Capitolo 3

Input/Output formattato

La funzione `printf`

- Per stampare i valori di variabili con `printf` dobbiamo specificare un ***stringa di formato*** seguita dai valori che devono essere inseriti nella stringa:

  `printf(`*string*`,` *expr*1`,` *expr*2`,` …`);`
- La stringa di formato specifica il formato dell'output e può contenere sia caratteri espliciti da stampare sia delle ***specifiche di conversione***, che iniziano con il carattere `%`.
- Una specifica di conversione è ***un segnaposto*** che indica dove deve essere inseriro il valore
  - `%d` è un segnaposto per valori `int`
  - `%f` è un segnaposto per valori `float`

La funzione `printf`

- Caratteri espliciti vengono stampati così come specificati nella stringa di formato. I segnaposto, invece, vengono sostituiti con i valori delle variabili.
- Esempio:

```
int i, j;
float x, y;

i = 10;
j = 20;
x = 43.2892f;
y = 5527.0f;

printf("i = %d, j = %d, x = %f, y = %f\n", i, j, x, y);
```

- Output:

```
i = 10, j = 20, x = 43.289200, y = 5527.000000
```

La funzione `printf`

- I compilatori non controllano che il numero di segnaposto sia uguale al numero di variabili o espressioni specificati nella chiamata alla funzione

- Esempio di chiamata con troppi segnaposti

```
printf("%d %d\n", i);    /*** WRONG ***/
```

- Esempio di chiamata con troppe variabili

```
printf("%d\n", i, j);    /*** WRONG ***/
```

La funzione `printf`

- Il compilatore non controlla che la specifica di conversione del segnaposto sia appropriata

- Se il programmatore usa un segnaposto non appropriato l'output sarà senza senso

```
int i;
float x;
printf("%f %d\n", i, x);   /*** WRONG ***/
```

Specifiche di conversione

- La specifica di conversione ha la forma `%`*m*`.`*p*X oppure la forma `%-`*m*`.`*p*X
  - dove *m* e *p* sono costanti intere e X è una lettera
- Nella specifica di conversione `%10.2f`
  - *m* è 10, *p* è 2, e *X* è `f`.
- Sia *m* che *p* sono opzionali
  - se *p* viene omesso allora anche il punto fra *m* e *p* viene omesso
- Nel segnaposto `%10f`
  - *m* è 10 e *p* (insieme al punto) è stato omesso
- Nel segnaposto `%.2f`
  - *p* è 2 mentre *m* è stato omesso.

Specifiche di conversione

- Il valore *m* specifica la ***grandezza minima del campo*** cioè il numero minimo di caratteri da stampare
- Se il valore da stampare richiede meno caratteri allora saranno aggiunti degli spazi bianchi alla sinistra dal valore da stampare (allineamento a destra)
  - `%4d` stampa il numero 123 come •`123`
  - • rappresenta lo spazio
- Se il valore da stampare richiede più di *m* caratteri allora il campo verrà esteso per accomodare tutti i caratteri necessari
- Specificare il segno meno significa utilizzare l'allineamento a sinistra
  - Il segnaposto `%-4d` per il numero 123 produce `123`•

Specifiche di conversione

- Il significato della ***precisione***, *p*, dipende dalla scelta della lettera *X*
- La conversione `d` viene usata per stampare numeri interi in notazione decimale
  - *p* indica il minimo numero di cifre da stampare (degli zero addizionali sono stampati all'inizio del numero se necessario)
  - Se *p* viene omesso, si utilizza il valore di default 1

Specifiche di conversione

- Valori floating-point:

  `e` — Formato esponenziale . *p* indica quante cifre devono apparire dopo il punto (il default è 6). Se *p* è 0, non viene visualizzato nemmeno il punto decimale.

  `f` — Formato decimale . *p* ha lo stesso significato del caso `e`.

