Università degli Studi di Milano

Corso di Laurea in
Sicurezza dei Sistemi e delle Reti Informatiche

Lezione 17 – Input/Output formattato

FABIO SCOTTI

Laboratorio di programmazione per la sicurezza

Indice

1. Input/Output formattato

Nelle lezioni precedenti sono stati presentati esempi di come in C sia possibile formattare l'output dei programmi tramite la funzione printf su terminale e tramite la funzione fprintf su file. In questa lezione approfondiamo l'aspetto che riguarda l'input. Il nostro obiettivo è quello di capire come sia possibile estrarre correttamente le informazioni da un ingresso del nostro programma.

Le tecniche che esporremo funzionano ugualmente sia che l'input venga dalla tastiera, da un file o da un socket ovvero da uno stream di dati proveniente dalla rete. Questo può avvenire grazie al particolare modo di gestire l'input/output che tratta tutti questi tipi di output secondo un unico modello chiamato "flusso di dati" o "data stream".

1.1 Semplici esempi di input/output formattato

Gli esempi che sono già stati presentati nel corso riguardavano la lettura da tastiera dei dati immessi da un utente. Ad esempio queste due funzioni

```
scanf("%f" , &f) ;
scanf("%d" , &d) ;
```

permettono di leggere i caratteri immessi da tastiera dall'utente come un float e i successivi come un numero intero, memorizzandoli correttamente in memoria.

Altri casi di input/output formattato visti durante il corso riguardavano la scrittura e lettura di campi (record) su file di testo. Ad esempio se un nostro programma andava a scrivere nome, cognome e telefono su un file separati da tabulatori con una chiamata del tipo

```
fprintf(Fp1, "%s\t%s\t%s\n", nome, cognome, tel);
```

abbiamo visto come sia possibile successivamente aprire quel file e recuperare le informazioni contenute in ogni riga con una chiamata del tipo

```
while ( fscanf(Fp1, "%s\t%s\t%s\n", nome, cognome, tel) == 3 )
```

1.2 Esempi avanzati di input/output formattato

Per completare l'insieme degli strumenti che servono per il nostro corso per la gestione dei flussi di dati (tastiera, monitor, file, ecc.) esaminiamo alcune altre importanti possibilità che ci offre l'input/output formattato (printf-scanf e fprintf-fscanf). Mediante esempi di codice vogliamo conoscere i seguenti aspetti:

- Costruzione della stringa di controllo
- Gruppo di scansione
- Gruppo di scansione invertito
- Leggere ed ignorare alcuni caratteri dallo stream di input

1.3 Familiarizzare con il concetto di stream

E' un errore pensare che una chiamata di una scanf prenda tutti i caratteri battuti dall'utente sulla tastiera fino alla pressione dell'enter e li memorizzi all'indirizzo specificato. Questo accade solo in alcuni casi. E' buona cosa immaginare che ogni dispositivo di

ingresso rispetto al programma C abbia un propria memoria tampone (buffer). Il dispositivo riempie il buffer di caratteri (per esempio la tastiera riempie il buffer di tastiera) e le nostre chiamate alla scanf o fscanf lo svuotano prendendo i caratteri che servono al programma.

Per esempio, cosa accade se l'utente immette da tastiera più caratteri dei quelli che la scanf legge e memorizza? La risposta è semplice: rimangono nel buffer di tastiera. Essi rimarranno nel buffer di tastiera fino a quando vi sarà un'altra chiamata di una scanf che li andrà a "prelevare". Chiariamo meglio questo concetto con il seguente esempio.

