Plan

Rappel: programme minimal

Un programme C se compose d'une fonction **main**, dont la version la plus simple est:

```
1 int main(){
2     // declaration de variables
3     // instructions du programme
4 return 0;
5 }
```

Rappel: programme minimal

Comme dans un langage algorithmique :

- les expressions et variables ont un type (ex: entier, flottant)
- les instructions sont des expressions particulières modifiant l'environnement :
  - des affectations
  - des structures de contrôle (ex: tant que)

Les composants de base d'un programme C

Un programme C est constitué des composants élémentaires suivants :

- les types de données,
- les constantes,
- les variables,
- les opérateurs,
- les structures de contrôle,
- les appels de fonctions

Plan

Les types de base

- entiers : des parties finies `char`, `short`, `int`, `long`
- réels : des approximations bornées : `float`, `double`
- booléens : `_Bool`

Des conversions implicites d'un type à l'autre.

Le codage binaire des entiers

Les entiers sont exprimés sous format binaire

$103 = 64 + 32 + 4 + 2 + 1 = 2^6 + 2^5 + 2^2 + 2^1 + 2^0$

1	1	0	0	1	1	1

Le nombre de bit (case) fixe l'ensemble des entiers représentables.

Sur 7 bits on représente les entiers positifs dans $[0, 127]$

- le plus grand: $127 = 2^7 - 1$

1	1	1	1	1	1	1

- le plus petit: 0

0	0	0	0	0	0	0

Le codage binaire des entiers

Les entiers négatifs sont également représentés par un codage binaire: le codage en complément à deux

Complément à deux

$$-103 = -128 + 25 = -128 + 16 + 8 + 1 = -2^7 + 2^4 + 2^3 + 2^0$$

1	0	0	1	1	0	0	1

sur 8 bits

$$-103 = -256 + 128 + 16 + 8 + 1 = -2^8 + 2^7 + 2^4 + 2^3 + 2^0$$

1	1	0	0	1	1	0	0	1

sur 9 bits

On dit que le bit de poids fort (le plus à gauche) est le bit de signe.
$\hookrightarrow$ il a un poids négatif si non nul.

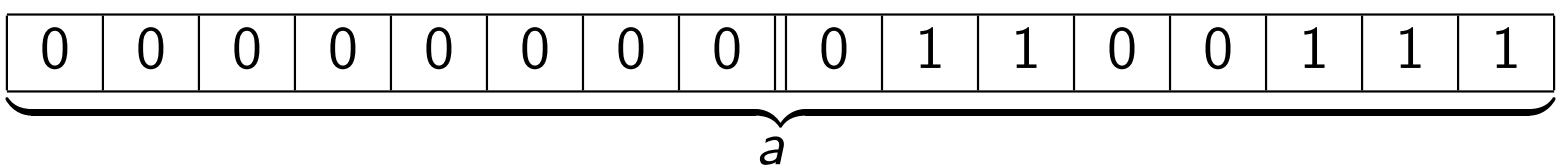
Les types de données entiers

Plusieurs variantes (complément à deux) qui utilisent plus ou moins d'octets et qui ont différentes plages de valeurs.

type en C	nbr octets	valeurs
`char`	≥ 1	$[-2^7, 2^7 - 1]$
`short`	≥ 2	$[-2^{15}, 2^{15} - 1]$
`int`	≥ 4	$[-2^{31}, 2^{31} - 1]$
`long`	≥ 8	$[-2^{63}, 2^{63} - 1]$

Exemple

```
short int a= 103;
```



Les types de données entiers

On peut étendre la valeur maximum des entiers en utilisant des entiers positifs (versions non signées = pas de complément à deux)

type en C	**nbr octets**	**valeurs**
`unsigned char`	≥ 1	$[0, 2^8 - 1]$
`unsigned short`	≥ 2	$[0, 2^{16} - 1]$
`unsigned int`	≥ 4	$[0, 2^{32} - 1]$
`unsigned long`	≥ 8	$[0, 2^{64} - 1]$