  `g` — Indica `e` oppure `f`, dipende dalla grandezza del valore da stampare
  - *p* indica il numero massimo di cifre significative da visualizzare
  - Se il valore da stampare non ha cifre decimali non verrà stampato il punto decimale

Programma: Uso della `printf`

- Il programma `tprintf.c` usa la funzione `printf` per stampare interi e numeri con virgola (float), in vari formati

tprintf.c

```
/* Prints int and float values in various formats */

#include <stdio.h>

int main(void)
{
  int i;
  float x;

  i = 40;
  x = 839.21f;

  printf("|%d|%5d|%-5d|%5.3d|\n", i, i, i, i);
  printf("|%10.3f|%10.3e|%-10g|\n", x, x, x);

  return 0;
}
```

- Output:

```
|40|   40|40   |  040|
|   839.210| 8.392e+02|839.21    |
```

Sequenze di "escape"

- La sequenza di 2 caratteri `\n` è una ***sequenza di escape.***
- Le sequenze di escape permettono di inserire caratteri di controllo (non stampabili) all'interno di una stringa
- I caratteri di controllo hanno significati particolari
- Esempi di sequenze di escape sono:

Alert (bell)	`\a`
Backspace	`\b`
New line	`\n`
Horizontal tab	`\t`

Sequenze di "escape"

- Una stringa può contenere quante sequenze di escape si vuole:

```
printf("Item\tUnit\tPurchase\n\tPrice\tDate\n");
```

- La precedente `printf` produce il seguente output su due linee:

```
Item    Unit    Purchase
        Price   Date
```

Sequenze di “escape”

- Un’altra sequenza di escape è `\"`.
  Essa rappresenta il carattere `"`:

```
printf("\"Hello!\"");
  /* prints "Hello!" */
```

- Per stampare il carattere `\`, si usa la sequenza di escape `\\`:

```
printf("\\");
  /* prints one \ character */
```

La funzione `scanf`

- `scanf` legge l'input in accordo ad una stringa di formato

- Una stringa di formato per la `scanf` può contenere sia caratteri normali sia segnaposto
  - È simile alla stringa di formato della `printf`

- Le specifiche di conversione riconosciute dalla `scanf` sono essenzialmente le stesse dellla `printf`.

La funzione `scanf`

- In molti casi, la `scanf` avrà una stringa di formato che contiene solo segnaposti:

```
int i, j;
float x, y;

scanf("%d%d%f%f", &i, &j, &x, &y);
```

- Esempio di input:

```
1 -20 .3 -4.0e3
```

`scanf` assegnerà i valori 1, –20, 0.3, e –4000.0 alle variabili `i`, `j`, `x`, e `y` (nell'ordine indicato).

La funzione `scanf`

- Quando usa la `scanf`, il programmatore deve controllare
  - che il numero di segnaposti sia uguale al numero di variabili di input
  - che il tipo specificato nel segnaposto corrisponda al tipo della variabile

- Il simbolo `&` è richiesto ed è responsabilità del programmatore metterlo

Come opera la funzione `scanf`

- `scanf` prova a raggruppare i caratteri dell'input
  - in modo da soddisfare le specifiche di conversione della stringa di formato
- Per ogni segnaposto, `scanf` cerca di individuare i giusti caratteri, tralasciando gli spazi bianchi dell'input, se necessario

- Individuati i caratteri, `scanf` li legge fermandosi quando incontra un carattere che non può appartenere al formato richiesto
  - Se riesce nell'operazione `scanf` assegna il valore alla variabile e continua con le successive variabili
  - Altrimenti, `scanf` si ferma (anche se ci sono altre variabili alle quali assegnare un valore).

Come opera la funzione `scanf`

- Quando cerca un numero, `scanf` ignora lo ***spazio bianco*** (***white-space characters)***:
  - spazio, tab, form-feed, e new-line.
- Esempio: la chiamata a `scanf` che legge 4 numeri:

```
scanf("%d%d%f%f", &i, &j, &x, &y);
```

- I numeri possono essere su una sola linea o su più linee:

```
   1
-20   .3
   -4.0e3
```

- `scanf` usa un flusso di caratteri (¤ rappresenta new-line):

```
••1¤-20•••.3¤•••-4.0e3¤
ssrsrrrsssrrssssrrrrrr
```

(`s` = skipped; `r` = read)

- `scanf` "controlla" il new-line finale senza leggerlo

Come opera la funzione `scanf`

- Quando deve leggere un numero intero, `scanf`
  - Prima trova una cifra, un segno più o un segno meno
  - Poi legge le successive cifre fino ad un carattere che non è una cifra
- Quando deve leggere un numero floating-point
  - Prima cerca un segno + o – (opzionale) seguito da
  - cifre (opzionalmente con un punto decimale) seguite da
  - Un esponente (opzionale). L'esponente consiste di una lettera `e` (oppure `E`), un segno opzionale e una o più cifre
- `%e`, `%f`, e `%g` sono equivalenti per la `scanf`.

