Numeri Reali

Esistono due modi per rappresentare un numero reale:

Virgola fissa

0.0015 3.14 123.27 52.0

Virgola mobile

- Mantissa
- Esponente base 10
- Separate dal simbolo "e"

ESEMPIO il numero 1853.0000059 può essere rappresentato come: 185300.00059e-2

18530000059e-7

0.0018530000059e+6

notazione standard **0.18530000059e+4**

Tipi `float` e `double`

- Rappresentano (*con diversa precisione*) l'insieme dei numeri REALI
- In realtà (così come per `int` e INTERI) sono solo un'approssimazione dei numeri reali, sia come **precisione** sia come **intervallo** di valori rappresentabili
- I formati per l'acquisizione e la visualizzazione sono
  - **%f** : L'argomento è convertito in notazione decimale nel formato [-]ddd.ddd
  - **%e** : L'argomento e' convertito in notazione esponenziale, in cui viene riportata la mantissa, la parte decimale e l'esponete preceduto da "e"
  - **%g** : L'argomento e' convertito in automatico con le regole definite per il carattere di conversione **f** o **e**.

Esercizio

Provare ad eseguire il seguente programma

```
#include <stdio.h>

void main()
{
float f;
printf("Inserisci un valore reale in virgola mobile:\n");
scanf("%e",&f);
printf("Rappresentazione in virgola fissa: %f\n",f);
printf("Inserisci un valore reale nel formato preferito:\n");
scanf("%g",&f);
printf("Rappresentazione in virgola mobile: %e\n",f);

}
```


Precisione nella rappresentazione

Il numero di cifre disponibili per rappresentare un numero all' interno del computer è **limitato**, per cui si potrebbero verificare delle approssimazioni (**troncamenti** o **arrotondamenti**) nella rappresentazione di numeri reali con molte cifre.

ESEMPIO

Il numero 277290.0010044

se si avessero massimo 10 cifre a disposizione potrebbe essere rappresentato come 0.277290001e+6

Tuttavia, questa rappresentazione trasformerebbe il numero originario

277290.0010044 ---> **277290.001**

In molte applicazioni questa approssimazione non costituisce un problema, ma in altre applicazioni, come ad esempio quelle di calcolo scientifico, costituisce una seria fonte di errori.

Tipi `float` e `double` (*cont.*)

STANDARD

float	4 byte
double	8 byte
long double	10 byte

Tipo	Precisione	Valori
float	6 cifre decimali	3.4×10^{-38} ... $3.4 \times 10^{+38}$
double	15 cifre decimali	1.7×10^{-308} ... $1.7 \times 10^{+308}$

Operatori su `float` e `double`

I seguenti operatori sono quelli più comunemente applicati ai tipi **float** e **double** (e tipi ottenuti da questo mediante qualificazione) :

Operatori aritmetici	**Risultato**
+ - * /	float o double

Operatori relazionali	**Risultato**
== !=	(0 se falso, ≠ 0 se vero)
< > <= >=	(0 se falso, ≠ 0 se vero)

ESEMPIO

5.0 / 2.0	=	2.5
7.1 > 4.55	=	intero diverso da 0

Attenzione!

A causa della rappresentazione su di un numero finito di cifre, ci possono essere errori dovuti al troncamento o all'arrotondamento di alcune cifre decimali

Quindi:

i test di uguaglianza tra valori reali (in teoria uguali) potrebbero non essere verificati.

ESEMPIO (x / y) *y == x

Limiti

- Anche per i numeri *float* e *double* esiste una libreria <`float.h`> che indica i limiti nel range ammissibile

- Ad esempio

FLT_MAX_10_EXP +38
il massimo valore di esponente per un float con base 10
DBL_MAX_10_EXP +308
il massimo valore di esponente per un double con base 10
FLT_EPSILON 1.19209290E-07
il più piccolo numero float tale che 1.0+x è 1.0
DBL_EPSILON 2.22044…E-16
il più piccolo numero double tale che 1.0+x è 1.0
FLT_MIN_10_EXP -38
il più piccolo valore di esponente che può essere rappresentato in un numero float con base 10

Tipo `char`

- Rappresenta l'insieme dei caratteri disponibili sul sistema.
- Una costante carattere è un carattere racchiuso tra apici singoli

ESEMPIO `'A'  'c'  '6'`

- Il **formato** per l'acquisizione e per la visualizzazione è `%c`

ESEMPIO

```
# include <stdio.h>
main(){
char car;
printf("Inserisci un carattere: ");
scanf("%c",&car);
printf("Il carattere che hai inserito e' %c", car);
car= 'c';
printf("Il carattere modificato e' %c", car);}
```


Tipo `char` (cont.)