Immaginiamo che l'utente immetta la stringa "sicurezza".

```
#include <stdio.h>

int main()
{
        char x, y[9];
        printf("Immetti una stringa :  ");
        scanf("%c%s",  &x , y);

        printf("Hai immesso  \n" );
        printf("Il carattere :%c    \n" ,  x );
        printf("e la stringa :%s    \n" ,  y );

        fflush(stdin);
        getchar();
        exit(0);

}
```

L'uscita del programma a terminale sarà:

```
Immetti una stringa :  sicurezza
Hai immesso
Il carattere :s
e la stringa :icurezza
```

L'esempio mostra chiaramente come il flusso di dati in ingresso (in questo caso i caratteri battuti da tastiera 's' 'i' 'c' 'u' 'r' 'e' 'z' 'z' 'a' 'enter') possa essere spezzato e finire in celle di memoria diverse (il primo carattere viene letto, tolto dal buffer di tastiera e memorizzato nella variabile x e la parte rimanente della stringa viene letta, tolta dal buffer e memorizzata nella variabile y).

La dimostrazione che i caratteri che non sono letti dalle scanf rimangono nel buffer di tastiera si ottiene controllando che il seguente codice produce lo stesso output del precedente programma

```
#include <stdio.h>

int main()
{
        char x, y[9];
        printf("Immetti una stringa :  ");
        scanf("%c",  &x );
        scanf("%s",   y );
        printf("Hai immesso  \n" );
        printf("Il carattere :%c    \n" ,  x );
```

```
        printf("e la stringa :%s    \n" ,  y );

        fflush(stdin);
        getchar();
        exit(0);
}
```

Infatti vediamo ancora:

```
Immetti una stringa :  sicurezza
Hai immesso
Il carattere :s
e la stringa :icurezza
```

1.4 Un errore molto frequente

Per evitare errori di difficile correzione è buona norma usare la funzione fflush(stdin) che pulisce il buffer da tastiera dopo aver usato una scanf per leggere da terminale. Dimenticandosi di fare ciò, se per errore leggiamo un numero di caratteri mediante scanf diverso da quello che l'utente ha immesso, i caratteri rimanenti rimangono nel buffer e potrebbero essere letti da altre scanf del nostro programma. Il caso più frequente accade quando l'utente immette una stringa con degli spazi.

Prendiamo come esempio un programma nel quale vogliamo chiedere all'utente di immettere il cognome ed il CAP di residenza. La soluzione apparentemente corretta potrebbe sembrare

```
#include <stdio.h>

int main()
{
        char c[20], CAP[20];
        printf("Immetti cognome:");
        scanf("%s",  c   );
        printf("Immetti CAP:");
        scanf("%s",  CAP );
        printf("Hai immesso  \n" );
        printf("Il cognome  :%s    \n" ,  c   );
        printf("e il CAP    :%s    \n" ,  CAP );

        fflush(stdin);
        getchar();
        exit(0);

}
```

Infatti immettendo il cognome "Rossi" ed il CAP "23100" si ottiene

```
Immetti cognome:Rossi
Immetti CAP:23100
Hai immesso
Il cognome  :Rossi
e il CAP    :23100
```

Diverso è il caso se l'utente immette "Della valle" come cognome:

```
Immetti cognome:Della valle
Immetti CAP:Hai immesso
Il cognome  :Della
e il CAP    :valle
```

E' facile immaginare che a causa dello spazio solo la prima parte della stringa "Della" è finita dove doveva e che la seconda parte "valle" è finita addirittura nel CAP!

Per questo motivo è necessario usare la funzione fflush(stdin) dopo una scanf per pulire il buffer di tastiera da ogni carattere rimasto "non voluto" prima di procedere con il programma. In questo modo, anche se vi fossero degli errori, essi rimarrebbero localizzati nella variabile scritta dalla scanf e non si ripercuoterebbero in tutte le altre variabili lette dalle scanf successive.

La gestione dei nomi con gli spazi può essere fatta in moltissimi altri modi (ad esempio leggendo più stringhe per il cognome e verificando poi se sono vuote, leggendolo carattere per carattere e sostituendo gli spazi con un altro carattere come '_', ecc.)

1.5 Gruppo di scansione e gruppo di scansione invertito

E' possibile prelevare dal flusso di dati (tastiera, file, ecc.) solo alcuni caratteri che appartengono ad un gruppo che specifichiamo. Gli altri caratteri non appartenenti al gruppo che vi erano nel buffer vengono scartati e non memorizzati.

