Espressioni e operatori

Espressioni

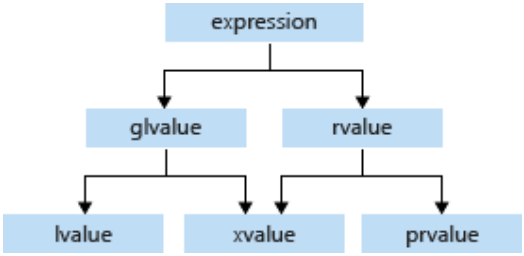
Definiamo come espressione una sequenza di *operatori* e di *operandi* che definiscono una computazione, cioè un'operazione. Lo svolgimento di un'espressione può generare un risultato (ad esempio `2 + 2 => 4`) e/o un effetto-laterale, come esempio possiamo prendere `cout << 3` che stampa in output, sulla console, il numero 3.

Tipi di espressioni (facoltativo)

Ogni espressione in C++ ha un tipo ed appartiene ad una categoria di valori. Le categorie di valori sono la base per le regole che i compilatori devono seguire durante la creazione, la copia e lo spostamento degli oggetti(temporanei) durante la valutazione delle espressioni. Alcuni esempi di oggetti temporanei sono le variabili, gli array, le funzioni e gli operatori.

Questo schema rappresenta i possibili tipi che può assumere un'espressione e sono importanti per avere una conoscenza approfondita del funzionamento del linguaggio. Questa sezione è prettamente conoscitiva (**non viene presa in oggetto per la valutazione**), se volete approfondire autonomamente l'argomento vi lascio un video efficace nella spiegazione:

https://www.youtube.com/watch?v=fbYknr-HPYE



Categorie di espressioni

Possiamo dividere le espressioni in varie categorie, queste categorie come vedremo alla fine hanno una gerarchia di esecuzione ben precisa che il compilatore segue per comprendere correttamente il codice.

Espressioni primarie

Sono i blocchi di espressioni più semplici rappresentati da valori letterali (cioè i literal che abbiamo visto nella sezione delle variabili, ad esempio `'a'`, `10`, `4.5`), i nomi globali e le espressioni dichiarate dentro le parentesi tonde, per esempio `(i + 1)`. Ne sono presenti altri ma non verranno trattati nelle lezioni.

Esempi di espressioni primarie

```
'a',                    //literal
variabile,              //nome
( i  + 1 )              //espressione fra parentesi
```

Espressioni di suffisso

Generalmente sono espressioni primarie (o anche operatori singoli) seguite da degli operatori specifici, successivamente vengono descritti alcuni di questi operatori.

expr++	Operatore incremento suffisso	Incrementa l'operando di 1
expr--	Operatore decremento suffisso	Decrementa l'operando di 1
expr[ ]	Operatore sottoscrizione (Array)	Permette la dichiarazione degli Array
expr( )	Operatore chiamata funzione (Funzioni)	Permette l'utilizzo delle funzioni

Espressioni con operatori unari

Queste espressioni contengono degli operatori che agiscono solo su un operando all'interno di un'espressione.

++*expr*	Operatore incremento prefisso	Incrementa l'operando di 1
--*expr*	Operatore decremento prefisso	Decrementa l'operando di 1
!*expr*	Operatore di negazione logica	Inverte il significato dell'operando dopo averlo convertito in `bool`
+*expr*	Operatore + (di valore)	Emette il valore dell'operando in `int`
-*expr*	Operatore di negazione	Emette il valore negativo dell'operando in `int`
( )*expr*	Operatore di cast	Permette di convertire il tipo di un oggetto
sizeof*expr*	Operatore di dimensione	Emette (in byte) la dimensione dell'espressione
~*expr*	Operatore di complemento a uno	Esegue il complemento ad 1 dell'espressione
&*expr*	Indirizzo di memoria di expr	
**expr*	Riferimento indiretto a expr	

Espressioni con operatori binari

Gli operatori binari agiscono su due operandi in un'espressione. Gli operatori che interessano a noi sono:

Operatori moltiplicativi

expr1 * *expr2*	Operatore di moltiplicazione	
expr1 / *expr2*	Operatore di divisione	
expr1 % *expr2*	Operatore di modulo	

Operatori additivi

expr1 + *expr2*	Operatore di somma	
expr1 - *expr2*	Operatore di differenza	

Operatori di spostamento

expr1 >> *expr2*	Operatore di spostamento a destra	
expr1 << *expr2*	Operatore di spostamento a sinistra	

Operatori relazionali e di uguaglianza

expr1 < *expr2*	Minore di	
expr1 > *expr2*	Maggiore di	
expr1 <= *expr2*	Minore o uguale a	
expr1 >= *expr2*	Maggiore o uguale a	
expr1 == *expr2*	Uguale a	
expr1 != *expr2*	Diverso da	

Operatori bit per bit

expr1 & *expr2*	AND bit a bit	
expr1 | *expr2*	OR bit a bit (inclusivo)	
expr1 ^ *expr2*	OR bit a bit (esclusivo)	

