Il linguaggio C
Variabili e tipi di dato primitivi

```
main()
{
/*definizioni variabili: */
char y='a'; /*codice(a)=97*/
int       x,X,Y;
unsigned int Z;
float SUM;
double r;

/* parte istruzioni: */
X=27;
Y=343;
Z = X + Y -300;
X = Z / 10 + 23;
Y = (X + Z) / 10 * 10;        /* qui X=30, Y=100, Z=70 */
X = X + 70;
Y  = Y % 10;
Z = Z + X -70;
SUM = Z * 10;       /* X=100, Y=0, Z=100 , SUM=1000.0*/
x=y;        /* char -> int: x=97*/
x=y+x; /*x=194*/
r=y+1.33; /* char -> int -> double*/
x=r; /* coercizione -> troncamento: x=98*/
}
```


Variabili

Una **variabile** rappresenta un dato che può cambiare il proprio valore durante l'esecuzione del programma.

In generale: nei linguaggi di alto livello una variabile è caratterizzata da un **nome** (***identificatore***) e 4 attributi base:

1. ***scope*** (***campo d'azione***), è l'insieme di istruzioni del programma in cui la variabile è nota e può essere manipolata;
   - C, Pascal, determinabile staticamente
   - LISP, dinamicamente
2. ***tempo di vita*** (o ***durata*** o ***estensione***), è l'intervallo di tempo in cui un'area di memoria è legata alla variabile;
   - FORTRAN, allocazione statica
   - C, Pascal, allocazione dinamica
3. ***valore***, è rappresentato (secondo la codifica adottata) nell'area di memoria legata alla variabile;
5. ***tipo***, definisce l'insieme dei valori che la variabile può assumere e degli operatori applicabili.

Variabili in C

- Ogni variabile, per poter essere utilizzata dalle istruzioni del programma, deve essere preventivamente definita.
- La definizione di variabile associa ad un identificatore (nome della variabile) un tipo.

<def-variabili> ::=
<identificatore-tipo> <identif-variabile> {, <identif-variabile> } ;

Esempi:

```
int  A, B, SUM;/* Variabili A, B, SUM intere  */
float     root, Root;   /* Variab. root, Root reali */
char      C ;           /* Variabile C carattere */
```

Effetto della definizione di variabile:

- La definizione di una variabile provoca come effetto l'**allocazione in memoria** della variabile specificata (allocazione automatica).
- Ogni istruzione successiva alla definizione di una variabile A, potrà utilizzare A

Assegnamento

- L' assegnamento è l'operazione con cui si attribuisce un (nuovo) valore ad una variabile.
- Il concetto di variabile nel linguaggio C rappresenta un'astrazione della cella di memoria.
- L'assegnamento, quindi, è l'astrazione dell'operazione di scrittura nella cella che la variabile rappresenta.

Esempi:

```
/* a, X e Y sono variabili */
int a ;
float X , Y ;
   ...
/* assegnamento ad a del valore 100: */
  a = 100 ;
/* assegnamento a Y del risultato di una espr.
   aritmetica: */
  Y = 2 * 3.14 * X ;
```

Tipo di dato

Il **tipo di dato** rappresenta una categoria di dati caratterizzati da proprietà comuni.

In particolare:

un tipo di dato T è definito:

- dall'insieme di **valori (dominio)** che le variabili di tipo T possono assumere;
- dall'insieme di **operazioni** che possono essere applicate ad operandi del tipo T.

Per esempio:

Consideriamo i numeri naturali

Tipo_naturali = [**N**, {+, -, *, /, =, >, <, … }]

- **N** è il *dominio*
- {+, -, *, /, =, >, <, … } è l'insieme delle operazioni effettuabili sui valori del dominio.

Il Tipo di dato

Un linguaggio di programmmazione è ***tipato*** se prevede ***costrutti specifici*** per attribuire tipi ai dati utilizzati nei programmi.

Se un linguaggio è tipato:

- Ogni dato (variabile o costante) del programma deve appartenere ad uno ed un solo tipo.
- Ogni operatore richiede operandi di tipo specifico e produce risultati di tipo specifico.

Vantaggi:

- **Astrazione**: l'utente esprime e manipola i dati ad un livello di astrazione più alto della loro organizzazione fisica (bit !). Maggior portabilità.
- **Protezione**: Il linguaggio protegge l'utente da combinazioni errate di dati ed operatori (controllo statico sull'uso di variabili, etc. in fase di compilazione).
- **Portabilità**: l'indipendenza dall'architettura rende possibile la compilazione dello stesso programma su macchine profondamente diverse.