1.5.1 Gruppo di scansione

Il gruppo di scansione si definisce nella stringa di controllo della scanf impiegando un carattere '%' seguito da il gruppo di caratteri chiuso da parentesi quadre es: %[aeiou].

Esaminiamo un esempio:

```
#include <stdio.h>

int main()
{
        char z[12];

        printf("Immetti una stringa :  ");
        scanf("%[sicurez]",  z );

        printf("abbiamo in memoria :%s    \n" ,  z );

        fflush(stdin);
        getchar();
        exit(0);

}
```

Lanciamo il programma per tre volte immettendo stringhe diverse e guardando cosa memorizziamo usando il gruppo [sicurez].

```
Immetti una stringa :  sicurezza
abbiamo in memoria :sicurezz

Immetti una stringa :  ssssiiiiiccccccuuuurrrrreeeezzzzzaa
abbiamo in memoria :ssssiiiiiccccccuuuurrrrreeeezzzzz

Immetti una stringa :  inizio
abbiamo in memoria :i
```

Attenzione: notiamo che nell'ultimo caso abbiamo memorizzato solo il carattere 'i' poiché il carattere successivo 'n' non appartiene al gruppo di scansione che abbiamo specificato [sicurez]. La memorizzazione dei caratteri si è quindi arrestata, anche se successivamente nella parola "inizio" troviamo il carattere 'z' che, effettivamente, appartiene al gruppo di scansione.

Per questo motivo possiamo immaginare il gruppo di scansione come un setaccio che fa passare solo i caratteri appartenenti al gruppo e che arresta la scansione NON APPENA incontra un carattere non appartenente al gruppo.

1.5.2 Gruppo di scansione invertito

Il gruppo di scansione ***invertito*** si definisce nella stringa di controllo della scanf impiegando un carattere '%' seguito il gruppo di caratteri chiuso da parentesi quadre con davanti il carattere '^' es: %[^aeiou].

Il gruppo di scansione invertito funziona al contrario. Ad esempio partendo dal programma mostrato si potrebbero provare ad estrarre solo le vocali dalla stringa e fermarsi alla prima consonante impiegando il gruppo di scansione [aeiou]. Usando un gruppo di scansione invertito [^aeiou] è invece possibile estrarre solo le consonanti dalla stringa e fermarsi alla prima vocale.

1.6 Ignorare i caratteri dallo stream in input

E' possibile estrarre solo le informazioni che servono da una stream che contiene più dati di quelli utili. Esaminiamo esempio seguente.

```
#include <stdio.h>

int main()
{
        int m1, g1, anno1, m2, g2, anno2 ;

        printf("Immetti una data nella forma mm-dd-yy:  \n");
        scanf("%d%*c%d%*c%d",  &m1, &g1, &anno1 );
        printf("giorno %d, mese %d, anno %d \n",  g1, m1, anno1 );


        printf("Immetti una data nella forma mm/dd/yy:  \n");
        scanf("%d%*c%d%*c%d",  &m2, &g2, &anno2 );  // codice uguale  !!
        printf("giorno %d, mese %d, anno %d \n",  g2, m2, anno2 );


        fflush(stdin);
        getchar();
```

```
        exit(0);
}
```

Il nostro obiettivo è quello di memorizzare solo i numeri del mese, giorno e anno immessi dall'utente e non i caratteri separatori '-'. Per imporre che la scanf salti un carattere, è sufficiente indicare nella stringa di controllo "*c". Infatti otteniamo in esecuzione

```
Immetti una data nella forma mm-dd-yy:
12-13-1998
giorno 13, mese 12, anno 1998
Immetti una data nella forma mm/dd/yy:
12-14-04
giorno 14, mese 12, anno 4
```

2. Scansione di un file di log

Uniamo i concetti visti precedentemente in un caso pratico: la scansione di file di log di una server Apache.

