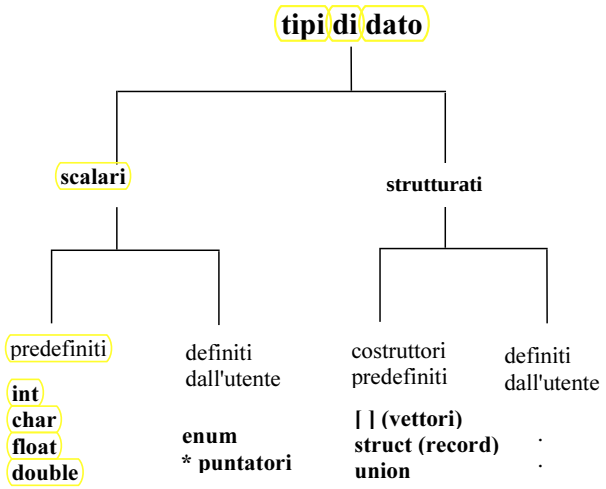
- 2° Modulo - parte 2^
  - Struttura del programma in C
  - Tipi Semplici del C
  - Costanti, Variabili, Operatori
  - Espressioni omogenee
  - Operatori aritmetici, logici, bitwise
  -

http://www.elet.polimi.it/upload/martucci/index.html

Classificazione dei tipi di dato in C



Tipi primitivi

Il C prevede quattro tipi primitivi:

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

☞ E` possibile applicare ai tipi primitivi dei **quantificatori** e dei **qualificatori**:

Quantificatori:

- I ***quantificatori*** (*long* e *short*) influiscono sullo spazio in memoria richiesto per l'allocazione del dato.
  - ***short*** (applicabile al tipo **int)**
  - ***long*** (applicabile ai tipi **int** e **double**)

Esempio:

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

Qualificatori:

- I ***qualificatori*** condizionano il dominio dei dati:
  - ***signed*** (applicabile ai tipo **int** e **char)**
  - ***unsigned*** (applicabile ai tipo **int** e **char)**

```
int A;     /*A in[-2e15,2e15-1] */
unsigned int B;    /*B in[0,2e16-1]*/
```

TIPI SEMPLICI BUILT-IN DEL C

I nomi di questi tipi sono delle parole chiave del linguaggio:

- **char**: (8 bit - 1 byte) Valori da 0 a 255 che rappresentano la codifica ASCII estesa del carattere corrispondente

- **int** (16bit - 2 byte) Rappresentano gli interi relativi. Valori in complemento a 2 da - 32768 a + 32767

- **float** (32 bit - 4 byte). Rappresentano i razionali espressi in virgola mobile (buona approssimazione dei reali). Valori espressi tramite mantissa e esponente (standard IEEE). Intervallo di valori rappresentabili: da -10^{38} a $+ 10^{38}$)

- **double** (8 byte). Sono float in doppia precisione.

Si dicono tipi aritmetici (char, int *integral*; float e double *floating*)

L'insieme di valori ammissibili (vmin e vmax) e lo spazio allocato in memoria possono essere modificati tramite **qualificatori** (specificatori) di tipo. I qualificatori sono parole chiave del linguaggio che si premettono al tipo.

Indirizzo di una variabile

- operatore: **`&nome_var`**
- valori assunti per gli indirizzi: interi >= 0

`&nome_var` rappresenta l'indirizzo di memoria del primo byte allocato per la variabile.

Il tipo int

Operatori:

Al tipo **int** (e tipi ottenuti da questo mediante qualificazione/quantificazione) sono applicabili i seguenti operatori:

Operatori aritmetici:

forniscono risultato intero:

+ , -, *, /	somma, sottrazione, prodotto, divisione intera.
%	operatore *modulo*: 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 ***"booleani"*** (cioe`, il cui valore puo` assumere soltanto uno dei due valori {*vero*, *falso*}):

==, !=	uguaglianza, disuguaglianza: 10==3 ➠ *falso* 10!=3 ➠ *vero*
<, >, <=, >=	minore, maggiore, minore o uguale, maggiore o uguale 10>=3 ➠ *vero*

Il tipo char: Operatori

I char sono rappresentati da interi (su 8 bit):

☞ sui dati **char** e` 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*** perche` *codice*('a') < *codice*('b')

Operatori aritmetici:

sono gli stessi visti per gli interi.

