a+2;  
    a=b*5+a/2-17;  
  
    d= 6.02252E+23;  
    e=5.*d;  
  
    /* curiosité ? */  
    b = b-1;  
    e = e/1000.;  
}
```

Opérateur d'affectation

Une affectation n'est pas une équation !

Lorsque l'on écrit $b = b-1$; (d'ailleurs l'équation n'aurait pas de solutions !), on calcule la valeur de $b-1$, puis on range cette valeur dans b .

retour sur les trois premières lignes du programme, pour exemple :

<code>a=4;</code>	l'expression vaut 4 : a reçoit cette valeur
<code>b=a+2;</code>	calcul de $a(4) + 2$: l'expression vaut 6, b reçoit 6
<code>b=b-1;</code>	calcul de $b(6) - 1$, l'expression vaut 5, b reçoit 5

Opérateur d'affectation

Une affectation est elle-même une expression ! Sa valeur est égale à la valeur calculée et donnée à la variable de gauche.

Reprise des exemples précédents :

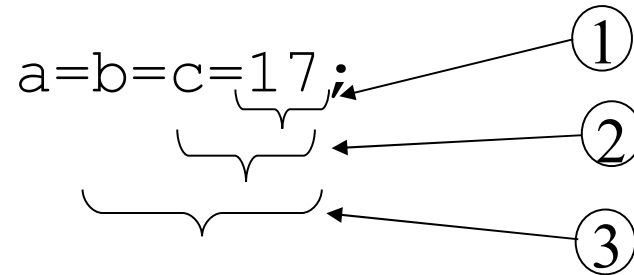
`a=4;` est une expression qui vaut 4
`b=a+2;` est une expression qui vaut 6
`b=b-1;` est une expression qui vaut 5

possibilité d'affectations multiples du type :

```
void main()  
{  
    int a,b,c;  
  
    a=b=c=17;  
}
```

Opérateur d'affectation

Expression calculée (ou évaluée) de droite à gauche :



étape ①: évaluation de 17, rangement dans la variable c;

étape ②: $c=17$ est une expression qui vaut 17, rangée dans b par :
 $b=c=17;$

étape ③: $b=c=17$ est une expression qui vaut 17, rangée dans a par
 $a=b=c=17;$

étape 4 : $a=b=c=17$ est une expression qui vaut 17, qui n'est pas rangée.

Opérateurs composés

Combinaison d'opérateurs mathématiques et d'affectation :

permet d'appliquer un opérateur mathématique à une variable puis d'affecter le résultat de cette opération à la variable.

Opérateurs :

addition et affectation : +=

soustraction et affectation : -=

multiplication et affectation : *=

division et affectation: /=

emploi par l'exemple :

$x += 4;$ signifie $x = x + 4;$

$e *= d + 5.1;$ signifie $e = e * (d + 5.1);$

$y /= x;$ signifie $y = y / x;$

et ainsi de suite

Changer le type d'une expression

Les expressions peuvent contenir des termes qui sont de type différents : le compilateur accepte ces expressions (elles sont valides) mais effectue des changements de type pour les valeurs.

Perte de précision possible (le compilateur émet alors des warnings)

cette opération s'appelle **cast** ou **transtypage**.

Exemple :

```
int a,b;
double x;
a=5;
x=3.1415926;
b=a+x; /* l'expression est affectée à un int ! */
```

```
warning C4244: '=' : conversion from 'double' to 'int',
possible loss of data
```

Changer le type d'une expression

Une opération peut ne pas donner le résultat escompté :

exemple classique, lorsque l'on divise 2 entiers entre eux : division entière même si le résultat est stocké dans un double !

Le transtypage est effectué au fur et à mesure de l'évaluation de l'expression, pas en analysant l'expression !

Exemple :

```
int a=5;  
int b=2;  
double x=a/b;  
printf("x=%lf\n", x) affichera 2.0000 !
```

a/b est une division entière, résultat 2, converti en double ensuite !

Changer le type d'une expression

Solution : effectuer soit même les transtypages pour être certain du type des valeurs !

Pour transtyper une expression d'un type en un autre, il suffit de noter entre parenthèses le type souhaité devant l'expression :

(type_souhaité)expression.

Soit i un entier : $(\text{double})i$ donne la même valeur mais en type double

soit x un double : $(\text{int})x$ transtypera x en int, en supprimant les chiffres après la virgule

Changer le type d'une expression

Exemple pour faire la division correctement : il faut transtyper a et b individuellement en double !

Si l'on écrit : $x=(\text{double})(a/b);$ ne change rien

$x=(\text{double})a/(\text{double})b;$ on aura le résultat souhaité