2.1 Composizione di un file di log

Di solito i file di log sono composti da un insieme di campi come i seguenti:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"
%P %T" debug

%...a:          Remote IP-address
%...A:          Local IP-address
%...B:          Bytes sent, excluding HTTP headers.
%...b:          Bytes sent, excluding HTTP headers. In CLF format i.e. a '-'
rather than a 0 when no bytes are sent.
%...c:          Connection status when response was completed.
                 'X' = connection aborted before the response completed.
                 '+' = connection may be kept alive after the response is
sent.
                 '-' = connection will be closed after the response is sent.
%...{FOOBAR}e:  The contents of the environment variable FOOBAR
%...f:          Filename
%...h:          Remote host
%...H       The request protocol
%...{Foobar}i:  The contents of Foobar: header line(s) in the request sent
to the server.
%...l:          Remote logname (from identd, if supplied)
%...m       The request method
%...{Foobar}n:  The contents of note "Foobar" from another module.
%...{Foobar}o:  The contents of Foobar: header line(s) in the reply.
%...p:          The canonical Port of the server serving the request
%...P:          The process ID of the child that serviced the request.
%...q       The query string (prepended with a ? if a query string exists,
otherwise an empty string)
%...r:          First line of request
%...s:          Status.  For requests that got internally redirected, this
is the status of the *original* request --- %...>s for the last.
```

```
%...t:          Time, in common log format time format (standard english
format)
%...{format}t:  The time, in the form given by format, which should be in
strftime(3) format. (potentially localized)
%...T:          The time taken to serve the request, in seconds.
%...u:          Remote user (from auth; may be bogus if return status (%s)
is 401)
%...U:          The URL path requested, not including any query string.
%...v:          The canonical ServerName of the server serving the request.
%...V:          The server name according to the UseCanonicalName setting.
```

Nota bene: il file da cui gli esempi di questa dispensa sono stati tratti e' stato ridotto, anonimizzato rispetto agli utenti. Anche i numeri di IP sono di fantasia.

Esaminiamo la seguente riga:

```
159.149.67.22 - - [02/Apr/2004:10:27:10 +0200] "GET / HTTP/1.1" 302 5 "-"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.4) Gecko/20030624" 0
www.dti.unimi.it
```

Mostriamo meglio i campi andando a capo:

```
151.42.178.116 -
   -
  [02/Apr/2004:10:27:30 +0200]
  "GET
  /insegnamenti.php?z=0;id_corso=8
  HTTP/1.1"
  302
  5
  "-
  " "
  Mozilla/5.0 (X11; U; Linux i686; it-IT; rv:1.6) Gecko/20040207
Firefox/0.8"
  0
  www.dti.unimi.it
```

Quello che emerge è che un utente con IP 151.42.178.116 collegandosi con un browser Firefox ha richiesto (get) la pagina /insegnamenti.php?z=0;id_corso=8 dal server che risponde a www.dti.unimi.it

2.2 Programma per la scansione di un file di log

Avendo a disposizione un file composto da righe di log di questo tipo (attenzione, controllare dove sono i veri a capo del file, non quelli della dispensa)

