

Unità Didattica 2

Linguaggio C

Espressioni, Operatori e
Strutture linguistiche per il
controllo del flusso

Espressioni e assegnazioni

- Le espressioni sono definite dalla grammatica:
espressione = variabile | costante | operatore (lista_espressioni)
- Gli operatori aritmetici binari si scrivono in notazione infissa, utilizzando parentesi, secondo le usuali convenzioni di precedenza. Ad esempio: $x + 25 * (y - 1) + x \% y$
- Le assegnazioni hanno la forma:
variabile = espressione;
- Un' assegnazione in C è un'espressione, il cui valore è il valore assegnato alla variabile a sinistra del simbolo '='
- Osservazioni:
 - E' possibile fare assegnazioni all'interno di espressioni, es:
if ((n = funz(s)) > 10)
 - E' possibile fare assegnazioni multiple simultanee, es:
x = y = z = 0;

Operatori aritmetici

- Oltre agli operatori aritmetici standard +, -, *, / e all'operatore % (modulo) per gli interi, in C si hanno anche gli operatori incremento ++ e decremento --, che possono essere preposti o posposti all'argomento.
- Se sono preposti il valore e' calcolato prima che l'espressione sia valutata, mentre se sono posposti il valore viene calcolato dopo la valutazione della espressione. Esempi:

```
int z = 2;
```

```
x = z++;    /* A questo punto x=2 e z=3 */
```

```
x = ++z;    /* A questo punto x=3 e z=3 */
```

Operatori aritmetici

- L'operatore "%" (modulo) può essere utilizzato solamente con le variabili di tipo int;
- La divisione "/" e' utilizzata sia per gli `int` che per i `float`.
- Esiste una forma contratta per espressioni del tipo

`expr1 = expr1 [op] expr2` (es: `i=i+2` o `x=x*(y+3)`)
diventano: `expr1 [op]= expr2`

- In particolare:

<code>x+=y;</code>	equivale a	<code>x=x+y;</code>
<code>x-=y;</code>	equivale a	<code>x=x-y;</code>
<code>x*=y;</code>	equivale a	<code>x=x*y;</code>
<code>x/=y;</code>	equivale a	<code>x=x/y;</code>

- **Osservazione:** l'espressione `x*=y+3` corrisponde a:

`x=x*(y+3)` e non a `x=x*y+3`.

Operatori di confronto

- $x==y$ Verifica se il valore di x è uguale a y
- $x>y$ Verifica se x è maggiore di y
- $x>=y$ Verifica se x è maggiore uguale di y
- $x<y$ Verifica se x è minore di y
- $x<=y$ Verifica se x è minore uguale di y
- **Osservazione:**
 - **if (i==j)** ... esegue il contenuto dell'if se i è uguale a j ;
 - ma **if (i=j)** ... è ancora sintatticamente esatto ma effettua l'assegnazione del valore di j ad i e procede se j è diverso da zero in quanto viene interpretato il valore TRUE ($\neq 0$) (è come scrivere **if (i)**... con $i \neq 0$). In questo caso si tratta di una "assegnazione di valore".

Operatori logici e relazionali ed espressioni booleane

- **&&** \Leftrightarrow **AND logico;**
- **||** \Leftrightarrow **OR logico;**
- **!** \Leftrightarrow **NOT logico.**
- Il C non prevede il tipo “booleano”. Al loro posto si usano gli interi:
 - 0 (false);
 - qualunque intero \neq 0 (true);
- Generalmente è preferibile definire le costanti:
 - `#define TRUE 1` oppure: `const int true = 1;`
 - `#define FALSE 0` oppure: `const int false = 0;`
- Le espressioni booleane si formano utilizzando:
 - Simboli relazionali: `>`, `>=`, `<`, `<=`, `==`, `!=`;
 - Operatori logici: `&&` (and), `||` (or), `!` (not)
- Esempio: `((x > 3) && ((y == 1) || !(y > 0)))`

Regole di precedenza degli operatori

- Tutti gli operatori hanno una propria priorità, e gli operatori ad alta priorità sono valutati prima di quelli a bassa priorità.
- Gli operatori con la stessa priorità sono valutati da sinistra a destra. Ad esempio: **a - b - c** è valutato **(a - b) - c**
- In generale gli operatori unari hanno priorità su quelli binari, e gli operatori relazionali su quelli logici;
- Le priorità possono essere variate mediante l'uso di parentesi;
- Esempio: `a < 10 && 2 * b < c;`
e' interpretato come:
`(a < 10) && ((2 * b) < c);`

Conversioni implicite del tipo di una variabile

- Il compilatore C sa solo valutare espressioni tra argomenti dello stesso tipo;
- Promozione (o conversione implicita) di tutti i dati al tipo 'più alto' contenuto nell'espressione (ad esempio genererà una copia degli operandi interi promuovendoli a `float`, nel caso in cui sia presente nell'espressione un operando `float`).