Operatori logici:

sono gli stessi visti per gli interi.

Esempi:

'A' < 'C' ➠ **1** (infatti 65 < 67 e` *vero*)

' " ' + ' ! ' ➠ **'C'** (*codice*(")+*codice*(!)=67)

! 'A' ➠ 0 (*codice*(A) e` diverso da zero)

'A' && 'a' ➠ 1

Uso dei qualificatori:
e` possibile, come per gli interi, applicare i qualificatori signed, unsigned a variabili di tipo char:

```
signed char C;
unsigned char K;
```

Data Types: Int vs Char

Why is it that given ...

```
char number1;
int    number2;
int    number3;

number1 = 160;              /* assigns values */
number2 = 565;              /* to variables */
number3 = number2;
number2 = number1;    /* this is OK */
number1 = number3;    /* this is NOT? */
```

I tipi float e double (reali)

Dominio:

Concettualmente, e` l'insieme dei numeri reali R.

In realta`, e` un sottoinsieme di R a causa di:

- **precisione** limitata
- **limitatezza** del dominio.

Lo spazio allocato per ogni numero reale (e quindi l'insieme dei valori rappresentabili) dipende dal metodo di rappresentazione adottato.

Differenza tra float/double:

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

Uso del quantificatore long:

si puo` applicare a **double**, per aumentare ulteriormente la precisione:

spazio(**float**) <= *spazio*(**double**) <= *spazio*(**long double**)

Esempio: definizione di variabili reali

```
float x;
double A, B;
```


Tipi float/double

Operatori

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.

Overloading:

Il C (come Pascal, Fortran e molti altri linguaggi) operazioni primitive associate a tipi diversi possono essere denotate con lo stesso simbolo (ad esempio, le operazioni aritmetiche su reali od interi).

ISTRUZIONI DI ASSEGNAMENTO

Indicano **l'operazione** che assegna un **valore** ad una **variabile**

Sintassi C:

```
<nome_variabile> = <espressione>;
```

- **<nome_variabile>** indica il nome della **variabile** a cui si vuole assegnare un valore, in sostituzione di quello precedentemente contenuto in essa ... è un indirizzo
- **=** è il simbolo di assegnamento
- **<espressione>** descrive come ottenere il **valore** da assegnare alla variabile

Significato (semantica):
si eseguono le operazioni descritte in **<espressione>**
il valore ottenuto viene inserito nella posizione di memoria indicata dalla variabile a sinistra del simbolo di assegnamento.

Compatibilità tra i tipi: il compilatore controlla la compatibilità tra tipi. In alcune situazioni, risolve la non compatibilità adottando delle regole di conversione implicita e automatica tra tipi.

- il valore generato da **<espressione>** dovrebbe essere dello stesso tipo della variabile da assegnare
- assegnamento tra tipi eterogenei (tipi aritmetici): all'atto dell'assegnamento, il valore di **<espressione>** viene «convertito» in un valore corrispondente appartenente al tipo della variabile da assegnare.

ESPRESSIONI E OPERATORI

Sintassi C

<espressione>: contiene identificatori (di variabili, di costanti, di funzioni), costanti esplicite, operatori, ( )

Semantica

descrive il modo con cui ottenere dal valore degli operandi e dall'applicazione degli operatori (operazioni) il valore dell'espressione.

Nelle espressioni complesse la **sequenza di esecuzione delle operazioni** è dettata dalla

- **precedenza** predefinita degli operatori
- **forzatura** mediante l'uso delle parentesi tonde

Operatori

- unari si applicano ad un solo operando
- binari si applicano a due operandi

TIPI DI OPERATORI

Operatori aritmetici (+ - * / %)

- operandi aritmetici e risultato aritmetico
- le operazioni sono eseguite in dipendenza del tipo degli operandi.