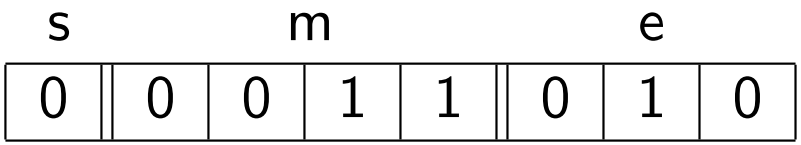
Les types de données réels

Les réels ne sont pas représentables en codage binaire. On représente un réel x par une approximation rationnelle particulière:

$$x \approx (-1)^s \times \frac{m}{2^e}$$

exemple: $0.75 = (-1)^0 \times \frac{3}{2^2}$

Le codage de x correspond au codage binaire de s,m et e.



Norme IEEE754 :

- `float` (32 bits): 1 bit pour s, 23 bits pour m, 8 bits pour e
- `double` (64 bits): 1 bit pour s, 52 bits pour m, 11 bits pour e

Les booléens

Il n'y a pas de base un type booléen en `C`. On utilise deux valeurs du type `int` pour représenter les booléens :

- 0 représente la valeur **faux**
- 1 représente la valeur **vrai**

Les règles implicites de conversion amènent à considérer que toute valeur non codée par une suite de bits à 0 représente le booléen vrai.

Remarque

Depuis la norme 99, un type `_Bool` a été introduit codé sur 1 octet et n'ayant que deux valeurs 0 et 1.

Plan

Les constantes

- Les constantes permettent de désigner des valeurs dans un programme C
- Ces valeurs sont automatiquement typées en fonction de la forme syntaxique de cette constante
- on distingue :
  - les constantes entières : de type int, unsigned int, long ou unsigned long
  - constantes réelles : de type float ou double
  - constantes caractères : de type int (et non char !)
  - constantes chaines de caractères : de type tableau de caractères
- Il n'y a pas de constantes de type char et short $\Rightarrow$ des conversions permettront de valuer les variables de ces types.

Les constantes entières

- Des valeurs positives entières données en représentation décimale, octale ou hexadécimale
  - 169, 0251 ou 0xA9 désignent la représentation machine `int` sur 4 octets [00000000|00000000|00000000|10101001]
- la valeur de la constante détermine le type de donnée de l'entier cible
  - de 0 à $2^{31} - 1 \rightarrow$ `int`
  - de 2^{31} à $2^{63} - 1 \rightarrow$ `long`
  - de 2^{63} à $2^{64} - 1 \rightarrow$ `unsigned long`
- on peut forcer une représentation non signée et/ou long avec les suffixes U et L
  - 169UL désigne le `unsigned long` sur 8 octets dont les 56 premiers bits sont à 0 et les 8 derniers sont 10101001

Remarque

Les valeurs négatives des entiers sont obtenues par l'opérateur unaire -

Les constantes réelles (flottantes)

- Des valeurs réelles positives données en représentation scientifique
  - ex: 2.34e4 qui correspond à 2.34×10^4 (le rationnel 23400)
  - ex: 2. qui correspond à 2.0×10^0 (le rationnel 2)
  - ex: 2e-2 qui correspond à 2.0×10^{-2} (le rationnel 0,02)
  - le . ou le e est obligatoire !
- les constantes réelles sont représentées par des double.
- on peut forcer le type `float` en suffixant par F
  - ex: 2.34e4F

Les constantes caractères

Les constantes caractères sont données entre deux quotes '...'

- Elles peuvent contenir un :
  - caractère alphanumérique :
    'a', 'b', ...
    'A', 'B', ...
    '0', '1', ...
    '?', ':', ' ', '<', ...
    mais pas un caractère composé : 'é', 'ç'...
  - un caractère spécial:

    $\backslash n$ $\rightarrow$ retour à la ligne
    $\backslash t$ $\rightarrow$ tabulation
    $\backslash b$ $\rightarrow$ backspace
    $\backslash 0$ $\rightarrow$ fin de chaîne
    ...
- Elles sont représentées par des **valeurs du type `int`** : l'entier correspondant à leur code ASCII.

