Linguaggio C

Gli operatori e le espressioni

(Parte 4)

Prof. Aniello Castiglione

castiglione@ieee.org

Sommario

- Gli operatori e le espressioni
  - Precedenza e associatività
  - Operatori aritmetici
  - Operatori relazionali e logici
  - Operatori per la manipolazione di bit
  - Conversione di tipo e `sizeof`
  - L'operatore condizionale
  - Operatori di accesso alla memoria

Gli operatori e le espressioni

```
/***************************************************************
 File: ope09.c
 Desc: operatore virgola
 Comm:
***************************************************************/
#include <stdio.h>

int main(void)
{
  int a, b;

  a = b = 1, b = b + 2;                              /* 001 */
  printf("a:%d   b:%d\n", a, b);

  a = b = 1, b = b + 2, b = b - 1;                   /* 002 */
  printf("a:%d   b:%d\n", a, b);

  return 0;
}
/******* End **************************************************/
```

Operatori ed espressioni - 1

- Gli operatori sono elementi del linguaggio C che consentono di calcolare valori
- Gli operatori sono rappresentati da uno o più caratteri speciali: nel caso in cui i caratteri siano più di uno, non possono essere separati da spazi
- Gli operatori sono i verbi e gli operandi soggetti ed oggetti della “frase” espressione
- Un’espressione consiste di uno o più operandi e di zero o più operatori

Operatori ed espressioni - 2

Espressioni costanti:
contengono solo valori costanti
	`5`
	`5+6*13/3.0`
	`‘a’`

Espressioni intere:
dopo le conversioni automatiche ed esplicite, producono un risultato di tipo intero
	`j` `j*k`
	`j/k+3` `k-‘a’`
	`3+(int)5.0`

Espressioni floating:
dopo le conversioni automatiche ed esplicite, producono un risultato di tipo floating-point
	`x` `x+3`
	`x/y*5` `3.0`
	`3.0-2` `3+`
	`(float)4`

Espressioni puntatore:
Contengono variabili di tipo puntatore, l’operatore &, stringhe e nomi di array, e producono come risultato un indirizzo di memoria
	`p`
	`&j`
	`p+1`
	`“abc”`

Precedenza e associatività - 1

- Precedenza e associatività sono proprietà degli operatori che regolano le modalità con cui vengono trattati gli operandi
- Gli operatori con precedenza superiore raggruppano a sé i propri operandi rispetto agli operatori con precedenza inferiore, indipendentemente dall'ordine con cui appaiono nell'espressione
- Nel caso in cui gli operatori abbiano la stessa precedenza, viene applicata la proprietà di associatività, per stabilire l'ordine secondo cui gli operandi sono raggruppati con gli operatori
- L'associatività può essere sia sinistra che destra: l'associatività sinistra implica che il compilatore analizza l'espressione da sinistra verso destra, viceversa nell'altro caso

```
a+b-c  /* somma a e b, quindi sottrae c */
a=b=c /* assegna c a b, quindi assegna b ad a */
```

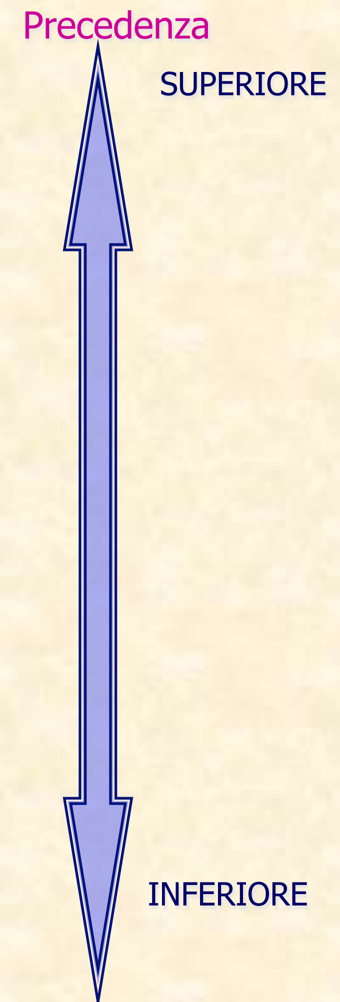
sinistra

destra

Precedenza e associatività - 2

Classe	Operatori	Associatività
primari	() [] → .	Da sinistra a destra
unari	cast sizeof & * - + ~ ++ -- !	Da destra a sinistra
moltiplicativi	* / %	Da sinistra a destra
additivi	+ -	Da sinistra a destra
scorrimento	<< >>	Da sinistra a destra
relazionali	< <= > >=	Da sinistra a destra
uguaglianza	== !=	Da sinistra a destra
AND tra bit	&	Da sinistra a destra
XOR tra bit	^	Da sinistra a destra
OR tra bit			Da sinistra a destra
AND logico	&&	Da sinistra a destra
OR logico	||	Da sinistra a destra
condizionale	?:	Da destra a sinistra
assegnamento	= += -= *= /= %= >>= <<= &= ^= |=	Da destra a sinistra
virgola	,	Da sinistra a destra

