Espressioni ed operatori in C

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

Quali espressioni elementari?

- costanti

  ‘A’ 23.4 -3 “ciao” ....

- simboli di variabile

  x pippo pigreco ….

- simboli di funzione

  f(x)
  concat(“alfa”,”beta”)
  ...

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)
  - ...

Classificazione degli operatori

- Due criteri di classificazione:
  - in base al *tipo* degli operandi
  - in base al *numero* degli operandi

In base al tipo degli operandi	In base al numero degli operandi
▪ aritmetici ▪ relazionali ▪ logici ▪ condizionale	▪ unari ▪ binari ▪ ternari ▪ …

Operatori aritmetici

Operazione	Operatore	C
Inversione di segno	unario	-
Somma	binario	+
Differenza	binario	-
Moltiplicazione	binario	*
Divisione fra interi	binario	/
Divisione fra reali	binario	/
Modulo (fra interi)	binario	%

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

Operatori: overloading

- In C (come in 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 o interi
- In realtà l'operazione è diversa e può produrre risultati diversi

```
int X, Y;
Se X=10 e Y=4;
X/Y vale 2
```

```
int X, float Y;
Se X=10 e Y=4.0;
X/Y vale 2.5
```

```
float X, Y;
Se X=10.0 e 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 $Z \subset R$
  - ma float *non è compatibile* con int

Compatibilità di tipo

- 3 / 4.2 è una divisione *fra reali,* in cui il primo operando è convertito automaticamente da **int** a **double**

- **3 % 4.2** è una operazione *non ammissibile*, perché 4.2 non può essere convertito in **int**

Conversioni di tipo

Data una espressione ***x op y***

1. Ogni variabile di tipo **char** o **short** viene convertita nel tipo **int**;

2. 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)

Conversioni di tipo - esempio

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

La valutazione dell'espressione procede da sinistra verso destra

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 di tipo

- 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

```
int x;
char y;
double r;

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


Compatibilità in assegnamento

- In generale, sono automatiche le conversioni di tipo che non provocano perdita d'informazione
- Tuttavia, le espressioni che *possono* provocare perdita di informazioni non sono illegali

Esempio

```
int i=5; float f=2.71F;; double d=3.1415;
f = f+i;  /* int convertito in float */
i = d/f;  /* double convertito in int !*/
f = d;    /* arrotondamento o troncamento */
```

Possible **warning**: conversion may lose significant digits

Casting

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

Sintassi

(<tipo>) <espressione>

Esempi

```
int i=5; long double x=7.77; double y=7.1;
i = (int) sqrt(384);
x = (long double) y*y;
i = (int) x % (int)y;
```


Esempio

```
main() {
/* parte dichiarazioni variabili */
   int X,Y;
   unsigned int Z;
   float SUM;
/* segue parte istruzioni */
   X=27;
   Y=343;
   Z = X + Y -300;          /* qui Z vale 70 */
   X = Z / 10 + 23;         /* qui X vale 30 */
   Y = (X + Z) / 10 * 10;   /* qui Y vale 100 */
   X = X + 70;               /* qui X vale 100 */
   Y = Y % 10;               /* qui Y vale 0 */
   Z = Z + X -70;            /* qui Z vale 100 */
   SUM = Z * 10;             /* qui SUM vale 1000.0 */
   /* qui X=100, Y=0, Z=100 , SUM =1000.0*/
}
```

Operatori relazionali

- Sono tutti operatori binari:

Relazione	C
Uguaglianza	==
Diversità	!=
Maggiore di	>
Minore di	<
Maggiore o uguale a	>=
Minore o uguale a	<=

Operatori relazionali

Attenzione:

- non esistendo il tipo *boolean,* in C le espressioni relazionali *denotano un valore intero*
  - **0** denota *falso*
    (*condizione non verificata*)
  - **Un valore diverso da 0** denota *vero*
    (*condizione verificata*)
    (NB: ricordiamo che per convenzione, spesso si usa 1 per denotare vero, anche se sottolineiamo che ciò è solo una convenzione)

Operatori logici

connettivo logico	Operatore	C
not (negazione)	unario	**!**
and	binario	**&&**
or	binario	**||**

- In C, non esistendo il tipo boolean, gli operatori logici:
  - operano su interi
  - e restituiscono un intero
    da interpretare come vero (**1**) o falso (**0**)

Operatori logici

- Anche qui sono possibili espressioni miste, utili in casi specifici

```
5 && 7          0 || 33          !5
```

- **Valutazione in *corto-circuito***
  - la valutazione dell'espressione cessa *appena si è in grado di determinare il risultato*
  - il secondo operando è valutato *solo se necessario*

Valutazione in corto circuito

- `22 || x`

  già vera in partenza perché 22 è vero

- `0 && x`

  già falsa in partenza perché 0 è falso

- `a && b && c`

  se a&&b è falso, il secondo && non viene neanche valutato

- `a || b || c`

  se a||b è vero, il secondo || non viene neanche valutato

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 nel suo complesso denota il valore denotato da *espr1*
- se *condiz* è falsa, l'espressione nel suo complesso denota il valore denotato da *espr2*

Espressione condizionale

ESEMPI

- `3 ? 10 : 20`
  - denota sempre 10 (3 è sempre vera)
- `x ? 10 : 20`
  - denota 10 se x è vera (diversa da 0),
  - oppure 20 se x è falsa (uguale a 0)
- `(x>y) ? x : y`
  - denota il maggiore fra x e y

Operatori infissi, prefissi e postfissi

- Le espressioni composte sono **strutture** formate da **operatori** applicati a uno o più **operandi**

Ma.. *dove* posizionare l'operatore rispetto ai suoi operandi?