Codage des caractères

- Les entiers associés aux constantes caractères codent les éléments du jeu de caractères de la machine (souvent ASCII sur 7 bits).

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

les constantes chaînes de caractères

Les constantes chaînes de caractères sont des suites de caractères données entre deux doubles quotes (guillemets) "..."

- Elles contiennent n'importe quel caractère y compris les caractères composés (é, à, ç...) et les caractères spéciaux (\n, \t ...)
- Elles sont représentées par des **séquences de `char`** dont la longueur dépend du nombre et des caractères de la chaîne
  - un caractère alphanumérique simple ou un caractère spécial est représenté par un `char`
  - un caractère composé ou spécifique à un alphabet est représenté par plusieurs **char** (selon le codage de caractères utilisé sur la machine)
  - le caractère spécial fin de chaîne \0 est ajouté à la fin
- "ça va\n" est représenté par la suite de 8 `char` (octets)
  [195|167|97|32|118|97|10|0] [ç | a | | v | a | \n | \0]

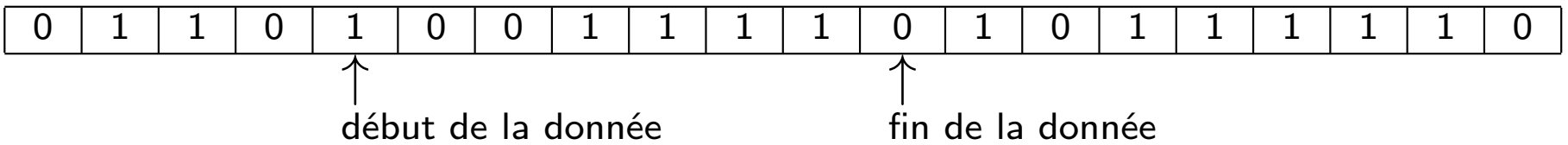
Plan

Stockage des données

Rappel:

- la mémoire est binaire (un grand tableau de 0 et de 1)
- le langage C manipule directement la mémoire

Les données sont stockées à un emplacement précis de la mémoire



Pour retrouver une donnée, il faut donc :

- une **adresse** de début de la donnée dans l'espace mémoire
- une taille indiquant l'espace mémoire occupé par la donnée

Variable C

Definition

Une variable C est une zone de la mémoire de l'ordinateur à laquelle on a donné un nom, ainsi qu'un type de données.

- le **nom** de la variable, appelé *identificateur*[1], permet au programmeur de manipuler facilement les données stockées en mémoire, mais **l'adresse** de la variable serait suffisante
- le **type** permet au compilateur de *réserver l'espace mémoire* suffisant pour stocker les valeurs.
- la suite des bits dans la zone mémoire définit la **valeur** de cette variable.

[1] on utilise un identificateur plutôt que l'adresse mais l'on peut récupérer l'adresse d'une variable à partir de son identificateur.

Les identificateurs

En plus de permettre de nommer les variables, ils permettent de nommer les fonctions et les types définis.

Construction d'un identificateur

Une suite de caractères parmi : les majuscules (A..Z), les minuscules (a..z), les chiffres (0..9), le tiret bas (_)

- le premier caractère ne peut être un chiffre,
- les majuscules et minuscules sont différenciées,
- pas de caractères accentués,
- l'identificateur doit être différents des mots clés du langage : `int`, `char`, `if`, `for`...

