Strutture di controllo, tipi di base e funzioni

Matteo Spanio

19 marzo 2024

Il C è basato su *espressioni* più che su istruzioni. Queste espressioni sono un insieme di **variabili**, **costanti** e **operatori**.

Operatori

Gli operatori in C sono fondamentali per eseguire operazioni su variabili e valori. Ogni operatore ha diverse caratteristiche che influenzano il modo in cui viene valutata un'espressione.

Tipi di Operatori

Gli operatori in C possono essere suddivisi in diverse categorie in base al tipo di operazioni che eseguono. Ogni operatore ha una priorità che determina l'ordine in cui viene valutato all'interno di un'espressione. Gli operatori con una priorità più alta vengono valutati prima degli operatori con una priorità più bassa. Nel caso di operatori della stessa priorità, l'associatività determina l'ordine di valutazione.

Ecco una tabella riassuntiva degli operatori C, ordinati da quelli con più alta priorità a quelli con minore priorità:

Categoria	Operatori	Associatività
Postfissi	`++ -- -> . () []`	Sinistra-Destra
Unari / Prefissi	`+ - ! ~ ++ -- (type) * & sizeof`	Destra-Sinistra
Moltiplicativi	`* / %`	Sinistra-Destra
Additivi	`+ -`	Sinistra-Destra
Shift	`<< >>`	Sinistra-Destra
Relazionali	`< > <= >=`	Sinistra-Destra
Uguaglianza	`== !=`	Sinistra-Destra
Bitwise AND	`&`	Sinistra-Destra
Bitwise XOR	`^`	Sinistra-Destra
Bitwise OR	`|`	Sinistra-Destra

Categoria	Operatori	Associatività
Logico AND	`&&`	Sinistra-Destra
Logico OR	`||`	Sinistra-Destra
Condizionale	`?:`	Destra-Sinistra
Assegnamento	`= += -= *= /= %= <<= >>= &= ^= |=`	Destra-Sinistra
Virgola	`,`	Sinistra-Destra

Side-Effects e Incubi

Gli operatori di incremento e decremento (`++` e `--`) sono potenti ma vanno usati con cautela. Possono essere utilizzati sia come operazioni prefisse che postfix. La differenza principale tra queste due forme sta nel momento in cui viene effettuata l'operazione di incremento o decremento.

Nel caso dell'operatore di post-incremento `i++`, l'incremento avviene dopo che il valore di `i` viene utilizzato nell'espressione corrente. Nel caso dell'operatore di pre-incremento `++i`, l'incremento avviene prima che il valore di `i` venga utilizzato nell'espressione.

Questo può portare a comportamenti diversi, soprattutto quando si utilizzano questi operatori all'interno della stessa espressione o dello stesso statement. Ad esempio:

```c
int i = 1;
printf("i is %d\n", i++);  // Stampa: i is 1
printf("i is %d\n", i);    // Stampa: i is 2
printf("i is %d\n", ++i);  // Stampa: i is 3
printf("i is %d\n", i);    // Stampa: i is 3
```

Nel primo `printf`, l'operatore post-incremento viene utilizzato, quindi il valore di `i` (1) viene stampato e poi incrementato a 2. Nel secondo `printf`, il valore incrementato di `i` (2) viene stampato. Nel terzo `printf`, l'operatore pre-incremento viene utilizzato, quindi `i` viene incrementato a 3 e poi stampato. Infine, nel quarto `printf`, viene stampato di nuovo il valore corrente di `i`, che è 3.

È importante prestare attenzione a questi comportamenti per evitare risultati inaspettati nel proprio codice.

Espressioni istruzione

Tutte le espressioni possono essere anche *statement*: in una linea posso avere anche solo una operazione singola seguita da `;`

```c
i = 1;      // utile
i++;        // utile
i * j + 2   // inutile, potrebbe dare warning
            // "statement with no effect"
```

Nella seconda riga dell'esempio il risultato viene scartato, ma la modifica avviene lo stesso.

Espressioni logiche

Le espressioni logiche, o booleane, in C sono fondamentali per l'esecuzione di operazioni condizionali e decisionali nel codice.

Operatori Booleani

Gli operatori booleani sono utilizzati per eseguire operazioni logiche su valori booleani o espressioni che possono essere valutate come vere o false. Ecco gli operatori booleani disponibili in C:

- `&&` (AND logico)
- `||` (OR logico)
- `!` (NOT logico)
- `==` (uguaglianza)
- `!=` (diverso)
- `>` (maggiore)
- `<` (minore)
- `>=` (maggiore o uguale)
- `<=` (minore o uguale)

Questi operatori producono tutti 0 o 1 come risultato, ma in C qualsiasi valore diverso da 0 è considerato vero.

AND Logico (`&&`)

L'operatore `&&` restituisce 1 se entrambe le espressioni booleane sono non-zero, altrimenti restituisce 0. È un operatore a corto circuito, quindi se il risultato è già noto dopo aver valutato la prima espressione, la seconda non viene valutata.

OR Logico (`||`)

L'operatore `||` restituisce 1 se almeno una delle due espressioni booleane è non-zero, altrimenti restituisce 0. Anche questo è un operatore a corto circuito.

NOT Logico (`!`)

L'operatore `!` inverte il valore di verità di un'espressione. Restituisce 1 se l'espressione è zero e 0 se l'espressione è non-zero.

Operatori di Confronto

Gli operatori di confronto (`==`, `!=`, `>`, `<`, `>=`, `<=`) confrontano due valori e restituiscono un valore intero in base alla relazione tra di essi.

Valutazione delle Espressioni Booleane

Le espressioni booleane vengono valutate in base alle regole della logica booleana. Il risultato di un'espressione booleana è un valore intero, dove 0 rappresenta "falso" e 1 rappresenta "vero". Ad esempio:

```
int i = 3, j = 2, k = 1;
int result = (i < j && j < k); // result sarà 0 perché entrambe le espressioni sono false
```

In questo caso, l'espressione `i < j && j < k` sarà valutata come `0 && 0`, il che restituirà 0 perché entrambe le espressioni sono false.