Precedenza



Le parentesi - 1

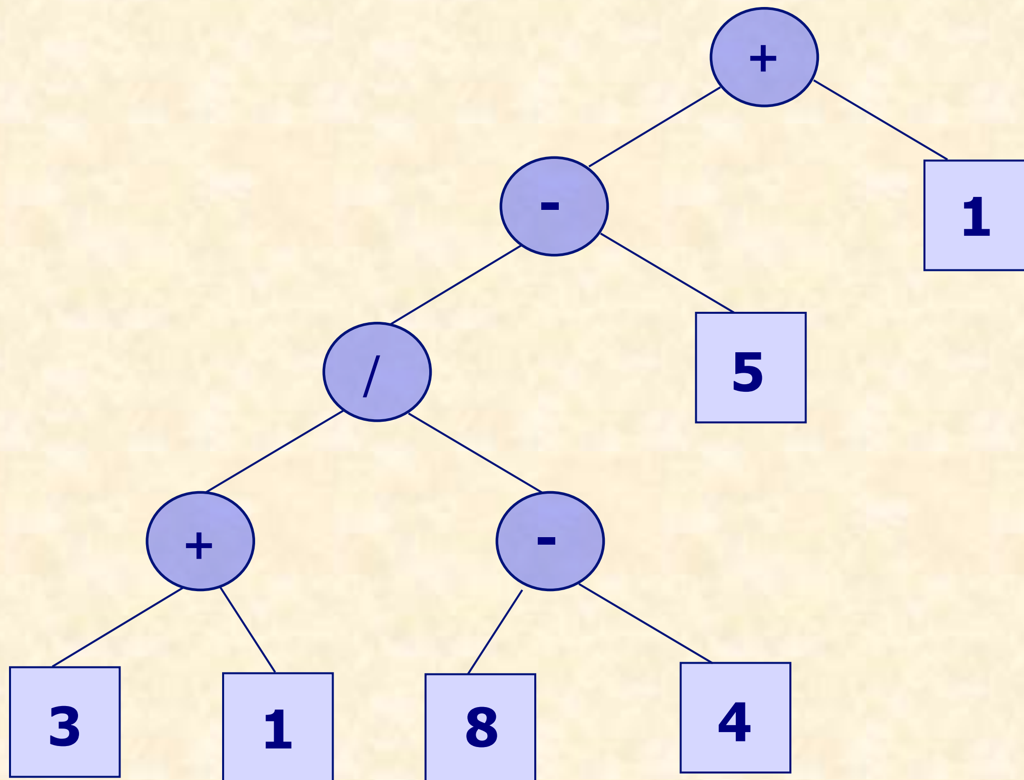
- Per definire specifici ordini di raggruppamento di operandi ed operatori si adoperano le parentesi tonde
- Il compilatore raggruppa per primi gli operandi e gli operatori che compaiono all'interno delle parentesi
- Nel caso di parentesi innestate, il compilatore interpreta per prima l'espressione racchiusa tra le parentesi più interne

Le parentesi - 2

- Per "interpretare" un'espressione, il compilatore crea una struttura ad albero binario:
  - Ogni nodo interno contiene un operatore, ogni foglia un operando
  - L'espressione viene valutata partendo dal livello più basso nell'albero
  - Il risultato della valutazione di ogni combinazione operatore-operandi viene posto nel nodo dell'operatore, che diviene un operando per l'operatore di livello superiore

Le parentesi - 3

La rappresentazione dell'espressione ((3+1)/(8-4)-5)+1 come albero binario



Le sottoespressioni che compaiono al livello più basso dell'albero sono valutabili in qualunque ordine

L’ordine di valutazione

- L’ordine di valutazione degli operatori è indipendente dall’ordine con cui il compilatore raggruppa gli operandi con gli operatori
- Per la maggior parte degli operatori, il compilatore è libero di valutare le sottoespressioni (sinistra e destra) in qualsiasi ordine, eventualmente riorganizzando l’intera espressione, a patto di non alterare il risultato finale
- L’ordine di valutazione può costituire un aspetto critico per le espressioni che implicano effetti collaterali
- La riorganizzazione delle espressioni può causare overflow

Gli operatori aritmetici unari

- Gli operatori + e - sono detti unari perché si applicano ad un solo operando, che può essere un qualunque tipo di variabile intera o floating-point: il tipo del risultato è quello che l'operando assume dopo la promozione ad intero

Operatore	**Simbolo**	**Formato**	**Operazione**
meno unario	-	-x	negazione di x
più unario	+	+x	valore dell'operando

- L'operatore unario - effettua la negazione del suo argomento
- L'unico effetto dell'applicazione del + unario è la promozione ad intero dei tipi interi più corti

Gli operatori aritmetici binari - 1

Operatore	**Simbolo**	**Formato**	**Operazione**
moltiplicazione	*	x*y	x moltiplicato per y
divisione	/	x/y	x diviso per y
resto	%	x%y	resto di x diviso per y
somma	+	x+y	x sommato a y
sottrazione	-	x-y	y sottratto a x