Déclaration des variables

Déclaration

Pour utiliser une variable il faut auparavant la déclarer en précisant son type et son nom : `type nom ;`
On peut déclarer simultanément plusieurs variables du même type en séparant les noms par des virgules : `type nom1, nom2, ... ;`

Exemple

```
int a ;
```

a est une variable **entier standard** :

- le compilateur réserve 4 octets en mémoire pour stocker ses valeurs
- le programmeur utilise le nom a pour travailler avec cette zone mémoire.

```
short b, c ;
```

b et c sont deux variables **entier court** :

- le compilateur réserve 2×2 octets en mémoire pour stocker leurs valeurs.

Plan

Les expressions

Definition

Une expression atomique est soit une constante, soit un nom de variable.
Des expressions plus complexes peuvent être construites à l'aide d'opérateurs.

- Le C est un langage typé ⇒ Toute expression est typée. Le type dépend de l'opérateur et de ses opérandes.
- Le C est un langage faiblement typé ⇒ De nombreuses conversions implicites de type rendent possible l'utilisation d'opérateurs sur des opérandes n'ayant a priori pas le type requis.

Les opérateurs arithmétiques

les opérateurs classiques binaires

- addition : $+$
- soustraction : $-$
- division : $/$
- multiplication: $*$
- modulo : $\%$
- opposé (unaire) : $-$

Attention: les opérateurs utilisent des opérandes du même type et renvoient un résultat du même type que les opérandes.

- `int + int = int;`
- `float * float = float;`
- `int + float = ???`

Opérateurs de comparaisons

Les comparateurs classiques sur les types numériques:

égalité	== (2 signes =)
différent	! =
supérieur	>
inférieur	<
supérieur ou égal	>=
inférieur ou égal 1	<=

Rappel : le langage C ne dispose pas d'un type booléen. Les comparateurs retourne une valeur du type `int` :

- 0 si l'expression est fausse (ex: $3 < 2$)
- 1 si l'expression est vraie (ex: $3 > 2$)

Opérateurs logiques

Les opérateurs logiques en C:

- et : `&&`
- ou : `||`
- non: `!`

Exemple

`!1` est évalué à 0 (faux)

`1 && 0 || 1` est évalué à 1 (vrai)

`(2.5 > 3.5) && (1 < 3)` est évalué à 0

`1 || (1 >= 3)` est évalué à 1

Opérateur d'affectation

Cette opérateur permet d'affecter la valeur d'une expression à une variable. L'affectation se fait avec =

- `var = exp`
  - `exp` : expression de **même type** que `var` (mais mécanisme de conversion implicite de type)
  - `var` : nom d'une variable déclarée
  - la variable `var` prend la valeur de l'expression `exp`
  - ex: `a=2+3;` `a` prend la valeur de l'expression 2+3 donc de 5

- comme en algorithmique, l'opérande de gauche ne peut être qu'une variable : l'affectation `a+b=3;` n'est pas correcte syntaxiquement.

Sizeof

L'opérateur unaire sizeof permet de connaitre la taille mémoire (en nombre d'octets) de son opérande qui peut être :

- un type de données : on renvoie le nombre d'octets occupés par une valeur de ce type.
- une expression : on renvoie le nombre d'octets occupés par une valeur du type de l'expression

Exemples :

- `sizeof(2)` vaut 4 (la constante 2 est de type `int`)
- soit a déclaré par : `char a ;`
  `sizeof(a)` vaut 1 (le type char réserve 1 octet)
- `sizeof(a+a)` vaut 4 (pour appliquer l'addition une conversion de la valeur de a en une valeur `int` a été faite)

Opérateur d'adresse mémoire

Chaque variable est stockée en mémoire à une adresse précise.
L'opérateur d'adresse `&` permet de récupérer l'adresse associée à une variable

Soit `a` une variable déclarée, `&a` renvoie la valeur de l'adresse de `a` :
un entier codé sur 8 octets quelque soit le type de `a`.
Sa valeur est généralement donnée en hexadécimale.