Il tipo di dato in C

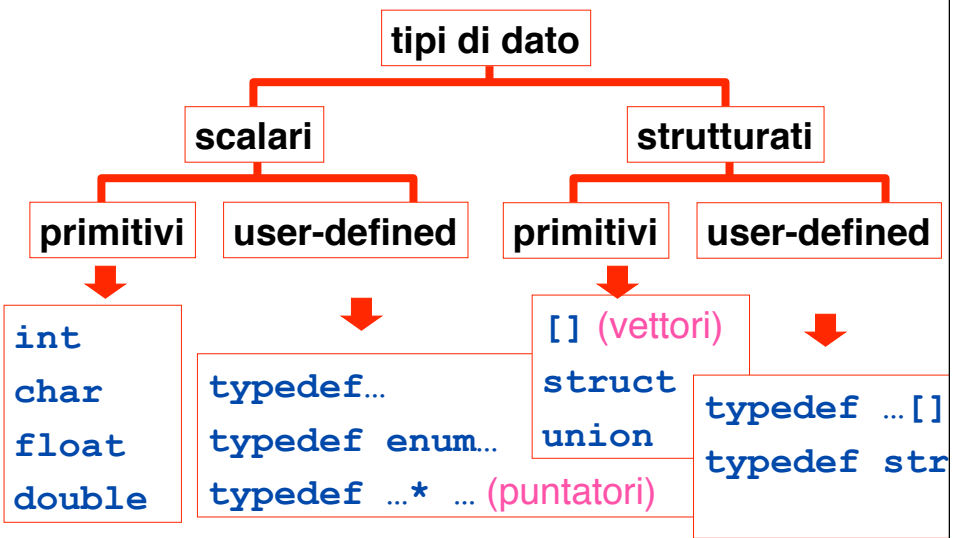
Il C è un linguaggio tipato.

Classificazione dei tipi di dato in C: due criteri di classificazione "ortogonali"

1. Si distingue tra:
- ***tipi scalari***, il cui dominio è costituito da elementi atomici, cioè logicamente non scomponibili.
- ***tipi strutturati***, il cui dominio è costituito da elementi non atomici (e quindi scomponibili in altri componenti).

2. Inoltre, si distingue tra:
- ***tipi primitivi***: sono tipi di dato previsti dal linguaggio (*built-in*) e quindi rappresentabili direttamente.
- ***tipi non primitivi***: sono tipi definibili dall'utente (mediante appositi costruttori di tipo, v. `typedef`).

Classificazione dei tipi di dato in C



Tipi scalari primitivi

Il C prevede quattro tipi scalari primitivi:

- **`char`** (caratteri)
- **`int`** (interi)
- **`float`** (reali)
- **`double`** (reali in doppia precisione)

Il tipo `int`

Dominio:
Il dominio associato al tipo **`int`** rappresenta l'insieme dei numeri interi con segno: ogni variabile di tipo **`int`** è quindi l'astrazione di un intero.

Esempio: definizione di una variabile intera.
(È la frase mediante la quale si *introduce* una nuova variabile nel programma.)

```
int A; /* A è una variabile intera */
```

- Poiché si ha sempre a disposizione un numero finito di bit per la rappresentazione dei numeri interi, il dominio rappresentabile è di estensione finita.
- Es: se il numero *n* di bit a disposizione per la rappresentazione di un intero è 16, allora il dominio rappresentabile è composto di:

$$(2^n) = 2^{16} = 65.536 \text{ valori}$$

Quantificatori e qualificatori

A dati di tipo `int` è possibile applicare i quantificatori *short* e *long*: influiscono sullo spazio in memoria richiesto per l'allocazione del dato.

- **`short`: la rappresentazione della variabile in memoria puo` utilizzare un numero di bit ridotto rispetto alla norma.**
- **`long`: la rappresentazione della variabile in memoria puo` utilizzare un numero di bit aumentato rispetto alla norma.**

Esempio:

```
int X;     /* se X e` su 16 bit..*/
long int Y;      /*..Y e` su 32 bit */
```

I quantificatori possono influire sul dominio delle variabili:

- Il dominio di un **`long int`** puo` essere piu` esteso del dominio di un int.
- Il dominio di uno **`short int`** puo` essere piu` limitato del dominio di un int.

