

Programmazione e Laboratorio di Programmazione

Manualistica 02 Gli operatori

Operatore &

- **Sintassi :**

&nome_variabile

- **Valore:**

indirizzo della variabile
nome_variabile

- **Esempio:**

```
int *x;  
int y;  
x = &y;
```

Operatore *

- **Sintassi :**

***nome_puntatore**

- **“Valore”:**

il nome della variabile il cui indirizzo è memorizzato in **nome_puntatore**

- **Esempio:**

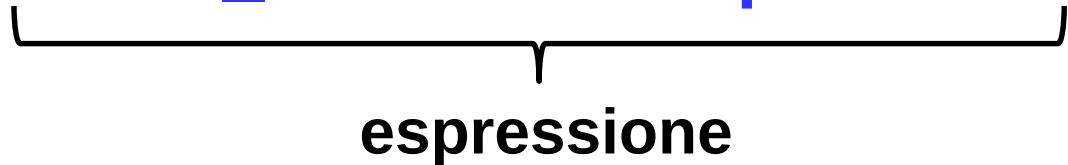
```
int *x, y;  
y = 1;  
x = &y;  
*x = *x + 1;
```

The diagram illustrates the memory state after the code execution. The variable `y` holds the value 1. The pointer variable `x` holds the address of `y` (`&y`). The final line of code, `*x = *x + 1;`, increments the value stored at the address pointed to by `x` (which is the address of `y`). The `*x` terms are circled in orange, and green arrows show the mapping from the variable `y` to these dereferenced pointers.

Operatore di assegnamento

- **Sintassi:**

nome_variabile = espressione



espressione

- **Valore:**

è il valore di **espressione**

- **Modifiche allo stato della memoria:**

assegna alla variabile

nome_variabile il valore di
espressione

Operatore di assegnamento

- **Attenzione:**

il tipo di **nome_variabile** e quello di **espressione** “devono” coincidere

Operatore di assegnamento

- **Esempio:**

```
/* sorgente: OpAss_1.c */  
/* programma che mostra il comportamento  
** dell'operatore di assegnamento */  
  
#include <stdio.h>  
  
int main ()  
{  
    /* definizione e inizializzazione delle variabili */  
    int x, y;  
    y=2;  
  
    /* visualizza il valore di y assegnandolo  
    ** al tempo stesso a x */  
    printf("\\nY: %d", x=y);  
  
    /* visualizza il valore di x */  
    printf("\\nX: %d", x);  
    return(1);  
}
```

Operatore di assegnamento

- **Sintassi:**

***nome_puntatore = espressione**



espressione

- **Valore:**

è il valore di **espressione**

- **Modifiche allo stato della memoria:**

assegna alla variabile il cui indirizzo è memorizzato in **nome_puntatore** il valore di **espressione**

Operatore di assegnamento

- **Esempio:**

```
/* sorgente: OpAss_2.c */  
/* programma che mostra il comportamento  
** dell'operatore di assegnamento */  
  
#include <stdio.h>  
  
int main ()  
{  
    /* definizione e inizializzazione delle variabili */  
    int x, y, *punx, *puny;  
    punx=&x; puny=&y;  
    *puny=2;  
    /* visualizza il valore di y assegnandolo  
    ** al tempo stesso a x */  
    printf("\\nY: %d", *punx=*puny);  
    /* visualizza il valore di x */  
    printf("\\nX: %d", *punx);  
    return(1);  
}
```


Operatore di auto-incremento prefisso

- **Sintassi:**

++nome_variabile

- **Valore:**

il valore di **nome_variabile**
incrementato di **1**

- **Modifiche allo stato della memoria:**

incrementa di **1** il valore di
nome_variabile

Operatore di auto-incremento prefisso

- **Esempio:**

B=++A;

2834			X	B
2835	4		X	
2836			X	
2837			X	
2838				
2839			X	A
2840	4		X	
2841			X	
2842			X	
2843				
2844				
2855				

Operatore di auto-incremento postfisso

- **Sintassi:**

nome_variabile++

- **Valore:**

il valore di **nome_variabile**

- **Modifiche allo stato della memoria:**

incrementa di **1** il valore di
nome_variabile

Operatore di auto-incremento postfisso

- **Esempio:**

B=A++;

