CORSO DI PROGRAMMAZIONE
A.A. 2013-14

Dispensa 3

Laboratorio

Dott. Mirko Ravaioli
e-mail: mirko.ravaioli@unibo.it

3.2 Output formattato: printf()

Per visualizzare un numero, occorre utilizzare le funzioni per l'output formattato. La funzione *printf()* accetta un numero variabile di argomenti, come minimo uno. Il primo e unico argomento richiesto è la stringa di formato, che indica come formattare l'output. Gli argomenti opzionali sono variabili ed espressioni di cui visualizzare i valori Ad esempio:

- L'istruzione *printf("Ciao a tutti!")*; visualizza il messaggio: "Ciao a tutti" sullo schermo. In questo caso è utilizzato un unico argomento: la stringa di formato che contiene la stringa letterale da visualizzare sullo schermo
- L'istruzione *printf("%d",i);* visualizza il valore della variabile intera i. La stringa di formato contiene soltanto l'indicatore %d, che indica di mostrare un singolo intero decimale. Il secondo argomento è il nome della variabile
- L'istruzione *printf("%d più %d uguale %d",a, b, a+b);* visualizza (supponendo che le variabili 'a' e 'b' contengano rispettivamente 2 e 3) "2 più 3 uguale 5" sullo schermo. In questo caso vi sono quattro argomenti: una stringa di formato contenente il testo letterale e indicatori di formato, due variabili ed un'espressione di cui visualizzare i valori.

La stringa di formato di *printf()* può contenere quanto segue:

- Zero, uno o più comandi di conversione che indicano come visualizzare un valore dell'elenco degli argomenti. Un comando di conversione è costituito da % seguito da uno o più caratteri
- I caratteri che non fanno parte di un comando di conversione sono visualizzati così come sono.

Di seguito sono descritti i componenti del comando di conversione. Quelli tra parentesi sono opzionali:

%[modificatore][campoMinimo][precisione][modificatoreLunghezza]specificaConversione

dove:

- **modificatore** può essere una combinazione (in qualsiasi ordine) dei seguenti simboli (tra parentesi i simboli):
  - (-) il risultato della conversione e' allineato a sinistra all' interno del campo definito da campoMinimo, il default e' l' allineamento a destra
  - (+) specifica che il segno davanti al numero verrà sempre stampato. Il risultato di conversioni di tipi con segno inizia con + (se positivi) o - (se negativi); il default e' che il segno (-) appare solo davanti ai negativi
  - (spazio) inserisce uno spazio davanti al valore se il primo carattere non e' un segno
  - (#) formato alternativo (con piccole variazioni) per la maggior parte delle specifiche di conversione
    Se il carattere di conversione e' o (ottale) e se il valore da convertire e' diverso da zero, il primo carattere stampato e' 0.
    Se il carattere di conversione e' x o X (esadecimale) e se il valore da convertire non e' nullo, i primi caratteri stampati saranno 0x o 0X a seconda che la direttiva sia x o X.
  - 0 (zero) il campo viene riempito con zeri invece che con spazi (il default)
- **campoMinimo**: se il valore convertito ha meno caratteri del campo questo viene riempito da spazi, il valore di campoMinimo puo' essere un intero o *. In quest' ultimo caso il valore deve essere un intero inserito nella lista degli argomenti della printf, subito prima dell' espressione interessata da questo formato.

- **precisione**: un valore nella forma .n (con n intero oppure * se viene letto nella lista degli argomenti) dove n e' il :
    - minimo numero di cifre per le specifiche i,d,o,u,x,X
    - minimo numero di cifre dopo il punto decimale per le specifiche a,A,e,E,f,F
    - massimo numero di cifre significative per le specifiche g,G

- **modificatoreLunghezza**: dichiara che la successiva specifica di conversione deve essere applicata a un sottotipo intero particolare modificatore.
  La lettera h seguita da uno dei caratteri di conversione d, i, u, o, x o X indica che l'argomento e' uno short int o uno unsigned short int.
  Il modificatore l (elle) con gli stessi caratteri di conversione, indica che l'argomento e' un long int o uno unsigned long int.
  Il modificatore L seguito da uno dei caratteri di conversione e, E, f, g o G indica che l'argomento e' un long double
  il successivo specificatore (i o d) si applica ad una espressione
    - signed o unsigned char
    - signed o unsigned short int
    - signed o unsigned long int
    - signed o unsigned long long int