Operatori infissi, prefissi e postfissi

Tre possibili scelte:

- **prima** → notazione ***prefissa***
  - Esempio: + 3 4
- **dopo** → notazione ***postfissa***
  - Esempio: 3 4 +
- **in mezzo** → notazione ***infissa***
  - Esempio: 3 + 4

È quella a cui siamo abituati, perciò è adottata *anche in C*

Operatori infissi, prefissi e postfissi

- Le notazioni *prefissa* e *postfissa* non hanno problemi di *priorità* e/o *associatività* degli operatori
  - non c'è mai dubbio su *quale* operatore vada applicato a *quali* operandi
- La notazione *infissa* richiede *regole* di **priorità** e **associatività**
  - per identificare univocamente ***quale*** operatore sia applicato a ***quali*** operandi

Priorità degli operatori

- **PRIORITÀ**: **specifica l'ordine di valutazione degli operatori quando in una espressione compaiono *operatori (infissi) diversi***

- Esempio: `3 + 10 * 20`
  - si legge come 3 + (10 * 20) perché l'operatore * è più prioritario di +

- NB: operatori diversi possono comunque avere *egual priorità*

Associatività degli operatori

- **ASSOCIATIVITÀ**: **specifica l'ordine di valutazione degli operatori quando in una espressione compaiono *operatori (infissi) di egual priorità***

- Un operatore può quindi essere *associativo a sinistra* o *associativo a destra*

- Esempio: `3 - 10 + 8`
  - si legge come (3 - 10) + 8 perché gli operatori - e + sono equiprioritari e **associativi a sinistra**

Priorità e associatività

- Priorità e associatività predefinite possono essere alterate mediante *l'uso di **parentesi** (tonde)*
- Esempio: `(3 + 10) * 20`
  - denota 260 (anziché 203)
- Esempio: `30 - (10 + 8)`
  - denota 12 (anziché 28)

Priorità e associatività

- Gli operatori relazionali hanno **priorità inferiore** agli operatori aritmetici

k < b+3

- equivale a k < (b+3)
- e non a (k<b) + 3

Incremento e decremento

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

- come **pre-operatori**: **++v**
  - *prima incremento e poi uso nell'espressione*
- come **post-operatori**: **v++**
  - *prima uso nell'espressione poi incremento*

Formule equivalenti:

- `v = v + 1;`
- `v +=1`
- `++v`
- `v++`

Cosa stampa?

```
main() {                          Soluzione:
int c;
c=5;
printf("%d\n",c);                      5
printf("%d\n",c++);                         5
printf("%d\n\n",c);                         6
c=5;
printf("%d\n",c);                      5
printf("%d\n",++c);                         6
printf("%d\n",c); }                         6
```

Altri 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 */

int i=4, j, k = 5;
j = i + k++;            /* j vale 9, k vale 6 */

int j, k = 5;
j = ++k - k++;              /* DA NON USARE */
                        /* j vale 0, k vale 7 */
```


Dove si sbaglia frequentemente…

Operazioni matematiche e tipi di dato

- Divisione tra interi e divisione tra reali
  (stesso simbolo /, ma significato differente)
- Significato e uso dell'operatore di modulo (%)
- Operatore di assegnamento (=)
  e operatore di uguaglianza (==)
- Notazione prefissa e postfissa di ++ e -- negli assegnamenti

Riassunto operatori del C

Priorità	Operatore	Simbolo	Associatività
1 (max)	Chiamate a funzioni Selezioni	`()` `[] -> .`	A sinistra

2	Operatori unari:
	▪ op. negazione
	▪ op. aritmetici unari
	▪ op. incr./decr.
	▪ op. indir. e deref.
	▪ op. sizeof
		`!`
		`+ -`
		`++ --`
		`& *`
		`sizeof`
			A destra

3	Op. moltiplicativi	`* / %`	A sinistra
4	Op. additivi	`+ -`	A sinistra

Riassunto operatori del C

Priorità	Operatore	Simbolo	Associatività
5	Op. di shift	`>> <<`	A sinistra
6	Op. relazionali	`> >= <= <`	A sinistra
7	Op. di uguaglianza	`== !=`	A sinistra
8	Op. di AND bit a bit	`&`	A sinistra
9	Op. di XOR bit a bit	`^`	A sinistra
10	Op. di OR bit a bit	`|`	A sinistra
11	Op. di AND logico	`&&`	A sinistra
12	Op. di OR logico	`||`	A sinistra

Riassunto operatori del C

Priorità	Operatore	Simbolo	Associatività
13	Op. condizionale	…? … :…	A destra
14	Op. assegnamento e sue varianti	= += -= *= /= %= &= ^= |= <<= >>=	A destra
15 (min)	Op. di concatenazione	,	A sinistra