Gli effetti di short e long sui dati dipendono strettamente dalla realizzazione del linguaggio.

In generale:

spazio(short int) ≤ spazio(int) ≤ spazio(long int)

Spazio: `sizeof(`*<tipo>*`)`

Quantificatori e qualificatori

A dati di tipo `int` e` possibile applicare i qualificatori *signed* e *unsigned*:

- **`signed`**: viene usato un bit per rappresentare il segno.Quindi l'intervallo rappresentabile e`:

  $[-2^{n-1}-1, +2^{n-1}-1]$

- **`unsigned`**: vengono rappresentati valori a priori positivi. Intervallo rappresentabile:

  $[0, 2^n -1]$

I qualificatori condizionano il dominio dei dati.

Il tipo int

Operatori:

Al tipo int (e tipi ottenuti da questo mediante qualificazione/ quantificazione) sono applicabili gli operatori ***aritmetici***, ***relazionali*** e ***logici***.

Operatori aritmetici:

forniscono risultato intero:

`+ , -, *, /` somma, sottrazione, prodotto, divisione intera.

`%` operatore modulo: calcola il resto della divisione intera.

`10%3` → `1`

`++, --` incremento e decremento: richiedono un solo operando (una variabile) e possono essere postfissi (a++) o prefissi (++a) (v. espressioni)

Operatori relazionali

Si applicano ad operandi interi e producono risultati *logici* (o "*booleani*").

Booleani: sono grandezze il cui dominio e` di due soli valori (valori logici): {*vero*, *falso*}

Operatori relazionali:

`==, !=` uguaglianza (==), disuguaglianza(`!=`):

`10==3` → *falso*

`10!=3` → *vero*

`<, >, <=, >=` minore, maggiore, minore o uguale, maggiore o uguale

`10>=3` → *vero*

Booleani

Sono dati il cui dominio e` di due soli valori (valori logici):

{*vero*, *falso*}

➔ in C **non esiste un tipo primitivo per rappresentare dati booleani** !

Come vengono rappresentati i risultati di espressioni relazionali ?

Il C prevede che i valori logici restituiti da espressioni relazionali vengano rappresentati attraverso gli interi {0,1} secondo la convenzione:

- *0* equivale a *falso*
- *1* equivale a *vero*

Ad esempio:

l'espressione A == B restituisce:

→ 0, se la relazione non e` vera
→ 1 , se la relazione e` vera

Operatori logici

Si applicano ad operandi di tipo logico (in C: `int`) e producono risultati *booleani* (cioe` interi appartenenti all'insieme {0,1}).

In particolare l'insieme degli operatori logici e`:

`&&` operatore AND logico
`||` operatore OR logico
`!` operatore di negazione (NOT)

Definizione degli operatori logici:

a	b	a&&b	a||b	!a
falso	*falso*	*falso*	*falso*	*vero*
falso	*vero*	*falso*	*vero*	*vero*
vero	*falso*	*falso*	*vero*	*falso*
vero	*vero*	*vero*	*vero*	*falso*

Operatori Logici in C

In C, gli operandi di operatori logici sono di tipo int:

- se il valore di un operando e` **diverso da zero**, viene interpretato come ***vero***.
- se il valore di un operando e` **uguale a zero**, viene interpretato come ***falso***.

Definizione degli operatori logici in C:

a	b	a&&b	a||b	!a
0	0	0	0	1
0	≠ 0	0	1	1
≠ 0	0	0	1	0
≠ 0	≠ 0	1	1	0

Operatori tra interi: esempi

37 / 3	→	12
37 % 3	→	1
7 < 3	→	0
7 >=3	→	1
5 >=5	→	1
0 || 1	→	1
0 || -123	→	1
15||0	→	1
12 && 2	→	1
0 && 17	→	0
! 2	→	0

I tipi reali: `float` e `double`

Dominio:
Concettualmente, è l'insieme dei numeri reali R.

In realtà, è un sottoinsieme di R :
- limitatezza del dominio (limitato numero di bit per la rappresentazione del valore).
- precisione limitata: numero di bit finito per la rappresentazione della parte frazionaria (*mantissa*)

Lo spazio allocato per ogni numero reale (e quindi l'insieme dei valori rappresentabili) dipende dalla realizzazione del linguaggio (e, in particolare, dal metodo di rappresentazione adottato).

