

# Lezione 7: Funzioni di I/O avanzate

Laboratorio di Elementi di Architettura e Sistemi Operativi

27 Marzo 2013

## Vettori e stringhe

### I vettori in C

- Uno dei tipi non primitivi più usati in C è il *vettore* (o *array*)
- Definizione:

```
tipo nome[dimensione];  
tipo nome[] = {elem_0, elem_1, ..., elem_N};
```

- Gli elementi di un vettore si identificano mettendo l'indice tra parentesi quadre:

```
int v[10];  
v[0] = 3;
```

- *Gli indici dei vettori partono da ZERO!*

Esempio:

```
#include <stdio.h>  
  
main()  
{  
    float v[5];  
    int cont, ret, i;  
  
    printf("Inserisci max 5 numeri; premi q per terminare prima\n");  
  
    for(cont = 0; cont < 5; cont++)  
    {  
        ret = scanf("%f", &(v[cont]));  
        if(ret==0) break;  
    }  
  
    for(i = 0; i < cont; i++)  
    {  
        printf("%f\n", v[i]);  
    }  
}
```

## Le stringhe in C

- Le stringhe in C sono *vettori di char* terminati dal carattere NULL
- Definizione:

```
char str[10];  
char str[] = "Ciao!";
```

C	i	a	o	!	\0
---	---	---	---	---	----

- *NOTA*: la stringa vuota non è un vettore vuoto!

```
char vuota[] = "";
```

\0
----

## Vettori e stringhe non sono tipi base!

1. Non è possibile fare assegnamenti tra vettori/stringhe

```
int a[5], b[5];  
char str[10];  
a = b; // NON CONSENTITO  
str = "Pippo"; // NON CONSENTITO
```

2. I confronti tra vettori/stringhe non danno i risultati attesi!

```
char s1[10], s2[10];  
if(s1 == s2) { // QUESTO TEST E' SEMPRE FALSO  
    ...  
}
```

Per svolgere queste operazioni è necessario operare *elemento per elemento* o *carattere per carattere*

## I/O Formattato Avanzato

### I/O formattato avanzato

Le direttive della stringa formato di `printf` e `scanf` sono in realtà più complesse:

- `printf: %[flag][min dim][.precisione][dimensione]<carattere>`
  - [flag]: più usati
    - giustificazione della stampa a sinistra
    - + premette sempre il segno
  - [min dim]: dimensione minima di stampa in caratteri
  - [precisione]: numero di cifre frazionarie per numeri reali
  - [dimensione]: uno tra
    - h argomento è short
    - l argomento è long
    - L argomento è long double

- `scanf: %[*][min dim][dimensione]<carattere>`
  - `[*]`: non fa effettuare l'assegnazione (ad es., per "saltare" un dato in input)
  - `[min dim]`: dimensione massima del campo da leggere
  - `[dimensione]`: uno tra
    - h argomento è short
    - l argomento è long
    - L argomento è long double

## I/O formattato su stringhe

Esistono due versioni aggiuntive di `printf` e `scanf` che operano su stringhe anziché sullo standard input/standard output:

```
sprintf(stringa, formato, arg1, arg2, ....., argN)
```

```
sscanf(stringa, formato, arg1, arg2, ....., argN)
```

- `formato` e `arg1, arg2, ....., argN` sono come in `printf` e `scanf`
- i dati vengono letti/scritti su `stringa` invece che da tastiera o su schermo
- sono definite in `stdio.h`

## I/O a caratteri

### I/O a caratteri

- Acquisizione/stampa di un carattere alla volta
- Istruzioni:
  - `getchar()`
    - \* Legge un carattere dalla coda di input
    - \* In caso di terminazione della coda (con CTRL-d) il risultato è la costante `EOF` (definita in `stdio.h`)
  - `putchar(<carattere>)`
    - \* Stampa `<carattere>` su schermo
    - \* `<carattere>` è un dato di tipo `char`

### **Attenzione!**

`getchar` restituisce un valore `int` e non `char` in quanto deve restituire o il codice ASCII (numero tra 0 e 255) oppure `EOF` (non è un carattere ASCII)

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

```
int main(int argc, char*argv[]) {
    int tasto;
    printf("Premere un tasto...: ");
    tasto = getchar();
}
```

```

if (tasto != EOF) {
    printf("Carattere letto (putchar): ");
    putchar(tasto);
    putchar('\n');
    printf("Carattere letto (printf): %c\n", tast);
    printf("Codice ASCII: %d\n",tasto);
} else {
    printf("Letto end of file\n"); //è stato premuto CTRL-d
}
return 0;
}

```

```

prava@mas:~/teaching/LabS0/examples$ ./getchar_putchar.x
Premere un tasto...: G
Carattere letto (putchar): G
Carattere letto (printf): G
Codice ASCII: 71
prava@mas:~/teaching/LabS0/examples$ █

```

## getchar/putchar vs scanf/printf

- scanf e printf sono costruite a partire da getchar e putchar
- scanf/printf utili quando è noto il formato (tipo) del dato che viene letto  
*Esempio:* serie di dati tabulati con formato fisso
- getchar/putchar utili quando il formato del dato da leggere non è noto  
*Esempio:* un testo

## Esempio: riscriviamo il comando wc

```

#include <stdio.h>

#define IN 0 // all'interno di una parola
#define OUT 1 // all'esterno di una parola

// conta linee, parole e caratteri dell'input.
main()
{
    int c, linee, parole, caratteri, stato;

    stato = OUT;
    linee = parole = caratteri = 0;
    c = getchar();
    while(c != EOF) {
        caratteri++;
        if(c == '\n') {
            linee++;
        }
        if(c == ' ' || c == '\t' || c == '\n') {
            stato = OUT;
        } else if(stato == OUT) {
            stato = IN;
        }
    }
}

```

```

        parole++;
    }
    c = getchar();
}
printf("%8d %8d %8d\n", linee, parole, caratteri);
}

```

## I/O per righe

### I/O per righe

- Acquisizione/stampa di una riga alla volta
  - Riga = serie di caratteri terminata da '\n'
- Istruzioni:
  - gets(<variabile stringa>)
    - \* Legge una riga da tastiera (fino a fine riga o EOF)
    - \* La riga viene salvata come stringa in <variabile stringa> senza il carattere '\n' alla fine
    - \* In caso di errore il risultato è la costante NULL (definita in stdio.h)
  - puts(<stringa>)
    - \* Stampa <stringa> su schermo
    - \* Aggiunge sempre '\n' in coda alla stringa

Esempio:

```

#include <stdio.h>

main()
{
    char s[10];          /* NON char* s; */
    char *res;

    printf("Scrivi qualcosa\n");
    res = gets(s);
    if (res != NULL)    /* errore ? */
    {
        puts("Hai inserito: ");
        puts(s);
    }
}

```

**NOTE:**

- puts/gets sono costruite a partire da getchar/putchar
- Usate meno di frequente degli altre istruzioni di I/O
- puts(s) è identica a printf("%s\n", s);
- Uso di gets richiede l'allocazione dello spazio di memoria per la riga letta in input

**ATTENZIONE!**

se la riga in input contiene più caratteri di quelli allocati il programma termina con errore!