- Nel linguaggio C (a differenza di altri linguaggi) il tipo **`char`** non denota un nuovo tipo, ma è equivalente al dominio dei **valori naturali** rappresentati in un byte (range 0-255)
- Il valore intero è il valore numerico del carattere nel codice **ASCII**

ASCII è l'acronimo di
American **S**tandard **C**ode for **I**nformation **I**nterchange (ovvero *Codice Standard Americano per lo Scambio di Informazioni*)

- ASCII è un *sistema di codifica dei caratteri* comunemente utilizzato nei calcolatori perchè i calcolatori comprendono solo numeri

L'insieme dei caratteri ASCII

Valore ASCII	*Carattere*	*Valore ASCII*	*Carattere*	*Valore ASCII*	*Carattere*	*Valore ASCII*	*Carattere*
0	NUL	32	(blank)	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	–	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

I primi 32 caratteri e l'ultimo carattere costituiscono dei caratteri di controllo, che in genere non sono visibili.

Tipo `char` (cont.)

- Il *valore intero* di un carattere è il *valore numerico* del carattere nel codice ASCII

ESEMPIO

'A' è 65, 'c' è 99

Pertanto,
valgono gli stessi operatori degli `int`:

ESEMPIO

char car;
car = 'x';
ALLORA le espressioni

car / 'A'	ovvero 120 / 65	denota il valore 1
'R' < 'A'	ovvero 82 < 65	denota il valore 0
car – '4'	ovvero 120 – 52	denota il valore 68 (= 'D')
'x' – 4	ovvero 120 – 4	denota il valore 116 (= 't')

Caratteri speciali

- Per rappresentare alcuni caratteri, detti caratteri speciali, si usano le seguenti sequenze di controllo

Carattere	ASCII	Sequenza
Newline (riga a capo)	LF	\n
tab. orizzontale	VT	\t
Allarme	BEL	\a
Barra inversa	\	\\
Apice	‘	\’
Virgolette	“	\”

ESEMPIO

```
printf("E allora il lupo disse\n \"TI MANGIO\"");
```

Produce il seguente effetto a video

```
E allora il lupo disse
"TI MANGIO"
```


Osservazioni sui caratteri

- L’insieme dei caratteri è **ordinato**
- L’ordine, detto lessicografico, dipende dalla codifica ASCII

‘0’ < ‘1’ <…< ‘9’ <…< ‘A’ < ‘B’ <…< ‘Z’ <…< ‘a’ < ‘b’ <…< ‘z’

- Vale la regole di **prossimità** all’interno di ciascuna di queste tre classi ovvero, il carattere successivo di ‘a’ è ‘b’, di ‘B’ è ‘C’, di ‘4’ è ‘5’, ecc.
- Quindi ‘b’ -’ a’ = 1
  ‘d’ -’ a’ = 3

  … e, in generale, se X è una variabile carattere minuscolo allora

  X - ’a’ è l’offset di X da ‘a’ rispetto all’ordine lessicografico (lo stesso vale per le lettere maiuscole)
- **Tuttavia, le tre classi non sono contigue!**

Esercizio

Dato un carattere minuscolo, trasformarlo in maiuscolo

```
#include <stdio.h>

void main(){

  char c;

  printf("Digitare il carattere da convertire\n");
  scanf("%c", &c);

  c = 'A'+(c-'a');

  printf("Il corrispondente carattere maiuscolo è %c", c);

}
```


Altri Esercizi

- Stampare il codice ASCII corrispondente ad un carattere fornito da tastiera
- Stampare la distanza tra due caratteri forniti da tastiera
- Dato un carattere corrispondente ad una cifra, stampare la cifra come valore numerico
  - Esempio: Digitando '1' deve essere visualizzato 1

Tipo Boolean (*valori logici*)

- Il tipo Boolean è necessario
  - per dichiarare variabili che possono assumere valori booleani
  - per esprimere/usare il valore di verità di espressioni booleane
- In alcuni linguaggi di programmazione, come nel Pascal, esiste il tipo Boolean
- **Al contrario nel linguaggio C non esiste un tipo boolean e le espressioni logiche producono come risultato valori di tipo "int":**

  - il valore 0 (zero) indica **FALSO**

  Vengono considerati "falsi" anche: '\0' 0.0 EOF

  - ogni valore diverso da 0 indica **VERO**
- Per convenzione si usa 1 per denotare "vero", ma bisogna ricordare che è solo una convenzione

Scrivere espressioni booleane in C

Costanti booleane

- F è 0
- T è qualsiasi valore diverso da 0

Variabili booleane

Sono variabili booleane

- variabili di tipo int
  - nell' intenzione del programmatore dovrebbero rappresentare variabili booleane!
- le espressioni relazionali

Esempi 5 0 0 < a b%2 != (a+3)

Operatori booleani

operatore	*tipo*	C
not (negazione)	*unario*	**!**
and	*binario*	**&&**
or	*binario*	**||**

Esempi x&&y (a > b)|| (a > c) !x

Valutare espressioni booleane

- Ogni espressione logica scritta secondo le regole del linguaggio C è in corrispondenza biunivoca con un'espressione dell'algebra di Boole
- Quindi un'espressione logica in C può essere valutata seguendo le regole introdotte per l'algebra booleana

Esempi

5&&0 è falsa (5 > 7)||0 è falsa !0 è vera

Esercizio

Indicare se l'espressione è vera o falsa al variare di a e b.