Differenza tra float/double:

`float`	singola precisione
`double`	doppia precisione (maggiore numero di bit per la *mantissa*)

➔ Alle variabili `double` e` possibile applicare il quantificatore `long`, per aumentare ulteriormente la precisione: spazio(float) <= spazio(double) <= spazio(long double)

Esempio: definizione di variabili reali

```
float x;/* x e` una variabile reale "a singola precisione"*/
double A, B; /* A e B sono reali "a doppia precisione"*/
```


I tipi `float` e `double`

Operatori: per dati di tipo reale sono disponibili operatori aritmetici e relazionali.

Operatori aritmetici:

+, -, *, / si applicano a operandi reali e producono risultati reali

Operatori relazionali: hanno lo stesso significato visto nel caso degli interi:

==, != uguale, diverso

<, >, <=, >= minore, maggiore etc.

Operazioni su reali: esempi

5.0 / 2 → 2.5
2.1 / 2 → 1.05
7.1 < 4.55 → 0
17== 121 → 0

➔ A causa della rappresentazione finita, ci possono essere errori di conversione. Ad esempio, i test di uguaglianza tra valori reali (in teoria uguali) potrebbero non essere verificati.

(x / y) * y == x

Meglio utilizzare "un margine accettabile di errore":
(X == Y) → `(X<=Y+epsilon) && (X>=Y-epsilon)`

dove, ad esempio: `float epsilon=0.000001;`

Il tipo `char`

Dominio:
È l'insieme dei caratteri disponibili sul sistema di elaborazione (set di caratteri).

Comprende:
- le lettere dell'alfabeto
- le cifre decimali
- i simboli di punteggiatura
- altri simboli di vario tipo (@, #, $ etc.)
- caratteri speciali (backspace, carriage return, ecc.)
- ...

Tabella dei Codici
Il dominio coincide con l'insieme rappresentato da una ***tabella dei codici***, *dove*, ad ogni carattere viene associato un intero che lo identifica univocamente (il ***codice***).

- Il dominio associato al tipo char e` **ordinato**: l'ordine dipende dal codice associato ai vari caratteri.

La tabella dei codici ASCII

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
0	(null)	NUL	32	(space)	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	(beep)	BEL	39	'	71	G	103	g
8	◘	BS	40	(	72	H	104	h
9	(tab)	HT	41	)	73	I	105	i
10	(line feed)	LF	42	*	74	J	106	j
11	(home)	VT	43	+	75	K	107	k
12	(form feed)	FF	44	,	76	L	108	l
13	(carriage return)	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	(cursor right)	FS	60	<	92	\	124	|
29	(cursor left)	GS	61	=	93	]	125	}
30	(cursor up)	RS	62	>	94	^	126	~
31	(cursor down)	US	63	?	95	_	127	⌂

Il tipo char

Definizione di variabili di tipo char: esempio

```
char C1, C2;
```

Costanti di tipo char:
Ogni valore di tipo char viene specificato tra singoli apici.

```
'a'   'b'   'A'   '0'   '2'
```

Rappresentazione dei caratteri in C:
Il linguaggio C rappresenta i dati di tipo char come degli interi su 8 bit:

- ogni valore di tipo char viene rappresentato dal suo **codice** (cioe`, dall'intero che lo indica nella tabella ASCII)

➔ Il dominio associato al tipo `char` e` **ordinato**: l'ordine dipende dal codice associato ai vari caratteri nella tabella di riferimento.

Il tipo char: Operatori

I char sono rappresentati da interi (su 8 bit):
➔ sui char è possibile eseguire tutte le operazioni previste per gli interi. Ogni operazione, infatti, e` applicata ai codici associati agli operandi.

Operatori relazionali:
`==,!=,<,<=,>=,>` per i quali valgono le stesse regole viste per gli interi

Ad esempio:
```
char x,y;
```
`x < y` se e solo se codice(x) < codice(y)

`'a'>'b'` *falso* perché codice(`'a'`) < codice(`'b'`)

Operatori aritmetici:
sono gli stessi visti per gli interi.

Operatori logici:
sono gli stessi visti per gli interi.

Operazioni sui char: esempi

Esempi:

'A' < 'C' → 1 (infatti 65 < 67 è vero)

' " ' + ' ! ' → 'C' (codice(")+codice(!)=67)

! 'A' → 0 (codice(A) è diverso da 0)

'A' && 'a' → 1