- Gli operandi degli operatori moltiplicativi devono essere di tipo intero o floating-point, mentre gli operatori additivi sono applicabili anche ai puntatori
- Tutti gli operatori aritmetici hanno associatività sinistra

Gli operatori aritmetici binari - 2

- L'operatore resto % (detto anche modulo) è applicabile solo ad operandi interi e fornisce il resto della divisione intera del primo operando per il secondo
- Il risultato di una divisione fra numeri interi effettuata con l'operatore / è un numero intero:
  - Se entrambi sono positivi e non divisibili, la parte frazionaria viene troncata
  - Se uno degli operandi è negativo, il compilatore è libero di arrotondare il valore per eccesso o per difetto

Gli operatori aritmetici binari - 3

- Analogamente, il segno del risultato di un'operazione di resto non è definito nell'ANSI C in presenza di operandi negativi

Regola: a = (a/b)*b + a%b

-5/2 vale -2 oppure -3

-1/-3 vale 0 oppure +1

7%-4 vale 3 oppure -1

Evitare operazioni di / e % fra numeri negativi: il risultato dipende dal compilatore

- Quoziente e resto di una divisione con denominatore nullo sono indefiniti

Gli operatori di assegnamento aritmetico - 1

Operatore	Simbolo	Formato	Operazione
assegnamento	=	a=b	memorizza b in a
somma-assegnamento	+=	a+=b	memorizza a+b in a
sottrazione-assegnamento	-=	a-=b	memorizza a-b in a
moltiplicazione-assegnamento	*=	a*=b	memorizza a*b in a
divisione-assegnamento	/=	a/=b	memorizza a/b in a
resto-assegnamento	%=	a%=b	memorizza a%b in a

- L'operatore di assegnamento ha associatività destra, cosicché l'espressione

  `a=b=c=d=1;`

  viene valutata come

  `(a=(b=(c=(d=1))));`

Gli operatori di assegnamento aritmetico - 2

- Gli operatori di assegnamento aritmetico diminuiscono la possibilità di commettere errori di battitura ed aumentano la leggibilità del codice
- Gli operatori di assegnamento aritmetico, generalmente, aumentano l'efficienza del codice oggetto generato, dato che molti calcolatori hanno istruzioni macchina speciali per realizzare combinazioni di assegnamenti ed operazioni aritmetiche
- Nota: Gli operatori di assegnamento hanno bassa precedenza…

```
j=j*3+4;   ⟶   j=((j*3)+4);
j*=3+4;    ⟶   j=(j*(3+4));
```

Gli operatori di assegnamento aritmetico - 3

- Esempio

  Date le seguenti dichiarazioni:

```
int m=3, n=4;
float x=2.5, y=1.0;
```

Espressione	Espressione equivalente	Risultato
`m += n+x-y`	m = (m+((n+x)-y))	8
`m /= x*n+y`	m = (m/((x*n)+y))	0
`n %= y+m`	n = (n%(y+m))	0
`x += y -= m`	x = (x+(y=(y-m)))	0.5

Gli operatori di incremento e decremento - 1

- Gli operatori di incremento e decremento unitario sono operatori unari
- L'operando deve essere un lvalue scalare (comprese le variabili puntatore)

Operatore	**Simbolo**	**Formato**	**Operazione**
incremento postfisso	++	a++	rende disponibile il valore di a, poi lo incrementa
decremento postfisso	--	a--	rende disponibile il valore di a, poi lo decrementa
incremento prefisso	++	++a	incrementa il valore di a, poi lo rende disponibile
decremento prefisso	--	--a	decrementa il valore di a, poi lo rende disponibile

Gli operatori di incremento e decremento - 2

- Gli operatori postfissi di incremento e decremento accedono al valore di una variabile e ne memorizzano una copia in una locazione temporanea; la variabile viene incrementata, o decrementata, in seguito
- Gli operatori prefissi di incremento e decremento modificano i loro operandi prima che il valore venga usato nelle espressioni

```
main()
{
   int j=5, k=5;

   printf("j: %d\t k: %d\n", j++, k--);
   printf("j: %d\t k: %d\n", j, k);
}
```

j: 5 k: 5
j: 6 k: 4

```
main()
{
   int j=5, k=5;

   printf("j: %d\t k: %d\n", ++j, --k);
   printf("j: %d\t k: %d\n", j, k);
}
```

j: 6 k: 4
j: 6 k: 4

Gli operatori di incremento e decremento - 3

- Esempio

```
x = 1 ;
y = x++ ;
alla fine x vale 2 ed y vale 1
y = x++; equiv  y=x ; x=x+1 ;
```

```
x = 1 ;
y = ++x ;
alla fine x vale 2 ed y vale 2
y = x++; equiv  x=x+1 ; y=x ;
```

