Introduzione alla programmazione in C

Barbara Guidi

Dipartimento di Informatica
Università di Pisa

Corso Informatica I - 2012/2013

Outline

Ricapitolando...

- Errori/warning a tempo di compilazione
  - Ma è giusto pensare che tutto vada bene solamente perché un programma compila senza errori?
- Approfondimento su printf ed esempi d'uso

Problemi di esecuzione

Eventuali errori durante l'esecuzione del programma sono chiamati **errori di esecuzione**

- non necessariamente terminano l'esecuzione del programma.
- sono quasi sempre molto più subdoli e difficili da individuare di quelli sintattici.
- Alcuni errori saranno palesi durante l'esecuzione. Altri saranno errori logici, in cui tutto funziona anche se non come dovrebbe.

Errori a tempo di esecuzione (1)

```
#include <stdio.h>
int main(void) {
    int dividendo = 42;
    int divisore = 0;
    int risultato;
    risultato = dividendo / divisore;
    printf("Il risultato e' %d\n", risultato);
    return 0;
}
```

- **programma** `divisione.c`
- **cosa c'è che non va?**

Errori a tempo di esecuzione (2)

```
-> gcc -Wall -pedantic divisione.c -o divisione
-> ./divisione
Floating exception
```

- gli errori come la divisione per zero avvengono durante l'esecuzione del programma e sono errori fatali, ovvero, provocano la terminazione immediata del programma

Errori a tempo di esecuzione (3)

```
#include <stdio.h>
int main(void) {
    int addendo1 = 44;
    int addendo2 = 2;
    int risultato;
    risultato = addendo1 - addendo2;
    printf("Gli addendi sono %d e %d\n", addendo1,
addendo2);
    printf("Il risultato della somma e' %d\n",
risultato);
    return 0;
}
```

- il programma si chiama `somma.c`
- E' corretto?

Errori a tempo di esecuzione (4)

```
-> ./somma
Gli addendi sono 44 e 2
Il risultato della somma e' 42
```

- Sembra ovvio vedere cosa c'è di sbagliato.
- Ma questo tipo di errori è il più difficile da trovare e correggere.
- Spesso non ci accorgiamo nemmeno che c'è un errore.
- Notate come scegliere nomi di variabili **SIGNIFICATIVI** aiuti a trovare il problema!

Errori a tempo di esecuzione (5)

Vediamo uno dei classici errori con gli interi (errori di overflow/underflow):

```
#include <stdio.h>
#include <limits.h>
int main(void) {
int a;
a=INT_MAX;
printf("int a (INT_MAX) =%d, int a (INT_MAX) + 1 =%d\n",
a,a+1);
return 0;
}
```

```
-> int a (INT_MAX) =2147483647, int a (INT_MAX) + 1
=-2147483648
```

Input e Output in C

- Input con formato: scanf("stringa formato", lista variabili), legge dal dispositivo di ingresso (stdio) un valore e lo memorizza in una variabile;
- Output con formato: printf("stringa formato", lista variabili), scrive sul dispositivo standard di uscita (stdout);
  - Tramite la stringa di formato, si specifica "come"
  - Tramite i parametri successivi, si specifica "che cosa"

Nuova istruzione:scanf

La funzione scanf :

- legge da standard input (stdin, tastiera) dati complessi
- è quasi duale a printf (prende un formato e una lista di parametri) ma non del tutto
- è facilmente fonte di errori e incongruenze

scanf: il formato

```
int scanf(const char *format, ...);
```

- Il formato è una stringa costante che descrive esattamente quello che deve esserci in input e (non può essere una variabile stringa) che può contenere :
  - whitespaces: spazi,tabulazioni, invii a capo (\n)
  - caratteri standard (qualunque sequenza di caratteri non bianchi che non inizi per %)
  - specificatori di formato (che iniziano per %)

scanf: specificatori di formato

Gli specificatori di formato sono così fatti :