- **specificaConversione**: è l'unico obbligatorio (a parte %) Sotto sono elencati i caratteri di conversione ed il loro significato:

d, i	Visualizza un intero con segno in notazione decimale
u	Visualizza un intero senza segno di notazione decimale
o	Visualizza un intero in notazione ottale senza segno
x, X	Visualizza un intero in notazione esadecimale senza segno. Si utilizza 'x' minuscolo per l'uotput minuscolo e 'X' maiuscolo per l'output maiuscolo
c	Visualizza un singolo carattere (indicato dal codice ASCII)
e, E	Visualizza un float o un double in notazione scientifica (ad esempio 123.45 è visualizzato come 1.234500e+002) sei cifre sono visualizzate a destra del punto decimale a meno che non sia specificata un'altra precisione con l'indicatore f. si utilizzano 'e' o 'E' per l'output minuscolo o maiuscolo
f	Visualizza un float o un double in notazione decimale (ad esempio 123.45 è visualizzato come 123.450000). Sei cifre sono visualizzate a destra.
s	Visualizza una stringa
%	Visualizza il carattere %

La funzione printf() restituisce un valore che individua il numero di caratteri stampati, se il valore è negativo segnala un errore.

3.3 Le Istruzioni

Un' *istruzione* è un comando completo che indica al computer di eseguire un particolare compito. In generale in C le istruzioni vengono scritte una per ogni riga anche se alcune possono occupare anche più righe. Tutte le istruzioni in C (eccetto le direttive #define, #include...) devono terminare con un punto e virgola (;).

3.3.1 Spazi bianchi e istruzioni

Con *spazi bianchi* ci si riferisce ai veri e propri spazi, ai caratteri di tabulazione e alle righe vuote presenti nel codice sorgente. Il compilatore C non considera gli spazi bianchi e quando legge un istruzione dal codice sorgente cerca il carattere terminatore (il punto e virgola) ignorando tutti gli spazi bianchi presenti. Quindi l'istruzione:

```
y=4+3;
```

è equivalente a quella seguente:

```
y = 4 + 3;
```

che a sua volta è equivalente a:

```
y =
4
+
3;
```

In questo modo viene lasciata al programmatore la scelta sulla modalità di formattazione del proprio codice. Comunque non è consigliato utilizzare una formattazione come nell'ultimo esempio: le istruzioni dovrebbero essere introdotte una per riga seguendo uno schema standard per la spaziatura delle variabili e degli operatori. Guardare i vari esempi inseriti nelle dispense o i sorgenti in allegato per rendersi conto di come poter formattare il proprio codice.

Il C ignora gli spazi bianchi eccetto quando questi si trovano all'interno di costanti stringa letterali: infatti in questo caso gli spazi vengono considerati come parti delle stringhe (quindi come caratteri della stringa). Una *stringa* è una sequenza di caratteri e in particolare le *costanti stringhe letterali* sono stringhe racchiuse tra doppi apici che vengono interpretate dal compilatore in maniera assolutamente letterale, spazi compresi. Quindi per esempio l'istruzione (anche se solitamente non usata):

```
printf(
"Forza Cesena!"
);
```

è corretta, mentre quella che segue genera un errore in fase di compilazione:

```
printf("Forza
Cesena");
```

Per andare a capo in un'istruzione, quando ci troviamo in corrispondenza di una costante stringa letterale occorre utilizzare il carattere barra inversa (\) prima dell'interruzione. La forma corretta per l'esempio sopra riportato quindi sarà:

```
printf("Forza\
Cesena");
```

3.3.2 Istruzioni nulle

Se si inserisce un punto e virgola da solo su una riga si ottiene *istruzione nulla* cioè un'istruzione che non esegue alcuna operazione.

3.3.3 Istruzioni composte

Un'*istruzione composta*, solitamente chiamata *blocco*, è un gruppo di due o più istruzioni C racchiuse tra parentesi graffe. Ad esempio la porzione di codice che segue è un blocco:

```
{
      printf("Forza Cesena!");
```

```
        printf("Devi vincere");
}
```

In C i blocchi possono essere utilizzati in qualsiasi punto in cui sia possibile utilizzare un'istruzione singola. Le parentesi graffe possono essere posizionate in qualsiasi punto ad esempio:

```
{printf("Forza Cesena!");
printf("Devi vincere");}
```

comunque si consiglia di posizionare le graffe su righe diverse, mettendo così in evidenza l'inizio e la fine del blocco. In questo modo, oltre che rendere il codice più leggibile, è possibile accorgersi se ne è stata dimenticata qualcuna.