Operatori di confronto (> >= == (eguale) != (diverso))

- operandi entrambi dello stesso tipo qualsiasi
- risultato valore logico

Valori logici: in C non esiste un tipo «logico» che assume solo i valori False e True, ma per tale scopo viene usato il tipo `int`

- il valore FALSE (**falso**) è associato al valore **zero**
- il valore TRUE (**vero**) è associato ad ogni valore **diverso da zero**

Operatori logici

- operandi logici e risultato logico

&& (AND)
|| (OR)
! (NOT) operatore unario (con un solo operando)

Esempio

```
(num > valmin) && (num <= valmax)
```

Questa espressione logica vale TRUE solo se il valore di **num** è compreso tra `valmin` (escluso) e `valmax` (compreso).

Regole di Precedenza e Associativita` degli Operatori C (in ordine di priorita` decrescente)

Operatore	Associativita`
`() [] ->`	da sinistra a destra
`! ~ ++ -- & sizeof`	da destra a sinistra
`* / %`	da sinistra a destra
`+ -`	da sinistra a destra
`<< >>`	da sinistra a destra
`< <= > >=`	da sinistra a destra
`== !=`	da destra a sinistra
`&`	da sinistra a destra
`^`	da sinistra a destra
`|`	da sinistra a destra
`&&`	da sinistra a destra
`||`	da sinistra a destra
`?:`	da destra a sinistra
`=+ =- =* =/`	da destra a sinistra
`,`	da sinistra a destra

Precedenza e Associativita`

Esempi

3*5 % 2 ➠ equivale a: (3*5) % 2
X+7-A ➠ equivale a: (X+7) - A
3 < 0 && 3 < 10 ➠ (3<0) && (3<10) ➠ 0 && 1
3<(0 && 3) < 10 ➠ (3 < 0) < 10 ➠ 0 < 1
0 == 7 == 3 ➠ 0==(7==3) ➠ 0 == 0

Valutazione a "corto circuito" *(short-cut)*:

nella valutazione di una espressione C, se un risultato intermedio determina a priori il risultato finale della espressione, il resto dell'espressione non viene valutato.

Ad esempio, espressioni logiche:

Hp. Valutazione degli operandi da sin a destra

(3 > 0) && (X < Y) ➠ solo primo operando
falso && vero

☞ Bisognerebbe evitare di scrivere espressioni che dipendono dal metodo di valutazione usato ➠ scarsa portabilita` (ad es., in presenza di funzioni con effetti collaterali).

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 **int** e producono risultati *booleani*, cioe` interi appartenenti all'insieme {0,1} (il valore 0 corrisponde a "falso", il valore 1 corrisponde a " vero"). 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	$\neq 0$	0	1	1
$\neq 0$	0	0	1	0
$\neq 0$	$\neq 0$	1	1	0

Esempi sugli operatori tra interi:

37 / 3 ➠ 12

37 % 3 ➠ 1

7 < 3 ➠ 0

7 >=3 ➠ 1

0 || 1 ➠ 1

0 || -123 ➠ 1

12 && 2 ➠ 1

0 && 17 ➠ 0

! 2 ➠ 0

...e anche ...Operatori sui Bit

- ~ → complemento a uno
- << → scorrimento a sinistra
- >> → scorrimento a destra
- & → AND
- ^ → XOR