Gli operatori di incremento e decremento - 4

- Quando ciò che conta è il solo effetto collaterale, e non il risultato dell'espressione, si possono applicare indifferentemente le due tipologie di operatori di incremento e decremento
- In un assegnamento isolato, come nella terza espressione di un ciclo for, l'effetto collaterale è analogo nel caso di operatori prefissi e postfissi
- Nota: Gli operatori di incremento e decremento hanno la stessa precedenza ed associatività destra, pertanto l'espressione…

```
--j++
```

viene valutata come

```
--(j++)
```

non corretta poiché `j++` non è un lvalue, come richiesto dall'operatore di decremento

Gli operatori di incremento e decremento - 5

- Esempio

  Date le seguenti dichiarazioni:

```
int j=0, m=1, n=-1;
```

Espressione	Espressione equivalente	Risultato
`m++ - --j`	(m++)-(--j)	2
`m += ++j*2`	m = (m+((++j)*2)	3
`m++ * m--`	(m++)*(m--)	Dipendente dalla implementazione (2 o 0)

Gli effetti collaterali - 1

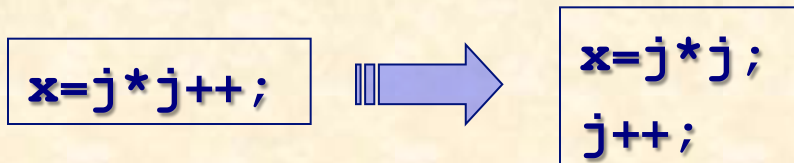
- Gli operatori di assegnamento, di incremento e decremento provocano effetti collaterali, ovvero modificano il valore di una variabile, oltre a produrre un valore come risultato di un'espressione
- L'ordine con cui si manifestano gli effetti collaterali non è predicibile
- Esempio: `x=j*j++;`

  il linguaggio C non definisce l'ordine di valutazione degli operatori moltiplicativi: compilatori diversi potrebbero valutare secondo ordini differenti i due operandi, producendo risultati diversi; se j=5, ad esempio, x=25 valutando prima l'operando di sinistra, 30 valutando prima il destro

Gli effetti collaterali - 2

- Istruzioni che provocano effetti collaterali imprevedibili non sono portabili e vanno evitate
- Il problema degli effetti collaterali si manifesta anche nelle chiamate di funzione, perché l'ANSI C non definisce l'ordine con cui vengono valutati gli argomenti
- Esempio: `f(a,a++)`
- Per prevenire errori dovuti agli effetti collaterali, occorre seguire la regola:

  Se in un'espressione si usa un operatore che implica effetti collaterali, la variabile coinvolta non deve essere usata in altro modo nella stessa espressione



L’operatore virgola - 1

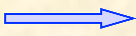
Operatore	Simbolo	Formato	Operazione
virgola	,	a, b	valuta a, poi valuta b, il risultato è b

- L’operatore , consente di valutare due o più espressioni distinte, dove è ammessa un’unica espressione: il risultato è il valore dell’espressione più a destra
- L’operatore virgola può rendere il codice confuso: per convenzione viene principalmente utilizzato nella prima e nella terza espressione dei cicli for
- La virgola può anche essere utilizzata per separare due istruzioni distinte sulla stessa linea: in termini di stile di programmazione è più opportuno porre ciascuna istruzione su una linea diversa

L’operatore virgola - 2

- Esempio

Uso standard dell’operatore virgola



```
for (j=0, k=100; k-j>0; j++, k--)
```

```
j = 0;
k = 100;
while((k-j) > 0)
{
    … … …
    j++;
    k--;
}
```

```
j = 0, k = 100;
while((k-j) > 0)
{
    … … …
    j++, k--;
}
```

Stile di programmazione involuto

Gli operatori relazionali - 1

Operatore	**Simbolo**	**Formato**	**Operazione**
maggiore di	>	a>b	1 se a è maggiore di b, 0 altrimenti
minore di	<	a<b	1 se a è minore di b, 0 altrimenti
maggiore o uguale di	>=	a>=b	1 se a è maggiore o uguale a b, 0 altrimenti
minore o uguale di	<=	a<=b	1 se a è minore o uguale a b, 0 altrimenti
uguaglianza	==	a==b	1 se a è uguale a b, 0 altrimenti
disuguaglianza	!=	a!=b	1 se a è diverso da b, 0 altrimenti

- Gli operatori relazionali di disuguaglianza **>**, **<**, **>=**, **<=** hanno lo stesso livello di precedenza, mentre gli operatori **==** e **!=** (detti di uguaglianza) hanno precedenza inferiore

Gli operatori relazionali - 2

- Gli operatori relazionali hanno precedenza inferiore rispetto agli operatori aritmetici: l'espressione **`a+b*c<d/f`** viene valutata come **`(a+(b*c))<(d/f)`**
- Esempio

  Date le seguenti dichiarazioni:

```
int j=0, m=1, n=-1;
float x=2.5, y=0.0;
```

Espressione	Espressione equivalente	Risultato
`y>=n<=m`	`((y>=n)<=m)`	1
`j<=x==m`	`((j<=x)==m)`	1
`-x+j==y>n>=m`	`((-x)+j)==((y>n)>=m)`	0
`x+=(y>=n)`	`x=(x+(y>=n))`	3.5
`++j==m!=y*2`	`((++j)==m)!=(y*2)`	1

