Istruzioni semplici in C

Espressioni

In C è fondamentale il concetto di espressione. Ogni espressione è composta da uno o più operandi e da uno o più operatori, e possiede:

- un valore: dato dal risultato della valutazione dell'espressione sulla base dei valori delle costanti e delle variabili al momento della valutazione;
- un tipo: che dipende dalle variabili, dalle costanti e dagli operatori usati nell'espressione.

Attenzione!!! Il valore è attribuito durante l'esecuzione mentre il tipo è determinato durante la compilazione.

```
1 /* tipo int valore 1 */
2 * 3 /* tipo int valore 6 */
4.9 * 2 /* tipo double valore 9.8 */
```

Espressione di chiamata di funzione

- É un espressione del tipo `f(H)` dove:
  - `f` è il nome di una funzione,
  - `H` è la lista, eventualmente vuota, degli argomenti richiesti dalla funzione.
- Esempi:

```
printf("Questa è una chimamata di funzione")
sqrt(4)
abs(-3)
```

- Il valore dell'espressione è il risultato dell'esecuzione della funzione.
- Il tipo dell'espressione è dichiarato nella definizione della funzione.

Espressioni aritmetiche

- Informalmente in C un'espressione aritmetica viene costruita utilizzando valori costanti, variabili, chiamate di funzioni e gli usuali operatori aritmetici (`+ - * / %`).

- Esempio:
  a + b
  i / 3
  ((a + b) / 5 * 2.0)

- Il valore dell'espressione è il risultato dell'espressione aritmetica.
- Il tipo dell'espressione dipende dagli operandi. In genere se un operando è di dipo `double`, l'espressione è di tipo `double`, altrimenti è di tipo `int`.

Espressione relazionali

- Sono espressioni costruite utilizzando gli operatori relazionali (`==`, `!=`, `>`, `>=`, `<`, `<=`) e logici (`!`, `&&`, `||`).
- Esempio:

```
x == 0
a + b > a - b
(x > y) || (x == y)
((x > y) && (x == y))
```

- I valori restituiti sono:
  - 1 se l'espressione è vera
  - 0 se l'espressione è falsa.
- Il tipo dell'espressione è `int`.

Espressione di assegnamento: definizione

- É un'espressione che usa l'operatore "=". La sua forma generale è:

  `variabile=espressione`

  dove:

  - `variabile` (lvalue) è un identificatore di variabile precedentemente definita,
  - `espressione` (rvalue) è una espressione C.
- Attenzione! L'espressione di assegnamento `variabile=espressione` non indica un'uguaglianza.

```
x = 0
x = 3 * 2 + x
x = sqrt(10) * 2.0
```

Espressione di assegnamento: valore e tipo

- La valutazione di `variabile=espressione` avviene nel seguente modo:
  1. Viene valutata `espressione`.
  2. Il valore prodotto dal punto precedente viene assegnato in `variabile` (eventualmente dopo una conversione di tipo).
- Il tipo di `variabile=espressione` è il tipo di `variabile`.
- Il valore di `variabile=espressione` è il valore di `variabile` dopo la valutazione.

Esempi

Se x e y sono rispettivamente variabili di tipo `int` e `double`

1. x = 0 /* 1 */
2. y = x + 7 /* 2 */
3. x = x + 1 /* 3 */

Osservazione

- É possibile fare assegnamenti multipli. Ad esempio:

```
x = y = 5
```

  consente di assegnare il valore 5 sia alla variabile x sia alla variabile y.

- L'assegnamento alle due variabili non viene comunque fatto simultaneamente. L'ordine di valutazione è da destra verso sinistra, cioè l'espressione precedente risulta parentesizzata nel seguente modo:

```
x = (y = 5)
```

Esercizi:

```
x = 3 + (y = 7)
x = (z = 2) * (y = 5)
x = (y = 4) + (y = 3)
```

Istruzioni semplici

- Una qualsiasi espressione, `x=0`, `a+b`, `i=i+1` oppure `printf(...)` diventa un'**istruzione** quando è seguita da un punto e virgola (`;`).
- Esempio:

```
x=0;
a+b;
i=i+1;
((a+b) / 5 * 2.0);
printf(...);
;                          /* istruzione vuota */
```

- In C '`;`' è un terminatore di istruzioni e non un separatore come in altri linguaggi di programmazione (Ad esempio: Pascal).