	0	0	1	1
	0	1	1	0
XOR	0	1	0	1

- | → OR

Logical and Bitwise Operators

- Logical (true/false): `&&`, `||`, `!`

```
int n = 7 || 0;    // n == 1
```

- One's complement: `~`

```
n = ~4;    // all bits on except third to last
```

- Bitwise shifts: `<<`, `>>`

```
n = 12 >> 2;    // n == 3
```

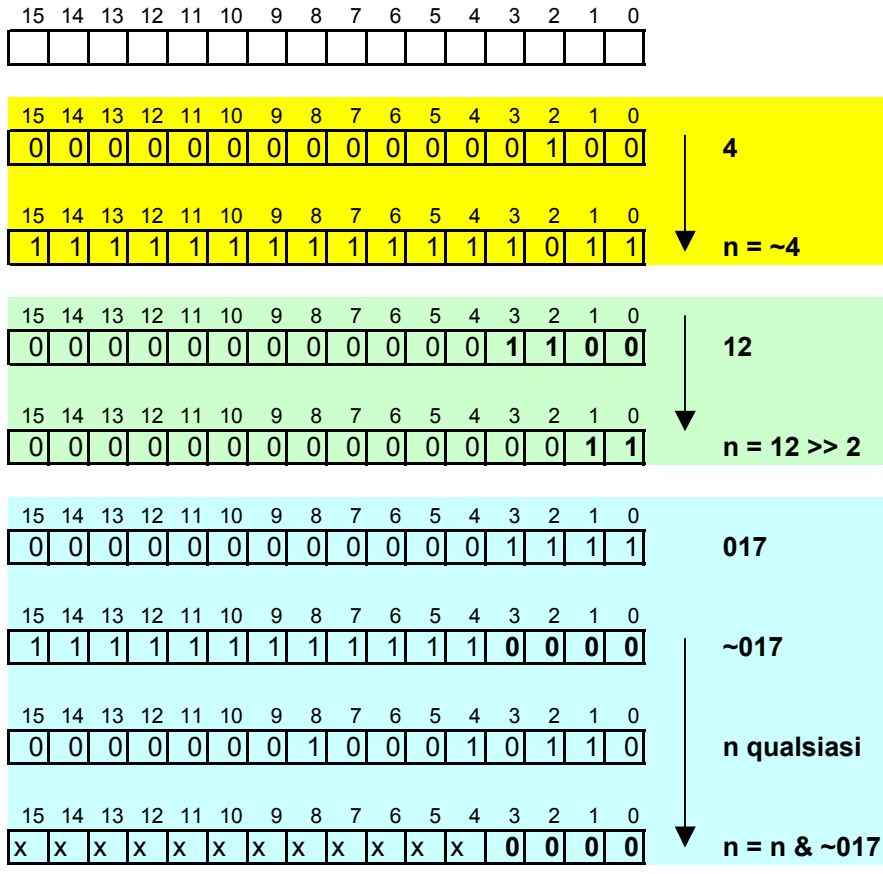
  * standard set for unsigned, integral types only
  * bit-wrap, 0-fill or 1-fill is direction and compiler dependent

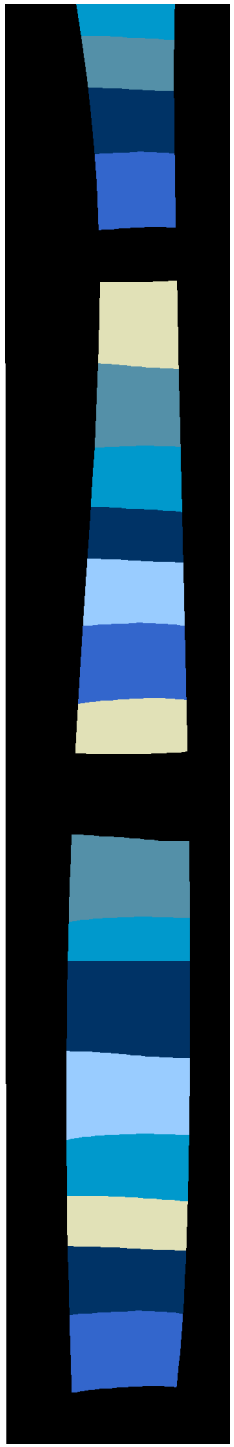
- Bitwise masks, logically bit-by-bit: `&`, `|`, `^`

```
n = n & ~017;    // zeros out last 4 bits
```

  Note: better than `n & 0177760` which assumes $\geq$ 16 bits

es su 16 bit


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
4
1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
n = ~4
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
12
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
n = 12 >> 2
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
017
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
~017
0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0
n qualsiasi
x x x x x x x x x x x x 0 0 0 0
n = n & ~017



Esempi

- foo1() {

```
    int x=0, y= 0, z=29;      → z=0001 1101
        x = z << 1;           → x=0011 1010
        x = z >> 1;           → x=0000 1110
        y = ~z; }             → y=1110 0010
```

- foo2() {

```
    int x=3, y= 5, z=0;       → x=0011, y=0101
        z = x & y;            → z=0001
        z = x | y; }          → z=0111
```