Tipi di dato

- Il concetto di ***tipo di dato*** viene introdotto per raggiungere due obiettivi:

- esprimere in modo sintetico
  - la loro rappresentazione in memoria, e
  - un insieme di operazioni ammissibili

- permettere di ***effettuare controlli statici*** (al momento della compilazione) sulla *correttezza* del programma

Tipi di dato PRIMITIVI in C

- Caratteri:
  - **char** (caratteri ascii)
- Interi
  - Con segno
    - **short**: *16 bit -32.768 a +32.768*
    - **int** *dipende dal compilatore*
    - **long**: *32 bit*
  - Senza segno (Naturali)
    - **unsigned short**: *16 bit da 0 a …*
    - **unsigned**
    - **unsigned long**: *32 bit da 0 a …*
- Reali
  - **float** (singola precisione 32 bit) da 10^{-38} a 10^{38}
  - **double** (doppia precisione 64 bit) da 10^{-308} a 10^{308}

Tipi di dato INTERESSANTI in C

- **Boolean**:
  - **zero**: indica **falso**
  - Tutto il resto: indica **vero** (utilizzare **uno**)

- **Stringhe**: sequenze di caratteri delimitata da virgolette

  “ciao” “Una lunga frase”

  l’ultimo carattere, ***sempre presente in modo implicito***, è `'\0'`

  `"ciao" ={'c','i','a','o','\0'}`

Espressioni

- Il C è un linguaggio basato su ***espressioni***
- Una ***espressione*** è una *notazione che* ***denota un valore*** mediante un processo di ***valutazione***
- Una espressione può essere *semplice* o *composta* (tramite aggregazione di altre espressioni)

ESPRESSIONI SEMPLICI

Costanti	**Variabili**	**funzioni**
‘A’ 23.4 -3 “esempio”	x pigreco pippo	f(x) concat(“ci”,”ao”)

Operatori ed espressioni composte

Ogni linguaggio introduce un **insieme di *operatori*** che permettono di ***aggregare altre espressioni (operandi)*** per formare ***espressioni composte*** con riferimento a diversi **domini / tipi di dato** (numeri, testi, ecc.)

Esempi

```
2 + f(x)
4 * 8 - 3 % 2 + arcsin(0.5)
strlen(strcat(Buf,"alfa"))
a && (b || c)
```

Operatori

Gli operatori si possono classificare:

- in base al *tipo* degli operandi
  - aritmetici, relazionali, logici, ecc
- in base al *numero* degli operandi
  - Unari, binari, ternari, ecc

Operatori aritmetici

operazione	*operatore*	*in C*
inversione di segno	*unario*	`-`
somma	*binario*	`+`
differenza	*binario*	`-`
moltiplicazione	*binario*	`*`
divisione fra interi	*binario*	`/`
divisione fra reali	*binario*	`/`
modulo (fra interi)	*binario*	`%`

la divisione `a/b` è fra interi se sia `a` sia `b` sono interi,
è fra reali in tutti gli altri casi

Operatori: overloading

In C (come in Pascal e molti altri linguaggi) ***operazioni primitive associate a tipi diversi possono essere denotate con lo stesso simbolo***.
Ad esempio, le operazioni aritmetiche su reali o interi

In realtà ***l'operazione è diversa e può produrre risultati diversi***

```
int x, y;
x=10;  y=4;
```

x/y vale 2

```
int x; float y;
x=10;  y=4.0;
```

x/y vale 2.5

```
float x, y;
x=10.0;  y=4.0;
```

x/y vale 2.5

Conversioni di tipo

In C è possibile combinare tra di loro operandi di tipo diverso:

- espressioni **omogenee**: tutti gli operandi sono dello stesso tipo
- espressioni **eterogenee**: gli operandi sono di tipi diversi

Regola adottata in C:

sono eseguibili le espressioni eterogenee in cui tutti i tipi referenziati risultano ***compatibili*** (cioè, dopo l'applicazione della regola automatica di conversione implicita di tipo del C risultano omogenei)

Compatibilità di tipo

Consiste nella possibilità di usare, *entro certi limiti*, oggetti di un tipo *al posto di oggetti di un altro tipo*

Un tipo T1 è *compatibile* con un tipo T2 se il dominio D1 di T1 *è contenuto* nel dominio D2 di T2