Visualizzazione: `printf`

- La funzione `printf()` (**formatted print**) permette di visualizzare una **stringa di controllo** contenuta fra doppi apici.
  Ad esempio: printf("questo e' un messaggio"); visualizza `questo e' un messsaggio`.
- La stringa di controllo può contenere delle **specifiche di conversione** che iniziano con il carattere % e ciascuna viene associata a un parametro.
- I parametri sono scritti dopo la stringa di controllo e sono separati dalla virgola.
- La specifica di conversione determina il formato secondo cui il corrisponente parametro viene stampato.

Uso di `printf`

Carattere di conversione	Formato
d	intero in notazione decimale
f	reale in notazione decimale

Esempio:

```
printf("%d",123);
printf("%f", 108.23);
```

visualizzano `123` e `108.23`.

Lettura da tastiera: `scanf`

- La funzione `scanf()` permette di leggere dallo standard input e di interpretare i caratteri letti in base al formato richiesto.

Carattere di conversione	**Formato**
`d`	intero in notazione decimale
`lf`	numero reale

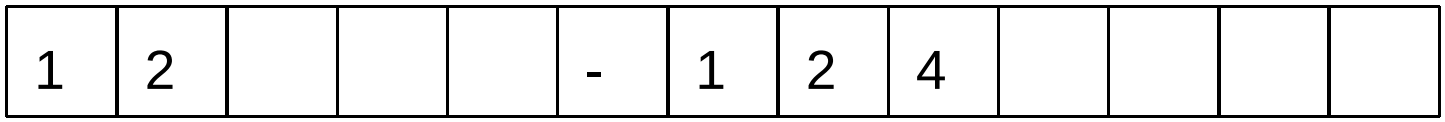
La funzione `scanf`

- Per leggere un dato di input in una variabile x, occorre passare come parametro l'**indirizzo** di x usando l'operatore di indirizzamento &.
  Ad esempio, leggere da standard input un intero e memorizzarlo nella variabile x occorre eseguire l'espressione scanf("%d", &x);
  Se si scrive `scanf(''%d'', x)` il programma viene compilato senza segnalare errori ma non funziona correttamente
- Utilizzando le specifice di converione `d` e `lf`, vengono **automaticamente** saltati i caratteri di spaziatura (spazio, tabulazioni, nuova linea) che si trovano prima del numero da leggere.

Esempi

```
int a,b;
scanf("%d%d", &a, &b);
```

Lo standard input si può rappresentare come una sequenza di caratteri, ad esempio



dove le caselle vuote contengono un carattere di spaziatura.
L'effetto dell'istruzione di lettura è:

- Viene convertita la sequenza di caratteri '1' e '2' nell'intero 12 e viene assegnato ad a il valore 12.
- Vengono saltati i caratteri di spaziatura fra '12' e '-124', viene convertito '-124' nel numero -124 e assegnato tale valore a b.

Esempi (2)

Supponiamo che questa volta l'input sia

		1	2		a		1	2	8

allora:

- Vengono saltati gli spazi e viene letto '12', quindi alla variabile a viene assegnato il valore 12.
- A questo punto la `scanf( )` si aspetta di trovare (saltando gli spazi) una sequenza di caratteri corrispondente alla rappresentazione decimale di un intero. PoichÈ invece viene incontrato il carattere 'a', Il valore della variabile b non cambia e la prossima richiesta di lettura esaminerà lo standard input a partire dal carattere 'a'.

Esempio

```
#include <stdio.h>
int main(void)
{
    int a, b;
    scanf("%d", &a);
    b = a + 1;
    printf("a=b: %d\ n",a==b);
    printf("a!=b: %d\ n",a!=b);
    printf("a<b: %d\ n",a<b);
    printf("a<=8: %d\ n", a<=8);
    return 0; }
```

Esecuzione

Supponiamo di dare in ingresso il valore 8.