Gli operatori relazionali - 3

- Avvertenza: Il confronto di uguaglianza fra operandi floating-point è molto pericoloso a causa dell’approssimazione insita nel tipo di rappresentazione
- Esempio: L’espressione

```
(1.0/3.0+1.0/3.0+1.0/3.0)==1.0
```

anche se algebricamente vera, viene valutata come falsa sulla maggior parte dei calcolatori; infatti, il risultato della divisione 1.0/3.0 non può essere rappresentato esattamente: la somma a sinistra dell’espressione non coincide con 1

⇨ Evitare confronti di uguaglianza fra floating-point per salvarsi da errori dovuti al tipo di rappresentazione

Uguaglianza fra floating-point

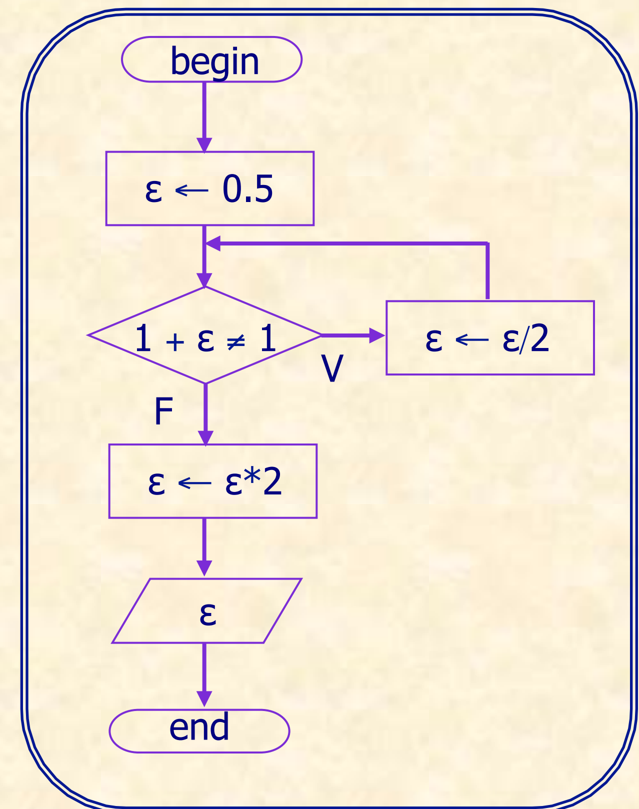
- In generale, due numeri floating-point, a e b, saranno uguali se

$$|a-b|<\varepsilon$$

- ε è la precisione di macchina, il numero più piccolo che, sommato ad 1, viene "percepito" dall'elaboratore; ovvero ε è il numero più piccolo tale che, nell'aritmetica dell'elaboratore,

$$1+\varepsilon \neq 1$$

- In singola precisione (sulla macchina di riferimento a 32 bit), $\varepsilon \approx 1.0e-6$, in doppia $\varepsilon \approx 1.0e-13$



Gli operatori logici - 1

- Dal punto di vista algebrico, l'espressione

  $$x<y<z$$

  è vera se y è maggiore di x e minore di z; in C, invece, verrebbe valutata come $(x<y)<z$, vera se $x<y$ e $z>1$ o $x>y$ e $z>0$
- L'equivalente C di $x<y<z$ è

  ```
  (x<y) && (y<z)
  ```

  con && operatore AND logico

<table>
<tr><th>Operatore</th><th>Simbolo</th><th>Formato</th><th>Operazione</th></tr>
<tr><td>AND logico</td><td>&&</td><td>a&&b</td><td>1 se a b sono diversi da 0, 0 altrimenti</td></tr>
<tr><td>OR logico</td><td>||</td><td>a||b</td><td>1 se a o b sono diversi da 0, 0 altrimenti</td></tr>
<tr><td>negazione logica</td><td>!</td><td>!a</td><td>1 se a vale 0, 0 altrimenti</td></tr>
</table>

Gli operatori logici - 2

- La negazione logica ha precedenza maggiore di AND, che ha precedenza maggiore di OR
- Gli operatori logici sono applicabili ad operandi di tipo intero e floating-point
- Per gli operatori logici, l'ordine di valutazione degli operandi da parte del compilatore è definito: da sinistra verso destra
- Inoltre, il compilatore non valuta un operando quando non sia necessario
- Esempio:

```
if((a != 0) && (b/a == 6))
```

se **`a`** è uguale a zero, l'espressione **`(b/a == 6)`** non viene valutata

Gli operatori logici - 3

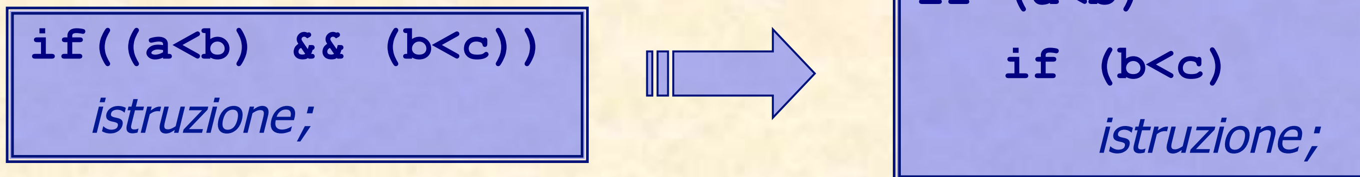
- Esempio
  Date le seguenti dichiarazioni:

```
int j=0, m=1, n=-1;
float x=2.5, y=0.0;
```