- **`int` è compatibile con `float` perché Z contiene R**
- **ma `float` *non* è *compatibile* con `int`!**

- 3 / 4.2: divisione *fra reali,* il primo operando è convertito automaticamente da **`int`** a **`double`**
- 3 % 4.2: operazione *non ammissibile*, perché 4.2 non può essere convertito in **`int`**

Conversione di tipo

Data una espressione *x op y*

2. Ogni variabile di tipo **char** o **short** viene convertita nel tipo **int**;
3. Se dopo l'esecuzione del passo 1 l'espressione è ancora eterogenea, rispetto alla seguente gerarchia

   int < long < float < double < long double

   si converte temporaneamente l'operando di tipo *inferiore* al tipo *superiore* (***promotion***)
3. A questo punto l'espressione è **omogenea** e viene eseguita l'operazione specificata. **Il risultato è di tipo uguale a quello prodotto dall'operatore** effettivamente eseguito (in caso di overloading, quello più alto gerarchicamente)

Conversione di tipo: esempio

```
int x;
char y;
double r;
(x+y) / r
```

- **Passo 1 `(x+y)`**
  - `y` viene convertito nell'intero corrispondente
  - viene applicata la somma tra interi
  - **risultato intero *`tmp`***

- **Passo 2 *`tmp`* `/ r`**
  - *`tmp`* viene convertito nel double corrispondente
  - viene applicata la divisione tra reali
  - **risultato reale**

Compatibilità in assegnamento

In un ***assegnamento***, ***l'identificatore di variabile e l'espressione*** devono essere dello ***stesso tipo***

- Nel caso di tipi diversi, se possibile si effettua la **conversione implicita**, altrimenti l'assegnamento può generare perdita di informazione
- In generale, sono automatiche le conversioni di tipo che non provocano perdita d'informazione
- Espressioni che *possono* provocare perdita di informazioni **NON sono però illegali**

```
int x; char y; double d;
x = y; /* char -> int*/
x = y+x;
d = y; /* char -> int -> double*/
x = d; /* troncamento*/
```

Warning:
conversion may lose significant digits

Cast

In qualunque espressione è possibile **forzare una particolare conversione** utilizzando l'***operatore di cast***

(<tipo>) <espressione>

Esempi

```
int i=5; double x=7.1;
i = (int) sqrt(384);
i = i % (int) x;
```

Operatori relazionali

Sono tutti **operatori binari:**

uguaglianza	==
diversità	!=
maggiore di	>
minore di	<
maggiore o uguale a	>=
minore o uguale a	<=

N.B.: in C **non esiste il tipo *boolean,*** *quindi* le espressioni relazionali *denotano un valore intero*

- **0 denota *falso*** (condizione non verificata)
- **1 denota *vero*** (condizione verificata)

Operatori logici

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

Anche le espressioni logiche *denotano un valore intero*

la valutazione dell'espressione cessa *appena si è in grado di determinare il risultato, quindi* il secondo operando è valutato *solo se necessario*

Es: 5 && 7 0 || 3 ! 0

Espressione condizionale

Una espressione condizionale è introdotta dall'operatore ternario

condiz* ? *espr1* : *espr2

L'espressione denota:

- o il valore denotato da *espr1*
- o quello denotato da *espr2*

in base al valore della espressione *condiz*

se *condiz* è vera, l'espressione denota il valore denotato da *espr1*

se *condiz* è falsa, l'espressione denota il valore denotato da *espr2*

Esempi:

```
x ? 10 : 20;    // se x è vera, denota 10, altrimenti 20
(x>y) ? x: y;   //denota il maggiore tra x e y
```

Incremento e decremento

Gli operatori di incremento e decremento sono *usabili in due modi*

- **come pre-operatori: ++i**

prima incremento e poi uso nell'espressione il valore della var incrementata

- **come post-operatori: i++**

prima uso nell'espressione il valore della var e poi incremento

Formule equivalenti:

v = v + 1; **v +=1;** **++v;** **v++;**

Esempi

```
int i, k = 5;
i = ++k;       /* i vale 6, k vale 6 */
```

```
int i, k = 5;
i = k++;       /* i vale 5, k vale 6 */
```