Operatori logici

expr1 && *expr2*	AND logico	
expr1 || *expr2*	OR logico	

Operatori di assegnazione

expr1 = *expr2*	Assegnazione	
expr1 += *expr2*	Assegnazione di addizione	*expr1* = *expr1* + *expr2*
expr1 -= *expr2*	Assegnazione di sottrazione	*expr1* = *expr1* - *expr2*
expr1 *= *expr2*	Assegnazione di moltiplicazione	*expr1* = *expr1* * *expr2*
expr1 /= *expr2*	Assegnazione di divisione	*expr1* = *expr1* / *expr2*
expr1 %= *expr2*	Assegnazione di modulo	*expr1* = *expr1* % *expr2*
expr1 <<= *expr2*	Assegnazione di spostamento a sinistra	*expr1* = *expr1* << *expr2*
expr1 >>= *expr2*	Assegnazione di spostamento a destra	*expr1* = *expr1* >> *expr2*
expr1 &= *expr2*	Assegnazione di AND bit a bit	*expr1* = *expr1* & *expr2*
expr1 |= *expr2*	Assegnazione di OR bit a bit (inclusivo)	*expr1* = *expr1* | *expr2*
expr1 ^= *expr2*	Assegnazione di OR bit a bit (esclusivo)	*expr1* = *expr1* ^ *expr2*

Operatore virgola ,

Dove previsto, ad esempio nell'utilizzo delle funzioni, permette di raggruppare due espressioni

expression, expression

Operatore condizionale

Questo operatore permette di sostituire la condizione if in modo elegante e su una sola riga.

expr1 ? *expr2* : *expr3*

Ordine di esecuzione delle espressioni

Quando dichiarate un'espressione questa viene compresa dal calcolatore in un ordine ben preciso, questo ordine viene diviso in primo luogo come associatività degli operatori, cioè il modo in cui il calcolatore legge i singoli operatori, e successivamente con la precedenza degli operatori gli uni rispetto agli altri

Associatività prioritaria a destra

Significa che l'espressione viene letta dal compilatore da destra verso sinistra, in questa sezione sono presenti gli **operatori unari** e gli **operatori binari di assegnamento**.

Alcuni esempi di questo tipo di associatività:

a = b = c -> a = (b = c)

a += b -> a = (a + b)

Associatività prioritaria a sinistra

Al contrario in questo tipo di associatività le espressioni vengono lette da sinistra verso destra, tutti gli operatori tranne quelli dichiarati prima fanno parte di questa categoria (compresi gli **operatori di suffisso**).

a + b + c -> (a + b) + c

a << b << c -> (a << b) << c

Precedenza di esecuzione degli operatori

Gli operatori vengono letti ed eseguiti dal compilatore in un ordine preciso descritto dalla tabella sottostante, seguendo queste regole di precedenza avrete, quasi sempre, la certezza di come vengono eseguite le vostre espressioni.

N.B. L'ordine con cui ho scritto gli operatori non è casuale.

1	Operatori di suffisso	e[ ] e( ) e++ e--
2	Operatori unari	sizeof *e* ++e --e ~e !e -e +e &e *e ()e
3	Operatori binari di moltiplicazione	e1*e2 e1/e2 e1%e2
4	Operatori binari di addizione	e1+e2 e1-e2
5	Operatori binari di spostamento	e1<<e2 e1>>e2
6	Operatori binari di confronto (1)	e1<e2 e1>e2 -> e1<=e2 -> e1>=e2
7	Operatori binari di confronto (2)	e1==e2 e1!=e2
8	Operatore AND bit a bit	e1&e2
9	Operatore OR bit a bit (esclusivo)	e1^e2
10	Operatore OR bit a bit (inclusivo)	e1|e2
11	Operatore AND logico	e1&&e2
12	Operatore OR logico	e1||e2
13	Operatori di assegnazione	e1=e2 -e1*=e2 e1/=e2 -e1%=e2 e1+=e2 e1-=e2 e1<<=e2 e1>>=e2 e1&=e2 e1|=e2 e1^=e2 e1 ? e2 : e3
14	Operatore virgola	e1, e2

Esercizi per la comprensione

Date le seguenti variabili:

```
int a;

bool b;

char c;
```

Calcolare il risultato delle espressioni e verificare poi il risultato in un programma.

Link al compilatore online

Operatori di suffisso

1. a = 0;
   a++;
   cout << a; //?
2. a = 5;
   a--;
   cout << a; //?
3. c = 'd'
   c++;
   cout << c; //?

Operatori unari

1.
```
a = 0;
++a;
cout << a;          //?
```

2.
```
a = 5;
--a;
cout << a;          //?
```

3.
```
a = 26;
cout << -a;         //?
```

4.
```
a = 12;
cout << +a;         //?
```

5.
```
b = 0;
b = !b;
cout << b;          //?
```

6.
```
a = 97;
cout << a;
cout << (char)a;    //?
```

7.
```
a = 97;
b = 1;
c = 'a';
//Numero intero
cout << sizeof(int);        //?
cout << sizeof(a);          //?

//Booleano
cout << sizeof(bool);       //?
cout << sizeof(b);          //?

//Carattere
cout << sizeof(char);       //?
cout << sizeof(c);          //?
```

Esercizi che comprendono anche le espressioni precedenti

1.
```
int d = 0;
a = 0;
cout << a++;        //?
cout << ++d;        //?
cout << d;          //?
```

Espressioni

1. Valutare l'ordine risoluzione della seguente espressione e verificare il risultato in un file espressioni.cpp
   23-56/8*6+34%2

2. Valutare l'ordine di risoluzione della seguente espressione e verificare il risultato in un file espressioni.cpp
   45+3*9-57%13/++a
   Con a variabile intera di valore 5

Programmi

1. Scambiare due variabili senza utilizzarne una terza.