Lezione 6: Funzioni di I/O avanzate

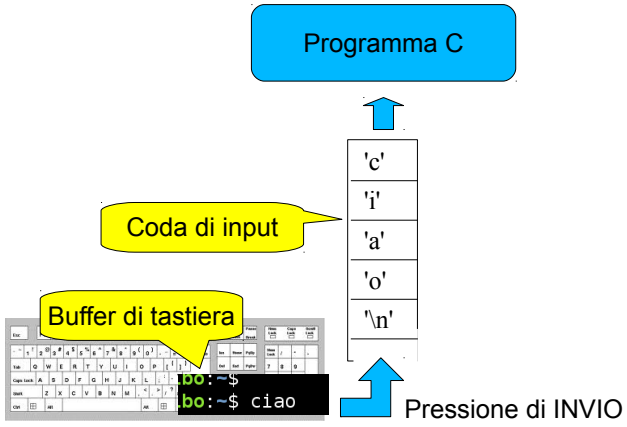
Laboratorio di Elementi di Architettura e Sistemi Operativi

11 Aprile 2012

Funzioni avanzate di I/O

Il buffer di tastiera

- I caratteri digitati da tastiera rimangono in un buffer di tastiera finché non viene premuto INVIO
- Possono essere cancellati con BACKSPACE e riscritti
- Alla pressione di INVIO vengono inseriti nella coda di input; un carattere in fila a quello succesivo compreso il carattere NEWLINE
- Le funzioni di lettura attingono i caratteri dalla coda di input:
  - Se la coda è vuota le funzioni di lettura bloccano il programma in attesa che vengano premuti dei tasti + INVIO
  - I caratteri non letti rimangono in coda in attesa di essere consumati dalle successive chiamate a funzioni di lettura
- La coda di input può essere chiusa con la pressione di CTRL-d
  - nessun nuovo carattere arriverà dal buffer di tastiera per il programma C
  - i caratteri ancora in coda possono essere consumati dalle funzioni di lettura dopodiché queste NON si bloccano in attesa di nuovi caratteri ma restituiscono un msg di errore



La funzione `scanf`

- `scanf` legge un carattere alla volta dalla coda di input finché tali caratteri corrispondono alla stringa di formato
- Alla prima violazione esce e i caratteri non letti rimangono nella coda di input: *verranno letti dalle funzioni di input successive*
- Nella stringa di formato i caratteri space, tab, newline sono equivalenti: Definiamolo il generico carattere "spazio"
- La presenza di un carattere "spazio" nella stringa di formato equivale a riconoscere la presenza di "uno o più caratteri spazio"

Un esempio: la calcolatrice

Modifichiamo l'esercizio della calcolatrice per chiedere all'utente se vuole fare un'altra operazione:

```
#include <stdio.h>

main()
{
  int a, b;
  char op = 's';
  while(op == 's' || op == 'S') {
    printf("Inserire l'operazione da eseguire: ");
    scanf("%d %c %d", &a, &op, &b);
    switch(op) {
        case '+':
            printf("%d + %d = %d\n", a, b, a + b);
            break;
        ....
        default:
            printf("Operazione sconosciuta!\n");
            break;
    }
    printf("Eseguire una nuova operazione (s/n)? ");
    scanf("%c", &op);
  }
}
```

- Il codice *non funziona correttamente*: il programma termina subito dopo la prima operazione!
- Questo perché il carattere di a capo alla fine dell'operazione rimane nel buffer di input, e viene letto dall'istruzione `scanf("%c", &op);`:
  - il valore di `c` diventa `'\n'`, che è diverso da `s` e `S`
  - il controllo del while fallisce ed il programma termina

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

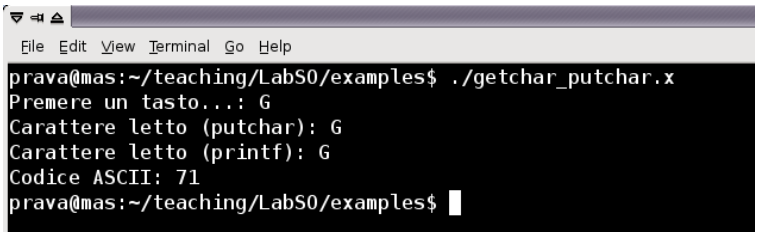
I/O a caratteri

- 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", tasto);
    printf("Codice ASCII: %d\n",tasto);
  } else {
    printf("Letto end of file\n"); //è stato premuto CTRL-d
  }
  return 0;
}
```



`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);
}
```

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

- Per effettuare qualsiasi operazione su stringhe è necessario utilizzare le funzioni della libreria `string.h`:

funzione	*definizione*
`char* strcat (char* s1, char* s2);`	**concatenazione di s1 e s2**
`char* strchr (char* s, int c);`	**trova c dentro s**
`int strcmp (char* s1, char* s2);`	**confronto**
`char* strcpy (char* s1, char* s2);`	**copia s2 in s1**
`int strlen (char* s);`	**lunghezza di s**
`char* strncat (char* s1,char* s2,int n);`	**concat. n car. max**
`char* strncpy (char* s1,char* s2,int n);`	**copia n car. max**
`int strncmp(char* dest,char* src,int n);`	**cfr. n car. max**

- `char *` rappresenta l'indirizzo di memoria del primo elemento di una stringa (*puntatore*)
  - usato nelle funzioni per rappresentare stringhe di lunghezza ignota.

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 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!