Attention

- l'adresse des variables n'est pas choisie par le programmeur: `&a=...` est interdit !!!
- l'adresse des variables peut être stockée dans une variable: `b=&a` si `b` a le bon type

Autres opérateurs

- incrémentation/décrémentation d'une variable entière a:
  - $a++$ incrémente la valeur de a par 1
  - $a--$ décrémente la valeur de a par 1
- affectations élargies: `+=`, `-=`, `*=`, `/=`
  - `a+=3` correspond à l'expression `a=a+3`
- l'opérateur conditionnel: `b ?` e_1 `:` e_2
  - selon la valeur de l'expression booléenne b, l'opérateur calcule la valeur de e_1 si b vrai, ou la valeur de e_2 si faux (e_1 et e_2 doivent être de même type). Ex : `0?-5:5` vaut 5
- les opérateurs bit à bit : `&`, `|`, `^` , `~` , `<<`, `>>`
- et beaucoup d'autres ...

Priorité des opérateurs arithmétiques

Les règles classiques de priorités mathématiques sont conservées:

- $a + b \times c$ sera interprété $a + (b \times c)$

Les parenthèses permettent d'imposer que certaines opérations soient évaluées avant d'autres :

- $(a + b) \times c$ impose de calculer l'addition avant la multiplication

Priorités des opérateurs : récapitulatif

Opération associative ($\star$):

- à droite: $a \star b \star c \Rightarrow a \star (b \star c)$
- à gauche: $a \star b \star c \Rightarrow (a \star b) \star c$

Catégorie	**Opérateurs**	**Associativité**
Unaire	$+$, $-$, $++$, $--$, !, $*$, &, sizeof, cast	Droite
Binaire	$*$, $/$, %	Gauche
Binaire	$+$, $-$	Gauche
Comparaison	$<$, $<=$, $>$, $>=$	Gauche
Comparaison	$==$, $!=$	Gauche
Logique	&&	Gauche
Logique	$\|\|$	Gauche
Affectation	$=$, $+=$, $-=$, $*=$, $/=$, $\%=$	Droite

Le tableau est classé par ordre de priorité décroissante.

Priorités des opérateurs : exemples

Expression	**Expression parenthésée**	**Valeur**
2+3*7	2+(3*7)	23
15*3%7/2	((15*3)%7)/2	1
x=y=2	x=(y=2)	x:2, y:2
1<4==7<5!=9<4	((1<4)==(7<5))!=(9<4)	0 (faux)

Plan

Conversions de types

On appelle conversion de type le fait de transférer une valeur d'un type dans une valeur d'un autre type.

- Les conversions de type peuvent se faire avec ou sans perte d'information.
  - La perte d'information aura lieu si le codage de la valeur dans le type cible n'est pas possible (car sa taille est trop petite)
- De plus une conversion peut être :
  - explicite : voulue par le programmeur
  - implicite : décidée par le compilateur
    - pour réaliser un calcul
    - pour affecter une variable

Conversions implicites

Calculs réalisés par le processeur

Les calculs ne se font que sur les types : int, long, unsigned int, unsigned long, float et double.

- les valeurs des types `char`, `_Bool`, `short`, `unsigned char`, ou `unsigned short` sont donc implicitement converties en un `int` avant de faire un calcul.

C'est une conversion sans perte.

Attention

Pas de perte, mais des dépassements de capacité peuvent avoir lieu lors des calculs engendrant des résultats non attendus.

- Ex. 2147483647 + 1 vaut -2147483648

Conversions implicites

Résolution d'expression de types mixtes

Le langage étant faiblement typé, on peut avoir à appliquer des opérateurs sur des valeurs n'ayant pas les types attendus.

- Ex : `(1+2.5) && 1`

Des conversions implicites sont faites vers le type le plus riche :

`int -> float -> double` `int -> long -> double`

`signed -> unsigned`

Stockage dans une variable

Lors d'une affectation, la valeur de l'expression doit être rangée dans la variable $\Rightarrow$ conversion si types différents

- `int a = 3.5` sera effectué comme `int a=3`

Cette conversion peut engendrer des pertes.