<table>
<tr><th>Espressione</th><th>Espressione equivalente</th><th>Risultato</th></tr>
<tr><td>j<=10 && x>=1 && m</td><td>((j<=10) && (x>=1)) && m</td><td>1</td></tr>
<tr><td>!x || !n || m+n</td><td>((!x) || (!n)) || (m+n)</td><td>0</td></tr>
<tr><td>x*y<j+m || n</td><td>((x*y)<(j+m)) || n</td><td>1</td></tr>
<tr><td>(x>y)+!j || n++</td><td>((x>y)+(!j)) || (n++)</td><td>1</td></tr>
<tr><td>(j || m)+(x || ++n)</td><td>(j||m)+(x||(++n))</td><td>2</td></tr>
</table>

Gli operatori logici - 4

- Un'espressione relazionale complessa viene normalmente inserita nella parte condizionale di if e cicli
- Collegare espressioni con l'operatore AND equivale ad usare if innestati



- Avvertenza: Poiché il compilatore valuta solo le espressioni necessarie alla determinazione del valore di verità dell'espressione condizionale, nel caso…

```
if ((a<b) && (c==d++))
```

d viene incrementato solo se **a** è minore di **b**

⇨ Evitare l'uso di operatori che implicano effetti collaterali nelle espressioni relazionali

Gli operatori di manipolazione di bit

- Gli operatori di manipolazione di bit permettono l'accesso a specifici bit all'interno di oggetti di tipo intero ed il confronto di sequenze di bit tra coppie di oggetti di tipo intero

Operatore	**Simbolo**	**Formato**	**Operazione**
scorrimento a destra	>>	x>>y	x scorre a destra di y bit
scorrimento a sinistra	<<	x<<y	x scorre a sinistra di y bit
AND tra bit	&	x&y	AND bit a bit tra x e y
OR tra bit	|	x|y	OR bit a bit tra x e y
XOR tra bit	^	x^y	XOR bit a bit tra x e y
complemento	~	~x	complemento dei bit di x

Gli operatori di scorrimento - 1

- Gli operatori di scorrimento, o shift, permettono lo scorrimento dei bit di un oggetto di un numero di posizioni specificato
- Gli operandi devono essere di tipo intero e viene effettuata la loro promozione intera automatica
- Il tipo del risultato coincide con il tipo dell'operando di sinistra dopo la promozione intera
- Lo scorrimento a sinistra equivale alla moltiplicazione per le potenze di due

  `x << y` è equivalente a $x*2^y$

- Lo scorrimento verso destra di numeri non negativi equivale alla divisione per le potenze di due

  `x >> y` è equivalente a $x/2^y$

Gli operatori di scorrimento - 2

- Quando i bit di un numero positivo scorrono verso destra o verso sinistra, i bit che vengono a mancare sono riempiti con zero
- Nel caso di spostamento a destra di numeri negativi, i bit che vengono a mancare possono essere riempiti con uno o con zero, producendo uno *scorrimento aritmetico* o *logico*, rispettivamente

Espressione	Operando sinistro	Risultato	Valore
255 >> 3	0000000011111111	0000000000011111	31
5 << 1	0000000000000101	0000000000001010	10
1 << 15	0000000000000001	1000000000000000	-2^{15}
-5 >> 2	1111111111111011	0011111111111110	$2^{14}-2$
-5 >> 2	1111111111111011	1111111111111110	-2

Gli operatori di scorrimento - 3

- Lo standard ANSI non specifica se il compilatore deve effettuare scorrimento logico o aritmetico sui numeri con segno
- Tuttavia, se l'operando di sinistra è unsigned, il compilatore deve effettuare uno scorrimento logico (l'istruzione è portabile)
- Si ottengono risultati impredicibili quando…
  - …l'operando di destra è più grande della dimensione dell'oggetto su cui si effettua lo scorrimento
  - …l'operando di destra assume valore negativo

Gli operatori logici tra bit - 1

- Gli operatori logici tra bit sono simili a quelli booleani, ma operano a livello dei singoli bit degli operandi
- Nelle espressioni che contengono operatori a livello di bit, le costanti sono normalmente scritte in notazione esadecimale, per facilitare l'individuazione del valore di ogni bit

Espressione	Valore esadecimale	Valore binario
9430	0x24D6	0010010011010110
5722	0x165A	0001011001011010
9430 & 5722	0x0452	0000010001010010