2834		X	B
2835	3	X	
2836		X	
2837		X	
2838			
2839		X	A
2840	4	X	
2841		X	
2842		X	
2843			
2844			
2855			

Operatore di auto-decremento prefisso

- **Sintassi:**

--nome_variabile

- **Valore:**

il valore di **nome_variabile**
decrementato di **1**

- **Modifiche allo stato della memoria:**

decrementa di **1** il valore di
nome_variabile

Operatore di auto-decremento prefisso

- **Esempio:**

B=--A;

2834			X	
2835	2		X	B
2836			X	
2837			X	
2838				
2839			X	
2840	2		X	A
2841			X	
2842			X	
2843				
2844				
2855				

Operatore di auto-decremento postfisso

- **Sintassi:**

nome_variabile--

- **Valore:**

il valore di nome_variabile

- **Modifiche allo stato della memoria:**

**decrementa di 1 il valore di
nome_variabile**

Operatore di auto-decremento postfisso

- **Esempio:**

B=A--;

2834		X	B
2835	3	X	
2836		X	
2837		X	
2838			
2839		X	A
2840	2	X	
2841		X	
2842		X	
2843			
2844			
2855			

Operatori di auto-incremento e decremento

- **Esempio**

```
/* Mostra il comportamento degli operatori
** di autoincremento e autodecremento */

#include <stdio.h>

int main ()
{
    int y = 1;

    ➔ printf("\nValore di y +suffisso = %d\n", y++);
    ➔ printf("Valore di y +prefisso = %d\n", ++y);
    ➔ printf("Valore di y -prefisso = %d\n", --y);
    ➔ printf("Valore di y -suffisso = %d\n", y--);
    ➔ printf("Valore finale di y = %d\n", y);

    return(1);
}
```

1

3

2

2

1

Operatori di relazione

- **Sintassi:**

- | | |
|-------------------------------------|--------------------------|
| a) <code>espr_1 == espr_2</code> | uguale |
| b) <code>espr_1 != espr_2</code> | diverso |
| c) <code>espr_1 > espr_2</code> | maggiore |
| d) <code>espr_1 >= espr_2</code> | maggiore o uguale |
| e) <code>espr_1 < espr_2</code> | minore |
| f) <code>espr_1 <= espr_2</code> | minore o uguale |

- **Valore:**

- 1** se i valori di `espr_1` e `espr_2` si trovano nella relazione specificata
- 0** altrimenti

Operatori di relazione

• Esempio

```
/* Sorgente: OpRel.c */

/* programma che mostra il comportamento degli
** operatori di relazione */

#include <stdio.h>

int main ()
{
    printf ("\nValore di 3 < 5: %d\n", (3 < 5));
    printf ("Valore di 3 > 5: %d\n", (3 > 5));
    printf ("Valore di 3 == 5: %d\n", (3 == 5));
    printf ("Valore di 3 != 5: %d\n", (3 != 5));
    printf ("Valore di 5 == 5: %d\n", (5 == 5));

    printf ("Numero di relazioni soddisfatte: %d", \
            (3 < 5)+(3 > 5)+(3 == 5)+(3 != 5)+(5 == 5));

    return(1);
}
```

1

0

0

1

1

3

Operatori “logici”

- **Sintassi:**

$\text{espr}_1 \parallel \text{espr}_2$ “or”

- **Valore:**

0 se espr_1 e espr_2 valgono entrambe **0**
1 altrimenti

- **Sintassi:**

$\text{espr}_1 \&\& \text{espr}_2$ “and”

- **Valore:**

1 se espr_1 e espr_2 sono entrambe $\neq 0$
0 altrimenti

Operatori “logici”

- **Sintassi:**

!espr

“not”

- **Valore:**

1 se **espr** vale **0**

0 altrimenti

Operatori “logici”

- Esempio

```
/* sorgente: OpLog.c */
```

```
/* programma che illustra il comportamento degli  
** operatori logici */
```

```
#include <stdio.h>
```

```
int main ()  
{
```

```
➔ printf ("\nValore di !(3 > 5): %d\n", !(3 > 5));
```

```
➔ printf ("Valore di ((3 < 5) || (3 > 5)): %d\n", \  
        ((3 < 5) || (3 > 5)));
```

```
➔ printf ("Valore di ((3 < 5) && (3 > 5)): %d\n", \  
        ((3 < 5) && (3 > 5)));
```

```
return(1);  
}
```

1

1

0