UNIMORE

UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

3 - Espressioni e Operatori in C

Programmazione 2 - [MN1-1141]

Corso di Laurea in INFORMATICA
Anno accademico 2024/2025

Dr. Alessandro Capotondi
alessandro.capotondi@unimore.it

Risorse di riferimento

Libro di testo “Programmare in C”

- Cap. 11 (sezioni 11.1 – 11.6)

Espressioni in C

Espressione

Una espressione è una notazione che denota un valore mediante un processo detto valutazione

- Notazione = testo con una sintassi formale
- La valutazione equivale a calcolare l'espressione
- "Denota" significa che produce un risultato (riutilizzabile)

Intuitivamente: sono le espressioni delle scuole medie

- Permettono di svolgere calcoli
- Si basano grossomodo sugli stessi concetti

Espressioni Semplici in C

Ci sono due categorie di espressioni in C:

- Espressioni semplici (o elementari)
- Espressioni composte (combinazione di altre espressioni)

Quali espressioni elementari sono disponibili in C?

Ne abbiamo già visto un esempio: le costanti

```
10
'a'
-0.1
```

Un secondo tipo è dato dalle variabili (che vedremo tra poco)

Chiamate a Funzione in C

- Ci sono due modi di definire espressioni composte in C
- Si possono usare chiamate a funzione, oppure operatori

Una chiamata a funzione ha la sintassi seguente:

```
<nome funzione>(<argomenti>)
<argomenti> ::= <espr.> | <espr.> {, <espr.>}
```

- Le parentesi tonde fanno parte della sintassi
- La chiamata può avere zero o più argomenti (espressioni)
- Se ci sono più argomenti, vanno separati da virgole

Chiamate a Funzione in C

- Ci sono due modi di definire espressioni composte in C
- Si possono usare chiamate a funzione, oppure operatori

Una chiamata a funzione ha la sintassi seguente:

```
<nome funzione>(<argomenti>)
<argomenti> ::= <espr.> | <espr.> {, <espr.>}
```

Qualche esempio:

```
arcsin(0.5)
pow(2, 4)
sqrt(9)
```

Chiamate a Funzione in C

Per quanto riguarda la semantica di una chiamata a funzione:

```
<nome funzione>(<argomenti>)
<argomenti> ::= <espr.> | <espr.> {, <espr.>}
```

- Esegue un algoritmo di nome `<nome funzione>`
- Con i parametri di ingresso specificati tra parentesi
- Denota (restituisce) il risultato

I parametri sono altre espressioni!

```
power(sin(0.5), 2)
sqrt(power(3, 3))
```

Operatori in C

Un operatore in C è una funzione con sintassi specializzata
Si usano per le operazioni più comuni. E.g.

```
10 + 2
2 * 3
12 / 4
3 + cos(3.14)
```

Si comportano come le funzioni:

- I parametri si dicono operandi
- Quando vengono valutati denotano (restituiscono) un risultato
- Hanno una implementazione specializzata (molto efficiente)

Classificazione degli Operatori

Gli operatori sono classificati secondo due criteri:

- In base al tipo degli operandi e del risultato
- In base al numero degli operandi

Tipo di operandi/risultato	Numero di operandi
• Aritmetici • Relazionali (di confronto) • Logici • Condizionali • ...	• Unari • Binary • Ternari • ...

Operatori Aritmetici in C

Operandi numerici, risultato numerico

Simbolo	Operazione	Numero di operandi
-	Inversione di segno	unario
+	somma	binario
-	differenza	binario
*	moltiplicazine	binario
/	divisione tra interi	binario
/	divisione in virgola mobile	binario
%	modulo (resto della divisione intera)	binario

- La divisione fra interi è quella delle elementari
  - E.g. 7/3 → 2
- Il modulo è il resto della divisione intera:
  - E.g. 7 % 3 → 1

Operatori Aritmetici in C

Qualche esempio:

- `3 * 4`
- `-(2 * 2)`
- `7+4`
- `10 % 4`

Operatori Aritmetici in C

Qualche esempio:

- `3 * 4` denota 12
- `-(2 * 2)` denota -4
- `7+4` denota 11
- `10 % 4` denota 2

Overloading (Sovraccarico) degli Operatori

In C, lo stesso simbolo può essere associato ad operatori diversi

Un primo esempio:

- Inversione di segno (unario):

```
-5
```

- Differenza (binario)

```
10 - 7
```

Secondo esempi:

- Divisione intera (se gli operandi sono interi)

```
10 / 4 (denota 2)
```

- Divisione in virgola mobile (almeno un operando reale)

```
10.0 / 4.0 (denota 2.5)
16 / 5.0 (denota 3.2)
3F / 4 (denota 0.75)
```

Espressioni Omogenee ed Eterogenee

In C una espressione si dice:

- Omogenea se gli operandi sono tutti dello stesso tipo
- Eterogenea se gli operandi sono di tipo diverso

In caso di espressioni eterogenee:

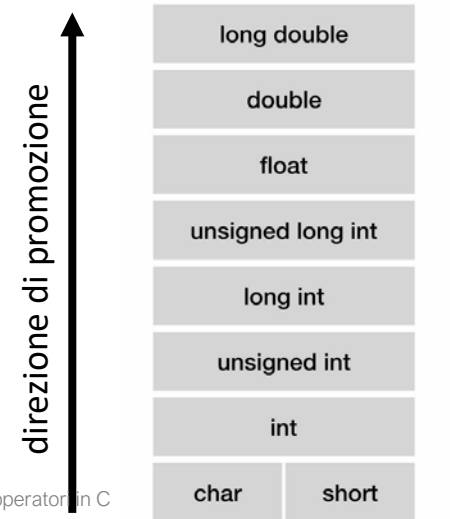
- Il linguaggio tenta una promozione (conversione) di tipo
- Se l'espressione è diventata omogenea, viene valutata
- Altrimenti si tenta una nuova promozione
- Se questo non è fattibile il processo fallisce

Espressioni Omogenee ed Eterogenee

Regole di promozioni di tipo

- Idea: da tipi meno espressivi a tipi più espressivi
- Il risultato è del tipo dell'espressione dopo le promozioni

Nel dettaglio, la catena di promozione è questa:



Espressioni Omogenee ed Eterogenee

Esempio di promozione di tipo:

```
(3 * 2.0) / 5
```

- 3 viene convertito in double (come 2.0)
- Viene valutato il *: il risultato è 6.0 (double)
- 5 viene convertito in double (come 6.0)
- Viene valutato il /: il risultato è 1.2 (double)

Espressioni Omogenee ed Eterogenee

Un secondo esempio di promozione di tipo:

```
(3 / 2) * 5.0
```

- Viene eseguita la divisione intera 3 / 2
- Il risultato è 1 (intero)
- 1 (intero) viene promosso a 1.0 (double)
- 1.0 viene moltiplicato per 5.0
- Il risultato è 5.0

Casting

È possibile <u>forzare</u> una particolare conversione

Basta usare l'operatore di cast, con sintassi:

```
( <tipo> )  <espressione>
```

Esempi:

- `7 / 3` denota 2
- `((float) 7) / 3` denota 2.5

Non si usa spesso, ma in alcuni casi è utile

Casting: facoltativo?

```
int main(){
    int a=5;
    float b;
    b = a;
    return 0;
}
```

Riusciamo a compilare questo programma?
Forse sì! Ma....

...meglio metterlo

```
int a;
float b;
b = (float) a;
```

È meglio indicare l'operazione di casting per evitare comportamenti inattesi da parte del compilatore

Provate a impostare **`-Wconversion`** fra i flag del compilatore

Casting impliciti: consigli

- È sempre bene indicare **esplicitamente il casting** delle variabili
  - Ci permette di controllare da vicino il tipo di operazione che vogliamo sia eseguita per davvero
  - Rende il codice più leggibile
  - **Evitiamo di capitare in diversi casi di errore**

Operazioni e promozioni

- Se vogliamo essere “sicuri”, possiamo imporre il casting su entrambe le variabili

```
int a=2,b=5;
float result;

result = (float)b / (float)a;
```

Utilizzo dei tipi appropriati nelle operazioni [1]


```
unsigned a, b, r;
a = 1;
b = 2;
r = b - a;
```

Il risultato dell'operazione è un numero negativo!