`%[*][width][modifiers]type`

- `*` Un asterisco fa si che i dati corrispondenti a questo argomento vengano letti dallo stdin ma ignorati (non salvati in un argomento)
- `width` Specifica il numero massimo di caratteri da leggere per questo argomento
- `modifiers` Modifica le dimensioni:
  - `h` : short int (per d, i and n), o unsigned short int (per o, u and x)
  - `l` : (è una elle minuscola) long int (per d, i and n), o unsigned long int (per o, u and x), o double (per e, f and g)
  - `L` : long double (per e, f e g)
- `type` Un carattere che specifica il tipo di dato da leggere (vedi sotto)

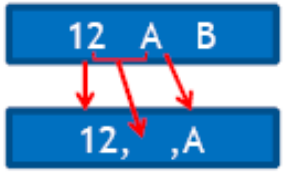
Problemi leggendo sulla stessa riga

Supponiamo di voler leggere sulla stessa riga più parametri.


```
scanf("%d%d",&intero1,&intero2);
```

Lo spazio è considerato un separatore e viene scartato
Però lo spazio è un carattere:

```
scanf("%d%c%c",&intero1,&car1,&car2);
printf("%d,%c,%c",intero1,car1,car2);
```



La scanf ha preso lo spazio come fosse il carattere che si intendeva leggere.

Soluzione: usare un separatore

Usare un separatore (anche lo spazio stesso)

```
scanf("%d %c %c",&intero1,&car1,&car2);
```

Uso un separatore (spazio) all'interno della stringa formato

12 A B

```
printf("%d,%c,%c",intero1,car1,car2);
```

12,A,B

scanf: la logica

- L'input da tastiera è bufferizzato e resta nel buffer finchè qualcuno non lo rimuove.
  - i caratteri letti/scritti sono memorizzati in un buffer (un'area di memoria che funge da contenitore temporaneo dei byte in attesa che questi vengano letti (in input) o scritti (in output))
- scanf cerca nel buffer i pattern (numeri decimali, floating poing, caratteri, ecc...) e "mangia" (cioè toglie dal buffer) quelli che riconosce
- I sistemi operativi rappresentano "a capo" con 1 o 2 caratteri di controllo:
  - Windows usa \r\n (carriage return + linefeed). Il primo è interpretato (correttamente) come separatore Il secondo rimane nel buffer ed è (erroneamente) interpretato come carattere inserito dall'utente

Possibili soluzioni

Prima soluzione:

```
printf("Inserire un numero:  ");
scanf("%f",&num1);
fflush(stdin); // si vuota il buffer
```

Seconda soluzione (equivalente alla prima):

```
printf("Inserire un numero reale:  ");
scanf("%f",&num2);
printf("Inserire un carattere:  ");
scanf("\n%c",&carattere);
printf("Letti:  %f,%c",num2, carattere);
```

scanf: esempi d'uso

Leggere una data (gg/mm/aaaa):

```
scanf("%2d/%2d/%4d", &giorno, &mese, &anno);
```

Leggere un'operazione aritmetica (su una riga):

```
scanf("%lf %c %lf", &num1, &simb, &num2);
```

Trasformare un esadecimale in decimale:

```
scanf("%x", &numero);
```

Leggere un singolo carattere:

```
scanf(" %c", &carattere);
```

Particolari da tenere a mente

Ci sono alcuni punti che è bene tenere a mente e ricordare.

1. La funzione scanf ritorna il numero di elementi correttamente letti: se leggete più di un elemento per volta, dovreste controllare il valore ritornato per assicurare che tutte le variabili siano inizializzate.
2. Per leggere un singolo carattere **DOVETE** premettere il `%c` (nel formato) con uno spazio bianco: come spiegato prima, il `%c` non elimina gli spazi bianchi rimasti sul buffer ed in particolare ogni invio a capo residuo da scanf precedenti!