3.4 Le Espressioni

In C un'espressione è una qualsiasi cosa che deve essere valutata come un valore numerico. Le espressioni in C possono avere qualsiasi livello di complessità.

3.4.1 Espressioni Semplici

Le espressioni più semplici consistono di un unico oggetto, ad esempio una variabile, una costante letterale o simbolica. Le costanti letterali vengono valutate secondo il loro valore, le costanti simboliche invece con il valore assegnato loro dalla direttiva #define, le variabili vengono valutate con il valore assegnate loro dal programma.

Espressione	***Descrizione***
TASSO	Una costante simbolica
28	Una costante letterale
stipendio	Una variabile
3.678	Una costante letterale

3.4.2Espressioni Complesse

Le espressioni complesse consistono di più espressioni semplici combinate tra di loro attraverso degli operatori. Per esempio:

```
3 + 9
```

è un espressione formata dalle due costanti letterali 3 e 9 e dall'operatore somma +. L'espressione 3 + 9 viene valutata come 10. Si possono scrivere anche espressioni molto più complesse:

```
3.78 + 56 – stipendio * mesi / giorni
```

Quando un'espressione contiene più operatori, come nell'esempio sopra riportato, il risultato dipende dalla precedenza degli operatori.

Consideriamo la seguente espressione:

```
y = b + 17;
```

in questo caso nell'istruzione viene calcolato il valore dell'espressione b + 17 e viene assegnato il risultato alla variabile y. A sua volta l'istruzione y = b + 17 è un'altra espressione che ha il valore nella variabile a sinistra dell'uguale, quindi è possibile anche la scrittura:

```
k = y = b + 17;
```

in questo caso il risultato viene assegnato sia a y che a k, in particolare prima viene valutata l'espressione b + 17, il suo valore assegnato alla variabile y, ed in fine il valore di y assegnato alla variabile k. Quindi alla fine dell'espressione k e y avranno lo stesso valore.

In C sono possibili anche espressioni di questo tipo:

```
k = 8 + (y = 3 + 4);
```

in questo caso dopo l'esecuzione dell'istruzione la variabile y avrà il valore 7, mentre la variabile x il valore 15. In questo caso però le parentesi sono essenziali per la corretta compilazione dell'istruzione.

3.5 Gli Operatori

Un *operatore* è un simbolo che indica al linguaggio di eseguire un'operazione, o un'azione su uno o più operandi. In C tutti gli operatori sono visti come delle espressioni.

3.5.1 Operatore di Assegnamento

L'operatore di assegnamento è il simbolo uguale (=). Il suo significato e utilizzo all'interno di un programma è diverso dal suo consueto utilizzo in matematica. Scrivendo:

```
y = k;
```

NON significa che y è uguale a k ma invece **assegna il valore di k a y**. In un'istruzione di assegnamento la parte a destra del segno uguale può essere una qualsiasi espressione, mentre la parte di sinistra deve essere il nome di una variabile, quindi la sintassi risulta la seguente:

```
variabile = espressione;
```

Quando l'istruzione viene eseguita prima viene valutata l'espressione poi il risultato viene assegnato alla variabile. Quindi per esempio:

```
y = 8 + 9;
```

prima viene calcolata la somma di 8 + 9 poi il risultato viene assegnato alla variabile y. Alla fine dell'istruzione avremo che y avrà il valore 17. Considerando invece:

```
y = 11;
k = y + 7;
```

prima viene assegnato a y il valore 11 poi, nell'istruzione successiva, viene calcolato y + 7 e il risultato assegnato a k. Quindi alla fine delle 2 righe di codice sopra riportate avremo che y sarà uguale a 11 e k a 18.

3.5.2 Operatori matematici

Gli operatori matematici effettuano operazioni aritmetiche e si dividono in due categorie: operatori unari e operatori binari.