Conversions explicites

On peut forcer le changement de type en effectuant un `cast`.

Conversion de type par cast

L'expression : `(type) exp`

- calcule la valeur `v` de `exp` dans le type `t_exp`
- puis convertit `v` du type `t_exp` dans le type `type`.

Exemple

```
1 int a=3,b=4;
2 double c= a/b;                    // c=0.0
3 double d= (double)a/(double) b    // d=0.75
```

Plan

Instructions

Un programme impératif est constitué d'une succession d'instructions exécutées les unes après les autres.

Definition

Une instruction correspond à une étape atomique dans le programme.
↪ toute instruction s'exécute complètement.

En C, une instruction est une expression terminée par un point-virgule.

Exemple

```
int a=3,c;
c=a+10;
```

Bloc d'instructions

Definition

Un bloc d'instruction est constitué par un ensemble d'instructions délimitées par des accolades

```
{ instr1; instr2; ...; instrn;}
```

Les blocs d'instruction peuvent être:

- imbriqués: `{ instr1; { instr2; instr3;} }`
- disjoints: `{ instr1; instr2; } { instr3;instr4;}`

Remarque

Le bloc d'instructions de la fonction `main` définit les instructions du programme.

Plan

Les entrées-sorties

Comme pour tout langage de programmation il est souhaitable de pouvoir interagir avec le programme:

- saisir des valeurs au clavier
- afficher des valeurs à l'écran

En C, les fonctionnalités d'entrée-sortie standards sont définies dans le fichier `stdio.h`.

```
1  #include <stdio.h>
2
3  int main() {
4      ...
5      return 0;
6  }
```

Instruction d'affichage

Definition

`printf("chaîne de contrôle", exp_1, ..., exp_n)`

- `printf` est le nom de la fonction d'écriture formatée sur la sortie standard (par défaut l'écran), fonction de la librairie C `stdio.h`
- `"chaîne de contrôle"` est une chaîne de caractères contenant le **texte à afficher** entrecoupé de n **spécifications de format** %d, %f, %c , %s ... une pour chacune des `exp_i`
- $n \geq 0$ expressions dont on veut afficher la valeur

```
1  #include <stdio.h>
2  int main() {
3     printf("Evaluation d'un calcul :\n");
4     printf("\tla valeur de %d + %d est %d\n",2,3,2+3);
5     return 0;
6  }
```

Fonctionnement de la fonction printf

Les spécifications de format spécifient :

- comment doit être affiché chaque expression `exp_i`
- le type attendu de chaque `exp_i`

Schéma d'exécution de `printf` :

1. Construction de la chaîne de caractères à afficher à partir de la chaîne de contrôle
   - par remplacement de chaque spécification de format par une séquence de caractères représentant la valeur de l'expression associée `exp_i`
   - les autres caractères de la chaîne de contrôle restent inchangés
2. Appel à une routine système d'entrée/sortie permettant l'écriture de la chaîne sur la sortie standard

Remplacement des spécifications de format

- À une spécification de format est associé :
  - un type de donnée
  - une notation comme suite de caractères des valeurs de ce type

  Exemple : à `%d` est associé :
  - *type :* `int`
  - *notation : séquence de caractères numériques évent. précédée d'un -*

- Soit un couple *(spéc. format, expression)* le remplacement consiste à :
  1. évaluer l'expression $\Rightarrow$ valeur du type de l'expression
  2. si besoin conversion implicite de cette valeur en une valeur du type du format
  3. transformation de cette dernière valeur en une suite de caractères correspondant à la notation associée au format

Les principales spécifications de format

format	**paramètre convertit en**	**notation à l'affichage**
%d	int	suite de chiffres (avec signe)
%hd	short	suite de chiffres (avec signe)
%ld	long	suite de chiffres (avec signe)
%u	unsigned int	suite de chiffres
%hu	unsigned short	suite de chiffres
%lu	unsigned long	suite de chiffres
%f	float	notation décimale ou scientifique
%lf	double	notation décimale ou scientifique
%c	unsigned char	caractère
%s	char*	chaîne de caractères
%p	void * (adresse mémoire)	valeur hexadécimale de l'adresse

Différentes options permettent de préciser ces formats (nombre de chiffres, affichage en octal...)

