

Laboratorio di Informatica Generale I UD

Terza esercitazione

Danilo Severina

9 marzo 2006

Espressioni

Definizione formale:

una espressione è una serie di simboli usati per produrre un valore.

L'*espressione* è uno degli elementi base del C ed analogamente ad una espressione matematica è formata da operandi ed operatori. In fase di compilazione tutte le espressioni sono sempre risolte in un valore dal compilatore.

Espressioni aritmetiche

L'espressione aritmetica è un insieme di variabili, costanti e funzioni interconnesse da operatori aritmetici. Il risultato di un'espressione numerica è sempre un valore numerico.

Gli operatori numerici hanno una loro gerarchia di priorità (in ordine decrescente di priorità):

- negazione -
- moltiplicazione *, divisione /, modulo %
- somma +, sottrazione -
- assegnamento =

Inoltre si possono utilizzare le parentesi per alterare la priorità degli operatori aritmetici:

$$y = 3$$

$$2 * y - 4 = 2$$

$$2 * (y - 4) = -2$$

Esempi di espressioni aritmetiche:

$a = 15$

assegnamento alla variabile a il valore 15;

$a + b$

somma di due variabili;

$x = a + b + c - d$

somma di variabili con assegnamento.

L'assegnamento avviene da destra verso sinistra:

$a = b + c$ significa $a \leftarrow b + c$

Espressioni logiche

Le espressioni logiche generano come risultato un valore **VERO** o **FALSO** (0).

NB!!: qualsiasi valore diverso da 0 (ZERO) è considerato come valore **VERO!!**

Le espressioni di controllo non inizializzano una variabile ad un valore, ma valutano una condizione.

Gli operatori che si utilizzano sono i seguenti in ordine di priorità decrescente:

- maggiore $>$, maggiore o uguale $>=$, minore $<$, minore o uguale $<=$
- uguale a $==$, diverso $!=$

NB!!: Attenzione a non confondere l'operatore di assegnamento $=$ con l'operatore di confronto $==$.

Esempi con $a = 3$, $b = 5$:

$a == b$

a è UGUALE a b ? FALSE

$a < b$

a è MINORE a b ? TRUE

$a >= b$

a è MAGGIORE O UGUALE a b ? FALSE

$a != b$

a è DIVERSO da b ? TRUE

Operatori logici

Gli operatori logici servono a concatenare più di una espressione logica ed in ordine di priorità sono:

- NOT logico $!$
- AND logico $\&\&$

- OR logico `||`

Esempi $a = 3, b = 5$:

`a&&b`

`a AND b` è una condizione verificata? `TRUE`

`a||b`

`a OR b` è una condizione verificata? `TRUE`

`!a`

la negazione di `a` è una condizione verificata? `FALSE` (è come dire `!a == TRUE`)

Costrutti condizionali e cicli

if

Sintassi:

```
if (espressione1)
    istruzione1;
else
    istruzione2;
```

Funzionalità:

Se *espressione1* è verificata (fornisce un risultato logico `TRUE`) allora si esegue *istruzione1*, altrimenti si esegue *istruzione2*.

NB!!: *istruzione1* o *istruzione2* possono essere delle singole espressioni che richiedono una singola riga di codice o possono essere dei **blocchi di istruzioni**. I blocchi di istruzioni sono SEMPRE contenuti all'interno di parentesi graffe.

Esempi:

```
/* determinare se due variabili a e b contengono
lo stesso valore */
int main()
{
    int a=2;
    int b=3;
    if (a==b)
        printf("\nSono uguali!!\n");
    else
        printf("\nSono diversi!!\n");
    return 0;
}
```

```

/* calcolare differenza tra due numeri:
maggiore - minore e stampare il risultato*/
int main()
{

    int a=2;
    int b=3;
    int risultato;
    if (a>b)
    {
        risultato=a-b;
        printf("\n%d - %d = %d\n", a, b, risultato);
    }
    else
    {
        risultato=b-a;
        printf("\n%d - %d = %d\n", b, a, risultato);
    }
    return 0;
}

```

while

Sintassi:

```

while(espressione1)
{
    istruzione2;
}

```

Funzionalità:

Fintatno che *espressione1* è vera (TRUE) esegui *istruzione2*.

Esempio:

```

/* calcolare la somma dei primi 10 numeri
interi positivi e stampare il risultato */
int main()
{

    int i=0;
    int max=10;
    int risultato=0;
    while(i<=max)

```

```

    {
        risultato=risultato+i;    // risultato+=i;
        i=i+1;                    // i++;
    }
    printf("\nRisultato = %d\n", risultato);

    return 0;
}

```

do

Sintassi:

```

do
{
    istruzione2;
}while(espressione1);

```

Funzionalità:

Fintanto che *espressione1* è vera (TRUE) esegui *istruzione2*.