Gli **operatori matematici binari** operano su due operandi. Questi operatori che comprendono anche le principali operazioni aritmetiche sono elencati nella tabella sottostante:

Operatore	***Simbolo***	***Azione***	***Esempio***
Addizione	+	Somma i due operandi	x + y
Sottrazione	-	Sottrae il secondo operando dal primo	x - y
Moltiplicazione	*	Moltiplica i due operandi	x * y

Divisione	/	Divide il primo operando per il secondo	x / y
Resto (modulo)	%	Fornisce il resto della divisione del primo operando per il secondo	x % y

Alcuni esempi di utilizzo:

```
/* Primo esempio di utilizzo degli operatori matematici */
#include <stdio.h>

int main()
{
      int a, b, ris;

      a = 10;
      b = 5;
      ris = a – b;
      printf("risultato di %d-%d=%d\n",a,b,ris);

      a = 7;
      b = 3;
      ris = a * b;
      printf("risultato di %d*%d=%d\n",a,b,ris);

      a = 21;
      b = 7;
      ris = a / b;
      printf("risultato di %d/%d=%d\n",a,b,ris);

      a = 21;
      b = 10;
      ris = a / b;
      printf("risultato di %d/%d=%d\n",a,b,ris);

      a = 13;
      b = 5;
      ris = a % b;
      printf("risultato di %d %% %d=%d\n",a,b,ris);

      return 0;
}
```

Gli **operatori matematici unari** hanno questo nome in quanto richiedono un unico operando. Il C dispone di due operatori unari: incremento e decremento e possono essere utilizzati solo con le variabili e mai con le costanti. Il loro scopo è quello di aggiungere o sottrarre un'unità dall'operando specificato.

Operatore	***Simbolo***	***Azione***	***Esempio***
Incrementa	++	Incrementa l'operando di una unità	y++ ++y
Decrementa	--	Decrementa l'operando di una unità	y-- --y

Ad esempio le istruzioni:

```
++x;
--y;
```

equivalgono a:

```
x = x + 1;
```

```
y = y - y;
```

Gli operatori possono essere *postfissi* o *prefissi*. Queste due sintassi non sono equivalenti, ma differiscono per quanto riguarda il momento in cui viene effettuata l'operazione:

- Gli operatori prefissi modificano il proprio operando prima che ne venga utilizzato il valore
- Gli operatori postfissi modificano il proprio operando dopo avere utilizzato il valore

Vediamo un esempio:

```
x = 7;
y = x++;
```

Dopo l'esecuzione di queste due istruzioni x vale 8 e y vale 7. Prima il valore di x è stato assegnato a y e solo allora x è stato incrementato. Se consideriamo invece le seguenti istruzioni:

```
x = 7;
y = ++x;
```

Dopo l'esecuzione di queste due istruzioni x vale 8 e y vale 8. L'operatore = è l'operatore di assegnamento e non un operatore di confronto. Eventuali successive modifiche al valore di x non hanno effetto su y.

Consideriamo un esempio:

```
#include <stdio.h>

int main()
{
      int a, b;

      a = 7;
      b = 7;

      printf("%d  %d\n",a--,--b);
      printf("%d  %d\n",a--,--b);
      printf("%d  %d\n",a--,--b);
      printf("%d  %d\n",a--,--b);
      printf("%d  %d\n",a--,--b);

      return 0;
}
```

Il risultato del programma:

```
7   6
6   5
5   4
4   3
3   2
```

La precedenza degli operatori segue il seguente ordine:

- incrementi e decrementi unari
- moltiplicazioni, divisioni e resti
- somme e sottrazioni

Se un'espressione contiene più di un operatore con lo stesso valore di precedenza le relative operazioni vengono eseguite da sinistra a destra. E' possibile modificare la precedenza attraverso l'utilizzo di parentesi tonde.

Ad esempio:

```
y = 4 + 2 * 3;
```

Dopo l'istruzione y vale 10. Prima viene eseguito 2*3 al risultato viene aggiunto 4 e il risultato assegnato a y.

Consideriamo:

```
y = 12 % 5 * 2;
```

L'operatore % (modulo) e * (moltiplicazione) hanno la stessa precedenza quindi le istruzioni vengono eseguite da sinistra a destra: prima viene calcolato 12%5 che viene 2 che moltiplicato per 2 fa 4 quindi alla fine a y viene assegnato il valore 4.

Se utilizziamo le parentesi:

```
y = 12 % (5 * 2);
```

Prima viene eseguita la sotto espressione individuata dalle parentesi: 5*2 poi il calcolo del modulo. Quindi alla fine dell'istruzione a y viene assegnato il valore 2.