Università degli Studi di Milano

Corso di Laurea in
Sicurezza dei Sistemi e delle Reti Informatiche

Lezione 3 – Input/Output elementare. Operatori, espressioni e istruzioni

FABIO SCOTTI

Laboratorio di programmazione per la sicurezza

Indice

1. Input/output elementare

Nelle prossime lezioni useremo dei comandi molto elementari per stampare a video il valore di alcune variabili e per scrivere nelle variabili dei valori immessi dall'utente dalla tastiera. Descriviamo ora come questo possa essere fatto.

1.1 Output: printf

In questa sezione vediamo come sia possibile dichiarare una variabile i intera, assegnarvi un valore e di stamparla a video.

Un semplice programma che esegue queste azioni e' il seguente

```
#include <stdio.h>
int main()
{
    int i;
    i = 3;
    printf("%d", i);
}
```

Nella parte dichiarativa dichiariamo la variabile i come variabile intera (int). Nella prima riga della parte esecutiva assegniamo il valore 3 alla variabile i.

La terza riga del main chiama la funzione printf che permette di stampare la variabile i passata (alla destra della virgola) a terminale. Alla sinistra della virgola è presente una stringa chiamata *stringa di controllo*. Essa ha il compito di definire in che modo il contenuto della variabile i debba essere stampato. Ad esempio "%d" significa che i verrà stampato come un intero: a monitor comparirà il numero 3.

Come già sappiamo, se scriviamo

```
printf("Hello WORD ");
```

compare a monitor `Hello WORD`

Ma se avessimo scritto

```
printf("Il valore contenuto e': %d", i)
```

sarebbe comparso a monitor

```
Il valore contenuto e': 3
```

Questo accade perché la funzione printf stampa la stringa di controllo e sostituisce alle parti della stringa che iniziano con % (in questo caso %d) i valori delle variabili passati alla destra della virgola (il valore di i).

Ritorneremo con maggiore dettaglio su questa funzione nel proseguo del corso.

1.2 Input: scanf

In questa sezione descriviamo come sia possibile dichiarare una variabile i intera, leggere un valore dalla tastiera e copiarlo in una variabile.

Un semplice programma che esegue queste azioni e' il seguente

```
#include <stdio.h>
int main()
{
    int i;
    printf("immetti un numero intero :");
    scanf("%d", &i);
    printf("il valore memorizzato e' %d", i); // lo stampiamo per controllo
}
```

Nella parte dichiarativa dichiariamo la variabile i come intera (int). Nella prima riga della parte esecutiva stampiamo un messaggio per l'utente affinché possa capire che gli stiamo chiedendo di battere un numero intero sulla tastiera. L'immissione si conclude automaticamente con la pressione del tasto enter.

La terza riga del main chiama la funzione scanf che permette di copiare il numero letto da tastiera nella variabile i. Alla sinistra della virgola è scritto come deve essere letto il dato immesso dalla tastiera. Nel nostro caso l'indicazione "%d" indica che deve essere letto come un numero intero.

Questa considerazione non e' banale. Infatti se l'utente digitasse 1024 sulla tastiera seguito dalla pressione dell'enter, il sistema potrebbe interpretare questo dato sia come il numero intero 1024 ma anche come la stringa composta dai caratteri '1', '0', '2' e '4'. Indicando %d, si toglie ogni possibilità sull'interpretazione di come il dato debba essere letto ovvero come un numero intero.

Alla destra della virgola è presente l'indirizzo dove questo valore letto deve essere memorizzato (&i significa "all'indirizzo di i").

Attenzione, non è corretto scrivere `scanf("%d", i);` senza &.

L'ultima riga del main semplicemente stampa il valore immesso per poterlo controllare nel modo che abbiamo già visto usando la printf.

Ritorneremo con maggiore dettaglio su questa funzione nel proseguo del corso.

2. Operatori, espressioni e istruzioni,

Il linguaggio C manipola i dati attraverso un insieme predefinito di operatori e di istruzioni. Vediamo in dettaglio questi componenti fondamentali.

Un ***programma C*** è una sequenza di istruzioni. Ad esempio i=3; è una istruzione. Le istruzioni terminano sempre con un punto e virgola.

In particolare una ***istruzione*** è una espressione terminata da un punto e virgola.

Una ***espressione*** è un insieme di variabili, costanti e richiami di funzione connessi da *operatori aritmetici*. Ad esempio a+b oppure x>3.

L' ***operatore*** è un simbolo che indica al linguaggio di eseguire un'operazione, o un'azione, su uno o più operandi. Ad esempio nella espressione a+b troviamo il segno + che è l'operatore, mentre a e b sono gli operandi.

In C gli ***operandi*** sono espressioni. Ricordo che una variabile è anch'essa una espressione.

Vediamo ora i tipi di espressioni e di operatori più usati.