Come viene gestita questa situazione?

Utilizzo dei tipi appropriati nelle operazioni [2]


```
unsigned a, b, r;
a = 1;
b = 2;
r = a - b;
```

Il risultato dell'operazione è un numero negativo!

La variabile unsigned il valore che avrebbe un intero signed della stessa dimensione (int)

Non possiamo affidarci al risultato di questa operazione!

Dobbiamo gestire la conversione fra due tipi di dati numerici differenti <u>prima</u> di effettuare l'operazione

Casting fra variabili unsigned e signed

- Se dobbiamo "convertire" variabili *unsigned* in *signed* e rischiamo di superare il range di valori consentito, dobbiamo *promuoverle* ai corrispondenti tipi di *dimensione maggiore per evitare errori*

```
unsigned a, b;
long r;
a = 1;
b = 2;
r = (long)b - a;
```

Operatori Relazionali (di Confronto) in C

Operandi numerici, risultato logico

Simbolo	Operazione	Numero di operandi
==	Uguaglianza	binario
!=	Diversità	binario
>	Maggiore di	binario
<	Minore di	binario
>=	Maggiore o uguale a	binario
<=	Minore o uguale a	binario

I valori logici sono gestiti mediante numeri!

- Denotano 0 se il confronto è falso
- Denotano un valore diverso da 0 se è vero
  - Di solito 1 o -1

Operatori Relazionali (di Confronto) in C

Qualche esempio:

- `1 == 1`
- `4 < 2`
- `3 >= 3`
- `1 != 2`

Operatori Relazionali (di Confronto) in C

Qualche esempio:

- `1 == 1` denota "vero" (i.e. un numero diverso da 0)
- `4 < 2` denota "falso" (i.e. il numero 0)
- `3 >= 3` denota "vero"
- `1 != 2` denota "vero"

Operatori Logici in C

Operandi logici, risultato logico

Simbolo	Operazione	Numero di operandi
!	Negazione	unario
&&	and (e)	binario
||	or (o)	binario

- Di nuovo: i valori logici sono gestiti mediante numeri
- Sono utili per controllare condizioni complesse:

```
(2 < 3) || (3 != 0)
(2 <= 3) && (3 <= 4)
```

Operatori Logici in C

Qualche esempio:

- `(1 <= 2) && (2 <= 3)`
- `(1 <= 2) && (3 >= 2)`
- `(1 <= 2) || (3 >= 2)`
- `!(1 == 1)`

Operatori Logici in C

Qualche esempio:

- `(1 <= 2) && (2 <= 3)` denota "vero"
- `(1 <= 2) && (3 >= 2)` denota "falso
- `(1 <= 2) || (3 >= 2)` denota "vero"
- `!(1 == 1)` denota "falso"

Occhio a Questi Tre

Quattro operatori a cui fare <u>attenzione</u>:

Operatore di uguaglianza: ==

- Non si usa =, perché è riservato per l'assegnamento
- Ne parleremo tra poco

Operatori and e or: && e ||

- Esistono anche & e | (simbolo ripetuto una solta volta)
- ...Ma fanno tutt'altro (non ne parleremo)

Operatore ^:

- Non è l'elevamento a potenza, ma tutt'altro!

Priorità degli Operatori

La 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 avere egual priorità

Associatività degli Operatori

L'associatività specifica l'ordine di valutazione degli operatori
...quando compaiono operatori (infissi) di egual priorità

- Un operatore può essere associativo a sinistra o a destra

Esempio:

```
3 - 10 + 8
```

- Si legge come (3 - 10) + 8
- ...Perché gli operatori - e + sono equiprioritari e associativi a sinistra

Priorità, Associatività e Parentesi

- Priorità ed associatività servono per interpretare le espressioni
- ...Ma possono essere alterate mediante l'uso delle parentesi

Esempio:

```
(3 + 10) * 20
```

- Denota 260 (anziché 203)

Esempio:

```
30 - (10 + 8)
```

- Denota 12 (anziché 28)