Esempio di applicazione dell'operatore AND tra bit

Gli operatori logici tra bit - 2

OR

Espressione	Valore esadecimale	Valore binario
9430	0x24D6	0010010011010110
5722	0x165A	0001011001011010
9430 | 5722	0x36DE	0011011011011110

XOR

Espressione	Valore esadecimale	Valore binario
9430	0x24D6	0010010011010110
5722	0x165A	0001011001011010
9430 ^ 5722	0x328C	0011001010001100

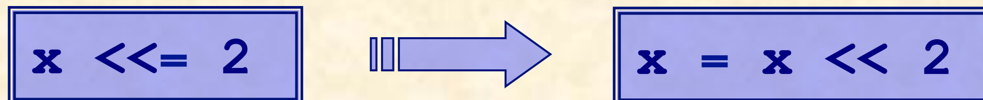
NOT

Espressione	Valore esadecimale	Valore binario
9430	0x24D6	0010010011010110
~9430	0xDB29	1101101100101001

Gli operatori di assegnamento bit a bit

Operatore	**Simbolo**	**Formato**	**Operazione**
scorrimento a destra con assegnamento	>>=	a>>=b	assegna a>>b ad a
scorrimento a sinistra con assegnamento	<<=	a<<=b	assegna a<<b ad a
AND con assegnamento	&=	a&=b	assegna a&b ad a
OR con assegnamento	|=	a|=b	assegna a|b ad a
XOR con assegnamento	^=	a^=b	assegna a^b ad a

- Gli operatori di assegnamento bit a bit sono analoghi agli operatori di assegnamento aritmetico



L'operatore di conversione di tipo - 1

Operatore	Simbolo	Formato	Operazione
conversione di tipo, cast	(tipo)	(tipo)e	converte l'espressione e nel tipo indicato

- Viene utilizzato spesso per la promozione di un numero intero in floating-point, per garantire che il risultato di una divisione non venga troncato
- L'operatore di conversione di tipo ha precedenza superiore a quella degli operatori aritmetici
- È possibile applicare l'operatore di conversione di tipo anche agli argomenti di funzione

L'operatore di conversione di tipo - 2

- Esempio: scrivere un programma che stampa le potenze del 2 fino a 2^{32}

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main()
{
   int j;
   long k;

   for (j=0; j<=32; j++)
   {
       k = (int) pow(2.0, (double)j);
       printf("%4d\t\t%13lu\n", j, k);
   }
   exit(0);
}
```

La funzione di libreria **`pow()`** gestisce solo argomenti di tipo double: occorre eseguire un cast prima di passare alla funzione l'argomento intero **`j`**

Il valore restituito da **`pow()`** è un double e quindi deve essere convertito ad intero prima dell'assegnamento a **`k`**: tale conversione avverrebbe comunque implicitamente, la presenza del cast ha carattere documentativo

L'operatore `sizeof` - 1

Operatore	Simbolo	Formato	Operazione
dimensione di	sizeof	sizeof(t) o sizeof e	ritorna la dimensione in byte del tipo di dati t o dell'espressione e

- Accetta come operandi un tipo di dati o un'espressione (che non sia una funzione o un void)
- L'espressione non viene valutata dal compilatore, che determina solo il tipo del risultato: se l'espressione implica effetti collaterali, tali effetti non si manifestano
- Lo standard ANSI richiede che il risultato di **`sizeof`** sia un tipo unsigned

```
/* Ritorna la dimensione di un int */
   sizeof(3+5);
```

```
/* Ritorna la dimensione di un double */
   sizeof(3.+5);
```

L'operatore **sizeof** - 2

- Nel caso delle espressioni, le parentesi sono opzionali, anche se vengono normalmente impiegate
- Se l'operando di **sizeof** è un tipo, le parentesi sono obbligatorie ed il risultato è la lunghezza in byte degli oggetti di quel tipo
- L'operatore **sizeof** viene usato per trovare la dimensione dei tipi di dati composti, come array e strutture
- Può essere anche utilmente impiegato per acquisire informazioni sulle dimensioni dei tipi di dati nello specifico ambiente C

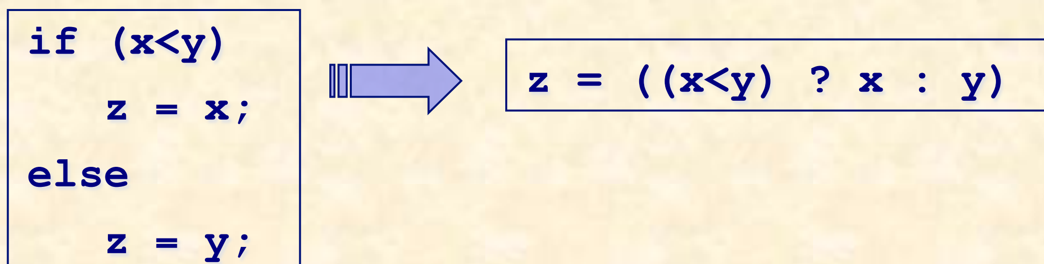
```
#include <stdio.h>
#include <stdlib.h>