2.1 Espressione di assegnamento

Una volta dichiarata una variabile è possibile utilizzarla per memorizzare un valore ed in seguito manipolarlo. Se si usa una variabile senza dichiararla il compilatore segnala un errore.

È possibile memorizzare un valore in una variabile con la seguente sintassi:
nome_variabile = *valore*;

Si faccia attenzione a non scambiare la variabile a cui assegnare il valore con quella che contiene il valore da assegnare: ciò che sta a sinistra dell'operatore `=` è la destinazione dell'assegnamento e può essere solo una variabile. Ciò che sta a destra è la sorgente e può essere qualsiasi espressione che dia un valore. L'esempio seguente:

```
a = 5;
```

memorizza il valore `5` nella variabile chiamata `a`. Sarebbe sbagliato sintatticamente scrivere l'espressione al contrario:

```
10 = a;
```

e come errore sintattico il compilatore lo segnalerebbe.

2.2 Espressioni ed operatori aritmetici

Si definisce *espressione aritmetica* un insieme di variabili, costanti e richiami di funzione connessi da *operatori aritmetici*. Il risultato di un'espressione aritmetica è sempre un valore numerico.

Ecco gli operatori aritmetici posizionati a seconda la loro priorità (alto: priorità massima; stesso livello: stessa priorità)

Negazione (- unario)		
Moltiplicazione (*)	Divisione (/)	Modulo (%)
Somma (+)	Sottrazione (-)	
Assegnamento (=)		

L'operatore modulo, %, calcola il resto della divisione intera tra i due operandi. E' possibile notare che anche l'assegnamento è un operatore.

E' sempre meglio usare le parentesi tonde per evidenziare (per chi legge, non per il compilatore) le priorità nella nostra espressione.

Leggete attentamente questi spunti

```
a = b = c = d + 1;  // NOOO
a = ( b = (c = d + 1));  // si !!

a [x = n *2, y = n++] = (b = 27) + (c = (f = e – 3));
                  // ok hai usato le parentesi...
                  // magari pero' e' meglio usare piu' righe....
```

Attenzione in C NON è presente l'elevamento a potenza. Pertanto 2^3 deve essere scritto come 2*2*2.

2.3 Operatori di incremento e decremento

Gli operatori ++ e -- sono di *incremento* e *decremento* rispettivamente, e possono essere applicati a variabili numeriche.

L'espressione ++i equivale a i = i + 1

Gli operatori di incremento e decremento possono essere sia *prefissi* che *postfissi*: essi possono apparire sia prima sia dopo l'operando a cui si riferiscono. Se l'operatore viene posto prima (prefisso) l'operazione viene eseguita prima di restituire il valore dell'espressione; se viene posto dopo (postfisso) l'operazione viene eseguita dopo aver utilizzato il valore originale. Ad esempio:

```
int i,a,b;
i = 15;
a = ++i;
b = i++;
printf("%d, %d, %d", a, b, i);
...
```

Il risultato che si ottiene è:

```
16, 16, 17
```

Infatti, l'espressione ++i incrementa i prima di stamparlo a video (quindi i è 16); la successiva i++ viene valutata al valore corrente di i (16) che verrà poi incrementato (a 17); infine l'espressione i è il valore di i dopo il postincremento (17).

Nel nostro corso l'operatore incremento e decremento verrà usato solo all'interno dei cicli FOR. L'uso non razionale di questi operatori rende facilmente il codice illeggibile per le altre persone.

2.4 Espressioni e operatori logici

Un'espressione logica è un'espressione che genera come risultato un valore vero o falso e viene utilizzata dalle istruzioni di controllo del flusso di esecuzione (if, while, ecc.).

La valutazione delle espressioni logiche può avere o un valore diverso da zero (interpretato come vero) oppure un valore pari a zero (interpretato come falso).

Un semplice esempio di espressione logica è una variabile: se il suo contenuto è diverso da zero, allora l'espressione è vera, altrimenti l'espressione è falsa.

Le espressioni logiche possono contenere gli operatori relazionali usate per confrontare tra loro dei valori. Ecco gli operatori logici posizionati a seconda la loro priorità (alto: priorità massima; stesso livello: stessa priorità)

maggiore(>) maggiore-uguale(>=) minore(<) minore-uguale (<=)
uguale (==) diverso (!=)

Infine, gli operatori logici consentono di concatenare fra loro più espressioni logiche e di negare il risultato di un'espressione logica; la scala di priorità è mostrata nella tabella successiva.

NOT Logico (!)
AND Logico (&&)
OR Logico (||)

ATTENZIONE:

Una direttiva del precompilatore (ad esempio #include) NON è una istruzione. Infatti si noti che non terminano con un punto e virgola come invece devono terminare tutte le istruzioni.

Non confondere una uguaglianza con un assegnamento! x==3 NON E' x=3;

x==3 è un espressione di uguaglianza che ritorna il valore 1 se x è uguale a 3, 0 viceversa. x non prende il valore 3!