Priorità ed Associatività degli Operatori in C

- In generale valgono le regole della matematica

Priorità	Simbolo	Operazione	Associatività
1	`()`	Chiamata a funzione	sx
1	`[]`	Indicizzazione	sx
1	`.`	Selezione (strutture)	sx
1	`->`	Selezione (puntatori a struttura)	sx
2	`!`	Negazione	dx
2	`+`	Mantenimento di segno	dx
2	`-`	Inversione di segno	dx
2	`++`	Incremento (postfisso e prefisso)	dx
2	`--`	Decremento (postfisso e prefisso)	dx
2	`*`	Indirizzamento	dx
2	`&`	Dereferenziamento	dx
2	`sizeof`	Dimensione	dx

operatori unari

Priorità ed Associatività degli Operatori in C

moltiplicatvi

Priorità	Simbolo	Operazione	Associatività
3	*	Moltiplicazione	sx
3	/	Divisione (intera)	sx
3	/	Divisione (virgola mobile)	sx
3	%	Modulo	sx
4	+	Somma	sx
4	-	Sottrazione	sx
5	<<	Shift sx	sx
5	>>	Shift dx	sx
6	<	Minore	sx
6	>	Maggiore	sx
6	<=	Minore o uguale	sx
6	>=	Maggiore o uguale	sx

Priorità ed Associatività degli Operatori in C

Priorità	Simbolo	Operazione	Associatività
7	==	Uguaglianza	sx
7	!=	Diverso	sx
8	&	AND bit a bit	sx
9	^	XOR bit a bit	sx
10	|	OR bit a bit	sx
11	&&	AND logico	sx
12	||	OR logico	sx
13	? :	Condizionale	dx
14	=	Assegnamento (incl. +=, -=, etc.)	dx
15	,	Concatenazione	sx

- La lista è abbastanza completa
- Include operatori che non useremo nel corso

Espressioni & Istruzioni

Una espressione seguita da ";" è una istruzione

- Quando viene eseguita viene valutata
- ...E non fa altro (non è particolarmente utile)

Qualche esempio (sano):

```
int main() {
    3 * 2 - 1;
    7 / 2 * 2;
    7 / 2.0 * 2;
    7 / 2 * 2.0;
    13 % 2 == 1;
    1 <= 3 && 2 <= 4;
}
```

Espressioni & Istruzioni

Una espressione seguita da ";" è una istruzione

- Quando viene eseguita viene valutata
- ...E non fa altro (non è particolarmente utile)

Qualche esempio (sano):

```
int main() {
    3 * 2 - 1;          // 6 - 1 → 5
    7 / 2 * 2;          // 3 * 2 → 6
    7 / 2.0 * 2;        // 3.5 * 2 → 7.0 (double)
    7 / 2 * 2.0;        // 3 * 2.0 → 6.0 (double)
    13 % 2 == 1;        // 1 == 1→ vero
    1 <= 3 && 2 <= 4;   // vero && vero → vero
}
```

Altre “Operazioni” (aka funzioni)

- Altre operazioni matematiche sulle variabili possono essere effettuate tramite chiamate a funzioni di librerie standard o esterne
- Ad esempio, la standard library del C mette a disposizione funzioni per il calcolo di elevamento a potenza e radici
  - necessario includere math.h e linkare la librerie libm *(ci torniamo a breve)*

Libreria matematica. Esempio elevato a potenza di n

1. ATTENZIONE: Non confondiamoci con l'operatore di XOR

~~a^2~~

Non è Matlab...
...e non esiste **a******2** *come in python*

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

int main(){
   float r, a=5;
   r = powf(a, 2);
   printf("r = %f\n", r);
}
```

Si usa una funzione apposita presente fra le funzioni matematiche della libreria standard del C

Libreria matematica. Esempio elevato a potenza di n


```
#include<stdio.h>
#include<math.h>
int main(){
    float r, a=5;  r =
    powf(a, 2);
   printf("r = %f\n", r);
}
```

Reference documentation
https://en.cppreference.com/w/c/numeric/math/pow

Se provate a compilare ottenete un errore:

```
/home/luca/work/papers/byod/code/hello/main.c:6:
undefined reference to `powf'
collect2: error: ld returned 1 exit status
```

Libreria matematica. Compilazione e Linking

```
#include<stdio.h>
#include<math.h>
int main(){
    float r, a=5;  r =
    powf(a, 2);
   printf("r = %f\n", r);
}
```

Reference documentation
https://en.cppreference.com/w/c/numeric/math/pow

Se provate a compilare ottenete un errore:

```
/home/luca/work/papers/byod/code/hello/main.c:6:
undefined reference to `powf'
collect2: error: ld returned 1 exit status
```

Libreria matematica. Compilazione e Linking [1]

```
#include<stdio.h>
#include<math.h>
int main(){
    float r, a=5;  r =
    powf(a, 2);
   printf("r = %f\n", r);
}
```

Reference documentation
https://en.cppreference.com/w/c/numeric/math/pow

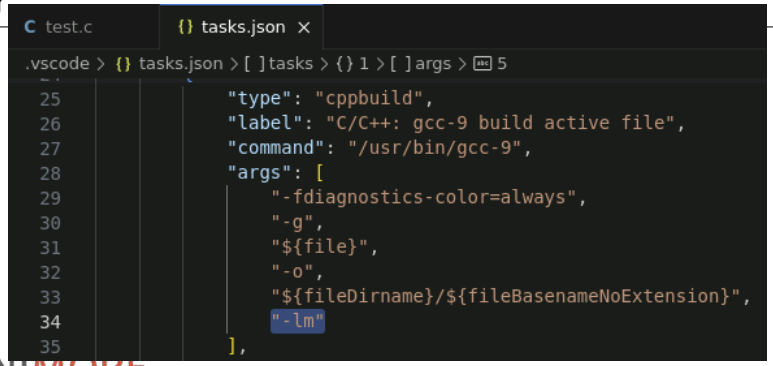
Se provate a compilare ottenete un errore:

```
/home/luca/work/papers/byod/code/hello/main.c:6:
undefined reference to `powf'
collect2: error: ld returned 1 exit status
```

Libreria matematica. Compilazione e Linking [2]

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

int main(){
    float r, a=5;
    r = powf(a, 2);
    printf("r = %f\n",
    r);
}
```



Non basta "importare" la libreria math.h tramite la direttiva include

*Dobbiamo anche configurare il **linking** della libreria al nostro programma nel file aggiungendo "–lm" nella linea di compilazione.*

(in VSCode bisogna aggiungere "–lm" al file task.json)

Operatore di Assegnamento

L'operatore di assegnamento (i.e. =):

- Serve a cambiare il contenuto della variabile
- Ha la sintassi:

```
<variabile> = <espressione>
```

Quando l'operatore viene eseguito:

- Valuta `<espressione>`
- Inserisce il valore denotato in `<variabile>`
- Denota il valore (i.e. lo restituisce)

Operatore di Assegnamento

Vediamo qualche esempio:

```
a = 10;
```

- Nella variabile "a" viene inserito il valore 10

```
a = 10;
a = -2;
```

- Dopo il primo assegnamento "a" contiene "10"
- Dopo il secondo assegnamento "a" contiene -2

R-value e L-value

Rispetto all'operatore di assegnamento

Una varabile può comparire a destra o a sinistra

```
a = a + 1;
```

- Quando compare a destra, è una espressione semplice
  - Denota (come già detto) il valore contenuto
  - Non a caso lo abbiamo chiamato R-value!
- Quando compare a sinistra, indica dove scrivere
  - Rappresenta l'indirizzo della cella di memoria
  - Non a caso lo abbiamo chiamato L-value!