Esempio:

```
float media, totale;  
int contatore;  
media = totale / contatore;
```

- Il compilatore genera una copia di "contatore" implicitamente convertita al tipo `float`;
- Si ottiene un risultato corretto di divisione fra due `float`:

```
Se totale == 3.0, contatore == 2  media= 3.0/2.0 = 1.5
```

Conversioni esplicite del tipo di una variabile

- Il C fornisce una modalità di conversione esplicita del tipo di una variabile mediante l'operazione di cast:

- `(tiponuevo)variabile;`

- **Esempio:**

```
float media;
```

```
int totale, contatore;
```

```
media = (float)totale / (float)contatore;
```

- L'operatore unario `(float)` ha la funzione di creare una copia temporanea in virgola mobile di **totale** e **contatore**:

- Se `totale == 3, contatore == 2` `media = 3.0/2.0 = 1.5`

- Senza l'utilizzo dell'operazione di casting avremmo avuto:

- Se `totale == 3, contatore == 2` `media = 3/2 = 1.0` (**Errore!**)

Costrutto di selezione *if*

- Usato per scegliere tra percorsi di azione alternativi;
- Esempio in pseudocodice:

*Se il voto dello studente è maggiore o uguale a 18
visualizza “promosso”;*

- In C:

```
if ( voto >= 18 )  
    printf("Promosso\n");
```

Costrutto di selezione *if/else*

- **if:** esegue un'azione solo se la condizione è vera
- **if/else:** Specifica ed esegue l'azione associata ad `if` nel caso in cui la condizione sia vera, quella associata ad `else` nel caso in cui sia falsa

- **Pseudocodice:**

Se il voto dello studente è maggiore o uguale a 18

visualizza “promosso”

altrimenti

visualizza “bocciato”

- **In C:**

```
if ( voto >= 18 )
    printf("promosso\n");
else
    printf("bocciato\n");
```

Costrutto di selezione *if/else*

- In C esiste anche l'operatore condizionale di selezione ternario (?:)

```
condizione? (valore se vero):(valore se falso);
```

- Esempio:

```
printf("%s\n", voto >= 18 ? "promosso" : "bocciato");
```

Esempio:

```
voto>=18 ? printf("promosso\n"):printf("bocciato\n");
```

Costrutto di ripetizione *while*

- Specifica un'azione da ripetere fino a quando una condizione rimane vera.

- **Formato:**

```
while (condizione)  
{  
    ...  
}
```

- **Esempio:**

```
int valore = 2;  
while (valore <= 1000)  
{  
    ...  
    valore = 2 * valore;  
}
```

Costrutto di ripetizione *do/while*

- Simile al costrutto `while`;
- La condizione di ripetizione viene valutata solo dopo l'esecuzione del blocco di azioni associato;
- Il blocco di azioni viene eseguito *almeno* una volta;

- Formato:

```
do
    { ...
    }while (condizione);
```

- Esempio:

```
int counter = 1;
do
{
    printf("%d ", counter);
}while(++counter <= 10);
```

Costrutto di ripetizione *for*

- Gestisce direttamente nella sintassi del ciclo una fase di inizializzazione e di aggiornamento di variabili;

- **Formato:**

```
for(inizializzazione; test_fine_ciclo; aggiornamento)  
{ ... }
```

- **Esempio:** `int i;`

```
    for (i = 0; i < 10; i++)  
        printf("%d\n", i);
```

- Il `for` è equivalente ad un ciclo gestito tramite `while`:

```
inizializzazione  
while (test_fine_ciclo) {  
    ...  
    aggiornamento;  
}
```

Costrutto di ripetizione *for*

- Note sui comandi di inizializzazione e aggiornamento:

- Possono essere liste di comandi separati da virgola

Esempio:

```
for (int i = 0, j = 0; j+i <= 10; j++, i++)  
    printf("%d\n", j+i);
```

- Inizializzazione, test e aggiornamento possono contenere espressioni aritmetiche.

Esempio:

```
for (j = x; j <= 4*x*y; j += y/x)
```

- Se il test di terminazione è inizialmente falso:
 - Il blocco del costrutto `for` non viene eseguito;
 - Il controllo passa all'istruzione successiva al blocco del `for`.

Comandi break e continue

- Break
 - causa l'uscita immediata da un while, for, do/while o switch
 - l'esecuzione continua con la prima istruzione seguente al blocco che contiene il comando break;
- Continue
 - Salta le istruzioni restanti all'interno del blocco di un while, for, do/while;
 - Procede alla successiva iterazione del ciclo:
 - while e do/while: il test di continuazione viene valutato immediatamente dopo l'esecuzione di continue.
 - for: viene valutata l'espressione di aggiornamento, quindi viene valutato il test di continuazione.

Esempi di uso di Break e Continue

```
for (i = 0; i < 100; i++)
{
    if (condizione(i))
        break;           //se condizione(i) e' vera si esce
                        //forzatamente dal ciclo for

    istruzioni;
}
```

```
-----
for (i = 0; i < 100; i++)
{
    if (condizione(i))
        continue;       //se condizione(i) e' vera si passa di
                        //nuovo il comando all'istruzione for
                        //per una nuova iterazione

    istruzioni;
}
```

Costrutto di selezione *switch*

- Controlla se una variabile assume un valore all'interno di un certo insieme di costanti intere e di espressioni costanti;

- **Formato:**

```
switch (var){
    case '1':
        ...;
        break;
    case '2':
        ...;
        break;
    default:
        ...;
        break;
}
```

- Se `var` (`int` o `char`) assume uno dei valori contemplati, il programma esegue le azioni associate a quel caso (`case`) particolare (`default` è il caso eseguito quando nessuna delle condizioni è verificata);
- I `break` garantiscono l'esclusività dei casi, altrimenti si eseguirebbero tutte le azioni a partire da quella corrispondente al caso verificato;