((a>0) || (a%2==0)) || ((b<=200) && (b>=100))

Dati	Vero/Falso
int a=1,b=2;	
int a=2,b=2;	
int a=1, b=123;	
int a=1, b=10;	

Esercizio (cont.)

Indicare se l'espressione è vera o falsa al variare di a e b.

((a>0) || (a%2==0)) || ((b<=200) && (b>=100))

Dati	Vero/Falso
int a=1,b=2;	Vero
int a=2,b=2;	Vero
int a=1, b=123;	Vero
int a=1, b=10;	Vero

Esercizio

Indicare se l'espressione è vera o falsa.

Dati	Espressione	Vero/Falso
int a=3,b=2;	(b%2 == a-2) || (b>a)	
int a=-2,b=3;	(((b*b)%2) != (a+3)) && (a<b)	
int a=1, b=3;	!(((b-a%2)+2*a) > (a+(b%2)))	
int a=4, b=-1;	(((a+b*3)%5) <= (a-(b*a))) && !(a<b)	

Esercizio (cont.)

Indicare se l'espressione è vera o falsa.

Dati	Espressione	Vero/Falso
int a=3,b=2;	(b%2 == a-2) || (b>a)	Falso
int a=-2,b=3;	(((b*b)%2) != (a+3)) && (a<b)	Falso
int a=1, b=3;	! (((b-(a%2))+2*a) > (a+(b%2)))	Falso
int a=4, b=-1;	(((a+(b*3))%5) <= (a-(b*a))) && (!(a<b))	Vero

Esercizio

Dati VAL1 e VAL2 interi, scrivere un'espressione booleana che risulti vera quando

Condizione	Espressione
VAL1 è PARI	
VAL1 è DISPARI	
VAL2 è nell'intervallo [0,30]	
VAL2 non è nell'intervallo [0,30]	
VAL1 è positivo o pari o il valore di VAL2 è nell'intervallo [100,200]	

Esercizio (cont.)

Dati VAL1 e VAL2 interi, scrivere un' espressione booleana che risulti vera quando

Condizione	Espressione
VAL1 è PARI	VAL1%2 == 0
VAL1 è DISPARI	!(VAL1%2==0) ovvero VAL1%2!=0
VAL2 è nell' intervallo [0,30]	VAL2>=0 && VAL2<=30
VAL2 non è nell' intervallo [0,30]	!(VAL2>=0 && VAL2<=30) ovvero (VAL2<0 || VAL2 > 30)
VAL1 è positivo o pari o il valore di VAL2 è nell' intervallo [100,200]	VAL1>0 || VAL1%2==0 || (VAL2>=100 && VAL2<=200)

Esercizio

Dato VAL, scrivere un' espressione booleana che risulti vera quando

Condizione	Espressione
VAL è un carattere minuscolo	
VAL non è un carattere minuscolo	
VAL è un carattere	
VAL non è un carattere	
VAL è diverso da 's' e da 'n'	

Esercizio (cont.)

Dato VAL, scrivere un' espressione booleana che risulti vera quando

Condizione	Espressione	
VAL è un carattere minuscolo	VAL >= 'a' && VAL<= 'z'	
VAL non è un carattere minuscolo	!(VAL >= 'a' && VAL<= 'z')	VAL < 'a' || VAL> 'z'
VAL è un carattere	(VAL >= 'a' && VAL<= 'z') || (VAL>= 'A' && VAL<= 'Z')	
VAL non è un carattere	!((VAL>= 'a' && VAL<= 'z') || (VAL >= 'A' && VAL<= 'Z'))	(VAL < 'a' || VAL> 'z') && (VAL < 'A' || VAL> 'Z'))
VAL è diverso da 's' e da 'n'	VAL!='s' && VAL!='n'	

Valutazione in *corto-circuito*

Gli operatori logici && e // sono valutati in corto-circuito: la valutazione dell' espressione logica termina appena si è in grado di determinare il risultato

→ I termini successivi sono valutati ***solo se necessario***

Esempi

22 || x L' espressione è già vera in partenza (22 è vero)

0 && x L' espressione è già falsa in partenza (0 è falso)