R-value e L-value

Un operatore di assegnamento:

- Valuta sempre l'espressione alla sua destra
- Scrive il valore denotato nella variabile specifica a sinistra

Es. 1: L'operatore non rappresenta una uguaglianza matematica:

```
a = b
```

- Scrive il contenuto di "b" in "a"
- Non indica che "a" e "b" sono uguali

Es. 2: questa notazione non ha senso:

```
2 = b
```

- "b" è una espressione, ma "2" non è una variabile

Inizializzazione di una variabile

Si può assegnare un valore ad una variabile alla sua definizione

Si dice inizializzazione. Un esempio:

```
int a = 10;
```

Definisce "a" e vi inserisce un valore

Una variabile non inizializzata non è vuota:

- Ha il contenuto presente in memoria nella cella allocata
- Tale contenuto non è controllabile (lo si può pensare casuale)
- Per non avere sorprese, inizializzate le variabili

Assegnamenti come Espressioni

In C l'operatore di assegnamento è una espressione

- Denota il valore inserito nella variabile

Può comparire in espressioni:

```
3 * (a = 2)
```

- Denota 3 * 2 (e inserisce 2 in "a")

È associativo a destra:

```
a = b = 2
```

- Corrisponde a "a = (b = 2)", inserisce 2 in "b", quindi in "a"

Evitate di usarlo in questo modo (poco leggibile)

Espressioni con Effetti Collaterali

L'assegnamento è una espressione con effetti collaterali

- Quando viene valutata, oltre a denotare un valore
- ...Altera il contenuto di una variabile

In C vi sono altre espressioni del genere:

- Operatore di incremento ++
- Operatore di decremento --
- Operatori di assegnamento compatti

Incremento e Decremento

Gli operatori di incremento e decremento hanno due varianti:

Variante postfissa:

`i++` oppure `i--`

- Incrementa/decrementa "i"
- Denota il valore di "i" prima all'incremento

Variante prefissa:

`++i` oppure `--i`

- Incrementa/decrementa "i"
- Denota il valore di "i" dopo all'incremento

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

i vale 6, k vale 6

Incremento e Decremento: Esempi

```
int i, k = 5;
i = ++k;
```

i vale 6, k vale 6

```
int i, k = 5;
i = k++;
```

Incremento e Decremento: 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

Incremento e Decremento: 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

Incremento e Decremento: 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++;
```

Incremento e Decremento: 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++;
```

Incremento e Decremento: 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++;
```
Dipende dal compilatore (EVITARE!)

Operatori di Assegnamento Compatti

Una notazione compatta per operatori binari + assegnamento:

```
<variabile> <op>= <espressione>
```

Viene espansa in :

```
<variabile> = <variabile> <op> <espressione>
```

Esempi:

```
k += j     →    k = k + j
k *= (a+b)     →    k = k * (a + b)
```

Compatibilità di Tipo

E se il tipo dell'espressione e quello della variabile sono diversi?

Ci sono due casi possibili:

- Il tipo della variabile è più espressivo dell'espressione

```
float a = 5 * 2;
```

  - Viene effettuata una promozione di tipo
  - Nessun problema particolare

- Il tipo della variabile è meno espressivo dell'espressione

```
int a = 5 / 2.0;
```

  - Si perde informazione!
  - In questo caso, "a" conterrà 2 invece che 2.5 (troncamento)

Un Semplice Esempio

- Data una temperatura espressa in gradi Celsius
- ...calcolare il corrispondente valore in gradi Fahrenheit

Soluzione:

```
int main() {
    float c = 18;

    /* Vale l'uguaglianza: c * 9/5 = f – 32 */

    float f = 32 + c * 9/ 5.0;
}
```

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono	Neutrale Buono	Caotico Buono
Legale Neutrale	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono	Caotico Buono
Legale Neutrale	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono
Legale Neutrale	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale `i -= -1`
Legale Malvagio	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale `i -= -1`
Legale Malvagio `i = i + i/i`	Neutrale Malvagio	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale `i -= -1`
Legale Malvagio `i = i + i/i`	Neutrale Malvagio `i += 73*139%2`	Caotico Malvagio

Un Ultimo Esempio (per nerd ;-))

Incrementi: tabella degli allineamenti

Legale Buono `i = i + 1`	Neutrale Buono `i++`	Caotico Buono `++i`
Legale Neutrale `i = ((i)+(1))`	Neutrale Puro `i += 1`	Caotico Neutrale `i -= -1`
Legale Malvagio `i = i + i/i`	Neutrale Malvagio `i += 73*139%2`	Caotico Malvagio `i += ++i/i--`