Esempio:

```

/* calcolare la somma dei primi 10 numeri
interi positivi e stampare il risultato*/

int main()
{
    int i=0;
    int max=10;
    int risultato=0;
    do
    {
        risultato=risultato+i;    // risultato+=i;
        i=i+1;                    // i++;
    }while(i<=max);

    printf("\nRisultato = %d\n", risultato);

    return 0;
}

```

NB!!: differenza tra ciclo `while` e ciclo `do...while`. Nel primo caso, PRIMA si fa il controllo e POI si esegue l'operazione, nel secondo caso PRIMA si esegue l'operazione e POI si fa il controllo. Nel secondo caso *operazione2* viene eseguita SEMPRE almeno

una volta.

Esempio:

```
/* Stampa dei primi n numeri interi positivi
con n inserito da utente */

int main()
{

    int i=0;
    int n;

    printf("\nInserisci un numero intero positivo: ");
    scanf("%d", &n);
    do
    {
        printf("\n%d", i);
        i++;
    }while(i<n);

    return 0;
}
```

for

Sintassi:

```
for ( inizializzazione ; condizione ; espressione )
{
    istruzione;
}
```

Funzionalità:

Il ciclo for esegue come prima cosa l'*inizializzazione* di una o piu' variabili e successivamente esegue il contenuto del blocco *istruzione* fino a che la *condizione* è verificata. Al termine di ogni iterazione si esegue *espressione*.

Il ciclo for e' analogo ad un ciclo while con il seguente costrutto:

```
inizializzazione;
while(condizione)
{
    istruzione;
```

```
    espressione;  
}
```

Esempi:

```
/* calcolare la somma dei primi 10 numeri  
interi positivi e stampare il risultato*/
```

```
int main()  
{  
    int i=0;  
    int max=10;  
    int risultato=0;  
    for(i=0 ; i<=max ; i++)  
    {  
        risultato=risultato+i;  
    }  
    printf("\nRisultato = %d\n", risultato);  
  
    return 0;  
}
```

Esercizi

Esercizio 1

Scrivere un programma che dato in ingresso un numero intero, determini se questo è primo o meno.

```
int main()
{
    int i=2;
    int n;
    int primo=1;

    scanf("%d", &n);
    while(i<n && primo==1)
    {
        if(n%i==0)
            primo=0;
        i++;
    }
    if(primo==1) /* if (primo) */
        printf("\n%d e' primo.\n", n);
    else
        printf("\n%d NON e' primo.\n", n);

    return 0;
}
```

```
int main()
{
    int i=2;
    int n;

    scanf("%d", &n);
    while(i<n && n%i!=0)
    {
        i++;
    }
    if(i==n)
        printf("\n%d e' primo.\n", n);
    else
        printf("\n%d NON e' primo.\n", n);

    return 0;
}
```

Esercizio 2

Scrivere un programma in grado di calcolare

$$\sum_{i=linf}^{lsup} i$$

con *linf* e *lsup* inseriti dall'utente.

Esercizio 3

Scrivere un programma in grado di calcolare

$$\sum_{i=linf}^{lsup} i^3$$

con *linf* ed *lsup* inseriti dall'utente.

Esercizio 4

Scrivere un programma in grado di calcolare

$$\sum_{i=linf}^{lsup} n^i$$

con *linf*, *lsup* ed *n* inseriti dall'utente.

```
/* sommatori_i=linf^lsup i^n
linf lsup ed n inseriti da utente
*/
int main()
{
    int linf, lsup, tot, i, j, n, add;

    printf("\nLimite inferiore: ");
    scanf("%d", &linf);
    printf("\nLimite superiore: ");
    scanf("%d", &lsup);
    printf("\nCoefficiente: ");
    scanf("%d", &n);

    for( i=linf, tot=0 ; i<=lsup ; i++)
    {
        printf("\n\tindex=%d", i);
        for( j=1, add=1 ; j<=i ; j++)
        {
            add = add * n;
            printf("\n add= %d", add);
        }
    }
}
```

```
        tot=tot+add;
    }

    printf("\nsomma = %d\n", tot);
    return 0;
}
```

Esercizio 5

Scrivere un programma che dato in ingresso un numero intero n , determini la somma dei numeri primi minori di n .

NB!!: due cicli annidati.

Esercizio 6

Scrivere un programma in grado di calcolare il fattoriale di un numero n inserito dall'utente.

NB!!: attenzione al massimo numero rappresentabile con il tipo di variabile utilizzata (`int`, `long int`, `long long int`). Per conoscere il numero di byte utilizzati per la memorizzazione di un `int`, `long int`, `long long int`;

`sizeof(int)`, `sizeof(long int)`, `sizeof(long long int)`