```
159.149.67.22 - - [02/Apr/2004:10:27:10 +0200] "GET / HTTP/1.1" 302 5 "-"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.4) Gecko/20030624" 0
www.dti.unimi.it
159.149.67.22 - - [02/Apr/2004:10:27:18 +0200] "GET / HTTP/1.1" 302 5 "-"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.4) Gecko/20030624" 0
www.dti.unimi.it
159.149.67.22 - - [02/Apr/2004:10:27:30 +0200] "GET / HTTP/1.1" 302 5 "-"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.4) Gecko/20030624" 0
www.dti.unimi.it
ecc..
```

scriviamo un programma che legge dal file gli *IP degli utenti connessi* e *l'oggetto che hanno richiesto al server* stampandoli con delle printf.

Una possibile soluzione potrebbe essere la seguente:

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


int main()
{
   int base;
   char c;

   int ip1, ip2, ip3, ip4 ;
   char data[256];
   char operation[256];
   char page[256];
   char nomefile[]="t2.log";
   FILE * Fp1;

   // Apro il file in modalita' write il file di testo
   Fp1 = fopen(nomefile, "r");
   if (Fp1==NULL){
        printf("File %s not found\n", nomefile);
        exit(-1);
   }

   while ( fscanf(Fp1, "%d.", &ip1  ) > 0 )
   {

        // approccio non molto raffinato ... esiste il tipo
        // di dato per rappresentare correttamente un IP in C

        fscanf(Fp1, "%d.%d.%d",  &ip2, &ip3, &ip4 );
        fscanf(Fp1, " - - [%s +0200] %s", data , operation);
        fscanf(Fp1, " %s HTTP/1.1", page);

        printf(  "%d.%d.%d.%d \n", ip1, ip2, ip3, ip4 );
        printf(  "  data:      %s  \n", data  );
        printf(  "  operation: %s  \n", operation  );
        printf(  "  page:      %s  \n", page  );

        //  vado fino in fondo alla riga cosi'.... ma si puo' migliorare
        while ( fscanf(Fp1, "%c", &c ) > 0 )
        {
               if (c ==  '\n' )  break;
        }
        printf( " \n\n\n" );
   }

   getchar();
   fclose(Fp1);
   exit(0);
}  // main
```

Nella soluzione apriamo il file ed iniziamo a leggere il primo elemento che dobbiamo incontrare. Nel nostro file di log abbiamo un IP, ad esempio il primo: `159.149.67.22`. Pertanto il codice potrebbe essere il seguente.

```
while ( fscanf(Fp1, "%d.", &ip1  ) > 0 )
{
```

…

Nel linguaggio C esiste il tipo di dato astratto per leggere e gestire un IP. Per semplicità non lo introduciamo ora ed andiamo a leggere i suoi 4 interi che lo compongono usando i punti come separatori. Il codice potrebbe essere il seguente.

```
  fscanf(Fp1, "%d.%d.%d",  &ip2, &ip3, &ip4 );
```

Nelle righe successive usiamo il concetto di maschera per andare a leggere nella porzione di log del tipo

```
 – – [02/Apr/2004:10:27:10 +0200] "GET
```

solo quello che ci interessa, ovvero la data ed il tipo di operazione. Il codice potrebbe essere il seguente.

```
  fscanf(Fp1, " – – [%s +0200] %s", data , operation);
```

Notare che tutti i caratteri che appartenevano alla stringa di controllo vengono tolti dal buffer di lettura, non solo `data` ed `operation`. In questo modo il prossimo carattere che verrà letto sarà il primo dopo il "`GET`"

Successivamente, troviamo la pagina richiesta ed il protocollo, ad esempio / HTTP/1.1", ovvero la home "/" ed il protocollo "HTTP/1.1". Possiamo memorizzare questi dati con la seguente chiamata:

```
  fscanf(Fp1, " %s HTTP/1.1", page);
```

Ora abbiamo tutto quello che serve e possiamo stampare l'IP dell'utente e l'oggetto richiesto al server con le seguenti chiamate:

```
  printf(  "%d.%d.%d.%d \n", ip1, ip2, ip3, ip4 );
  printf(  "  data:      %s  \n", data  );
  printf(  "  operation: %s  \n", operation  );
  printf(  "  page:      %s  \n", page  );
```

Attenzione: è necessario portare la "testina di lettura" alla fine della riga. Così facendo, il nostro ciclo while può correttamente ritrovare l'IP del prossimo utente sulla prossima riga. Questo può essere fatto in molti modi. Uno dei modi più semplici consiste nel continuare a leggere carattere per carattere la riga fino a trovare il carattere di fine riga '\n', ovvero il seguente codice:

```
while ( fscanf(Fp1, "%c", &c ) > 0 )
  {
     if (c ==  '\n' )  break;
  }
```

Una volta lanciato il nostro programma stampa a monitor dati simili a questi:

```
84.11.41.177
  data:      02/Apr/2004:10:40:23
  operation: "GET
  page:      /~liberali/anybrowser3.jpg
```