Come opera la funzione `scanf`

- Quando la `scanf` trova un carattere che non può essere parte della conversione richiesta
  - Rimette il carattere nel flusso di input (stream)
  - Il carattere sarà letto successivamente (dalla stessa scanf o da altre chiamate a scanf)

Come opera la funzione `scanf`

- Esempio di input:

  `1-20.3-4.0e3¤`

- La chiamata a `scanf` è quella precedente:

  `scanf("%d%d%f%f", &i, &j, &x, &y);`

- In questo caso `scanf` elabora l'input come segue:
  - `%d`. Memorizza 1 in `i` e rimette il carattere − nel flusso di input
  - `%d`. Memorizza –20 in `j` e rimette il carattere `.` nel flusso di input
  - `%f`. Memorizza 0.3 in `x` e rimette il − nel flusso
  - `%f`. Memorizza -4.0×10^3 in `y` e rimette il new-line nel flusso di input

Caratteri normali nelle stringhe di formato

- Quando la stringa di formato contiene caratteri "bianchi", `scanf` legge caratteri bianchi fino a quando non raggiunge un carattere non-bianco
  - Che viene rimesso nel flusso di input
- Quando la stringa di formato contiene un carattere non-bianco, `scanf` lo confronta con il prossimo carattere nel flusso di input
  - Se sono uguali, `scanf` elimina il carattere dall'input e continua la lettura usando la stringa di formato.
  - Se non corrispondono, `scanf` rimette il carattere letto nel flusso di input e termina l'esecuzione
    - anche se la stringa di formato non è terminata

Caratteri normali nelle stringhe di formato

- Esempi: la stringa di formato è "`%d/%d`"
  - Se il flusso di input è: •`5/•96`, `scanf` va a buon fine.
  - Se l'input è •`5•/•96` , `scanf` non va a buon fine perchè / nella stringa di formato non corrisponde allo spazio nel flusso di input

- Per permettere la presenza di spazio dopo il primo numero si deve usare la stringa di formato "`%d /%d`"

Differenze fra `printf` e `scanf`

- Sebbene le chiamate a `scanf` e `printf` siano simili ci sono differenze importanti fra le due funzioni

- Un errore comune è quello di usare il simbolo `&` per le variabili specificate nella `printf`:

```
printf("%d %d\n", &i, &j);  /*** WRONG ***/
```

Differenze fra `printf` e `scanf`

- Un altro errore comune è quello di assumere che le stringhe di formato della `scanf` debbano somigliare a quelle della `printf`
- Si consideri la seguente chiamata a `scanf`:

```
scanf("%d, %d", &i, &j);
```

  - `scanf` cercherà un valore intero nell'input e lo assegnerà alla variabile `i`.
  - Successivamente `scanf` cercherà il carattere `,` nell'input
  - Se nell'input c'è uno spazio e non una virgola, `scanf` termina l'esecuzione senza assegnare un valore a `j`.

Differenze fra `printf` e `scanf`

- Inserire un carattere di new-line alla fine della stringa di formato di una `scanf` non è una buona idea
- Per `scanf`, un carattere di new-line è equivalente ad uno spazio; entrambe richiedono alla `scanf` di avanzare fino al prossimo carattere non-bianco
- Se il formato è `"%d\n"`, `scanf` salterà i caratteri bianchi, leggerà un intero, e salterà di nuovo tutti i caratteri bianchi fino al successivo carattere non bianco
  - Il programma potrebbe fermarsi in attesa che l'utente digiti un carattere non-bianco

Programma: Somma di frazioni

- Il programma `addfrac.c` chiede all'utente due frazioni e stampa la somma

- Esempio di esecuzione:

```
Enter first fraction: 5/6
Enter second fraction: 3/4
The sum is 38/24
```

addfrac.c

```
/* Adds two fractions */

#include <stdio.h>

int main(void)
{
  int num1, denom1, num2, denom2, result_num, result_denom;

  printf("Enter first fraction: ");
  scanf("%d/%d", &num1, &denom1);

  printf("Enter second fraction: ");
  scanf("%d/%d", &num2, &denom2);

  result_num = num1 * denom2 + num2 *denom1;
  result_denom = denom1 * denom2;
  printf("The sum is %d/%d\n",result_num, result_denom)

  return 0;
}
```

… arrivederci alla prossima lezione