Instruction de saisie clavier

Definition

`scanf("chaîne de contrôle", adr_1, ..., adr_n)`

- `scanf` est le nom de la fonction de lecture formatée depuis l'entrée standard (par défaut le clavier)
- `"chaîne de contrôle"` est une chaîne de caractères contenant la chaîne à lire entrecoupée de **spécifications de format** des n données à lire et stocker aux adresses mémoires `adr_i`
- $n \geq 1$ adresses de variables déclarées que l'on veut affecter

```
#include <stdio.h>
int main() {
  int a;
  printf("Veuillez saisir un entier : ");
  scanf("%d", &a);
  printf("Vous avez saisi la valeur %d\n",a);
  return 0;
}
```

Tampons d'entrée/sortie

Pour éviter de trop nombreuses opérations physiques d'accès aux périphériques, des tampons d'entrée/sortie sont gérés par le système d'exploitation.

- Le système gère le remplissage du tampon d'entrée standard à partir du clavier
  - les caractères tapés au clavier ne sont introduits dans le tampon que lors d'un appui sur la touche <Entrée>
- La fonction `scanf` lit les caractères dans ce tampon
  - La lecture est **bloquante** : si aucun caractère à lire dans le tampon l'exécution du programme est bloquée jusqu'au remplissage du tampon

Fonctionnement de la fonction scanf

On traite itérativement chaque élément de la chaîne de contrôle :

- Les caractères "simples" (pas les spécifications de format) doivent être lus tels quels sur l'entrée standard
- Pour les spécifications de format :
  1. la **plus longue séquence de caractères correspondant à ce format** est lue sur l'entrée standard,
  2. convertit en une valeur du type associé au format,
  3. cette valeur est alors stockée à l'adresse de la variable correspondante.

Exemple : `scanf("%d,%c",&i,&c)` cherche à lire un entier relatif puis une virgule et un caractère.

Lecture/affectation d'une donnée

Cas d'erreurs

- La saisie s'interrompt (mais le programme continue) dès qu'un élément de la chaîne de contrôle n'est pas satisfait (c'est à-dire si les caractères sur l'entrée standard ne correspondent pas au caractère ou format de donnée attendu) $\Rightarrow$ les données restantes ne sont pas affectées.
- Si le type du format n'est pas compatible avec le type de la variable un débordement peut avoir lieu $\Rightarrow$ modification inattendue de la mémoire pouvant provoquer l'interruption du programme

Remarque : le format associé aux nombres (entier et réels) accepte qu'un nombre quelconque de caractères séparateurs préfixent ce nombre :

- tabulation
- retour à la ligne (touche <Entrée>)
- espace

Exemple : " 3.5" est une séquence de 9 caractères (6 espaces) qui peut être lue comme un flottant.

Exemples de saisies

exemple.c

```c
#include <stdio.h>
int main() {
    int i;
    float f;
    printf("Saisissez un entier et un reel\n");
    scanf("%d%f",&i,&f);
    printf("i vaut %d, f vaut %f\n",i,f);
    return 0;
}
```

Saisissez un entier et un réel
35 23.45e-4
i vaut 35, f vaut 0.002345

Saisissez un entier et un réel

23
2.3
i vaut 23, f vaut 2.300000

Saisissez un entier et un réel
val 23 2.3
i vaut 0, f vaut 0.000000

Saisissez un entier et un réel
2.3
i vaut 2, f vaut 0.300000

Saisissez un entier et un réel
.3 2.4
i vaut 0, f vaut 0.000000