main()
{
  printf("TIPO\tDIMENSIONE\n\n");
  printf("char\t\t%d\n",sizeof(char));
  printf("short\t\t%d\n",sizeof(short));
  printf("int\t\t%d\n",sizeof(int));
  printf("float\t\t%d\n",sizeof(float));
  printf("double\t\t%d\n",sizeof(double));
  exit(0);
}
```

L’operatore condizionale - 1

Operatore	Simbolo	Formato	Operazione
condizionale	?:	a?b:c	se a è diverso da zero il risultato è b, altrimenti il risultato è c

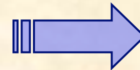
- L’operatore ?: è l’unico operatore ternario del C e rappresenta una forma abbreviata per il costrutto **`if…else`**
- Il primo operando di ?: è la condizione scalare, il secondo ed il terzo operando rappresentano il valore finale dell’espressione
- Il secondo e il terzo operando possono essere di qualsiasi tipo, purché compatibili secondo le usuali regole di conversione



L’operatore condizionale - 2

- A causa della difficoltà di interpretazione, è bene impiegare l’operatore condizionale con attenzione: comunque esistono situazioni in cui è utile per evitare la ridondanza

```
if (j<0)
   printf("Ecco %d", j);
else
   printf("Ecco %d", k);
```



```
printf("Ecco %d", j<0 ? j : k);
```

Gli operatori di gestione della memoria

Operatore	Simbolo	Formato	Operazione
indirizzo di	&	&x	restituisce l'indirizzo di x
accesso all'indirizzo	*	*a	restituisce il valore dell'oggetto memorizzato all'indirizzo contenuto in a (puntato da a)
elemento di array	[]	x[5]	restituisce il valore dell'elemento 5 dell'array
punto	.	x.y	restituisce il valore del campo y della struttura x
freccia destra	→	p → y	restituisce il valore del campo y della struttura puntata da p

Esempio 1: Test di primalità

```
/*  Test di primalità: il programma richiede l'inserimento di un intero n>1 e stampa
**  se è o non è primo. L'algoritmo utilizzato è quello delle divisioni successive,
**  non particolarmente efficiente, ma di facile implementazione */
#include <stdio.h>
#include <stdlib.h>
main()
{
    int i=2, n;
    do
    {
         printf("Digita un numero maggiore di 1: ");
         scanf("%d", &n);
    } while (n <= 1);
    while ((n%i != 0) && (i <= n/2))
         i++;
    if (i==n/2+1)
         printf("Il numero %d è primo\n", n);
    else
         printf("Il numero %d non è primo\n", n);
    exit(0);
}
```

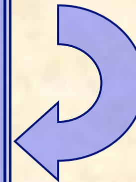
Esempio 2

- Problema

  Scrivere un programma che accetta in input una stringa di 32 bit, che costituiscono le risposte ad un test (sì/no) e calcola, in base al risultato corretto del test, il punteggio ottenuto dal candidato

```
#include <stdio.h>
#include <stdlib.h>
main()
{
   extern double grade_test();
   extern long get_answers();
   double grade;
   printf("Introdurre le risposte:\n");
   grade = grade_test(get_aswers());
   printf("La valutazione è %3.0f\n", grade);
   exit(0);
}
```

Lo specificatore `%3.0f` fa sì che si stampino almeno tre cifre, arrotondando le cifre decimali



Esempio 2 (continua)

```
#include <stdio.h>

/* Legge e memorizza le risposte inserite
 * dal candidato */
long get_answers()
{
     long answers=0;
     int j;
     char c;

     for (j=0; j<=31; j++)
     {
         scanf("%c", &c);
         if  (c=='y' || c=='Y')
              answers |= 1<<j;
     }
     printf("Risposte fornite = (%lx)", answers);
     return answers;
}
```

Per ogni risposta positiva, il bit appropriato viene posto al valore 1

Operando un OR tra il valore corrente di **`answer`** e l'espressione **`1<<j`**, ad ogni iterazione, si ottiene infine che tutti i bit corrispondenti a risposte positive vengano posti ad 1

Esempio 2 (continua)

```
#define CORRECT_ANSWERS 0x3A7C2FB5
/* Le risposte esatte sono
 * nnyy ynyn nyyy yynn nnyn yyyy ynyy nyny
 * 0011 1010 0111 1100 0010 1111 1011 0101
 */

double grade_test(answers)
long answers;
{
   extern int count_bits();
   long wrong_bits;
   double grade;

   wrong_bits = answers^CORRECT_ASWERS;
   grade = 100*((32.0-count_bits(wrong_bits))/32.0);
   return grade;
}
```

```
int count_bits(long_num)
long long_num;
{
   int j, count=0;

   for (j=0; j<=31; j++)
   if  (long_num & (1<<j))
        ++count;
   return count;
}
```

Calcola il numero di risposte errate (utilizzando una operazione di XOR bit a bit) e produce una valutazione in centesimi