Lezione 17 e 18

- Esempio di scrittura con il C di un file html
- Input/Output formattato
- Esempio di file formattato

Fabio Scotti (2004-2009)

Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)

Laboratorio di programmazione

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 17 e 18

Scrittura di un file html

Obiettivo:

- Creare un file HTML in C

Creare file HTML con il C

- **Creare un file HTML in C significa creare un normale file di testo contenente gli elementi necessari perché possa essere un file HTML**
- **Limitando il file di testo da creare proprio all'essenziale troveremo almeno questo codice:**

```
<HTML>
<HEAD>
</HEAD>
<BODY>
</BODY>
</HTML>
```

3

Estensione del file

- **La corretta estensione del file di testo che il programma creerà dovrà essere**
  - .html oppure
  - .htm
- **Questo affinché un browser possa aprirlo e visualizzarlo interpretando i tag che vi sono all'interno**

4

Etichette (1)

- **L'idea più semplice per scrivere del codice C che crea dei file HTML consiste**
  - nel creare delle etichette nel nostro programma
  - contenenti le parti di testo HTML che devono essere presenti
  - e poi stamparle su file con delle **`fprintf`**

5

Etichette (2)

```
<html>
<head>
<title> Creato da un mio programma C! </title>
</head>
<body bgcolor="#FFFFFF">
creato da un mio programma C,<br />
<b>Magnifico</b><br />
scrivo altre cose ……. chiudo il body ed il tag html
</body>
</html>
```

```
#define MIO_TITOLO "Creato da un mio programma C!"

// ATTENZIONE: Per spezzare il define su più di una riga dobbiamo
// utilizzare il carattere \
// Perché spezzare su più righe la costante stringa?
// Solo perché è molto più facile da leggere!
#define HTML_TOP "<html>\n<head>\n<title>" \
    MIO_TITOLO \
    "</title>\n</head>\n"
#define BODY_START "<body bgcolor=\"#FFFFFF\">\n"
#define HTML_BOTTOM "</body>\n</html>"
```

6

Pagine web create in C (1)

- **L'uso del C diviene più interessante quando occorre ripetere molte volte una porzione di codice HTML**
  - come avviene nel caso di tabelle con molte righe
  - in questo caso la parte di tabella che viene ripetuta può essere stampata da un ciclo

7

Pagine web create in C (2)

- **Procedimento utilizzato dai server che creano pagine attive su richiesta dei browser:**
  - il numero ed il tipo di elementi che comporranno la pagina HTML finale che verrà spedita al browser dell'utente dipende da condizioni e dati spediti al server dal browser.
- **Esempio:**
  - un utente attraverso il browser clicca sul link di una pagina degli orari ferroviari delle prossime 2 ore
  - il server accede ad un database, crea una tabella con tutti i treni delle prossime 2 ore e manda il file HTML così creato al browser che lo avevo richiesto.

8

Tabelle (1)

- **Come potrebbe essere composto il codice HTML di una tabella con 100 righe contenenti un numero ed un nome?**

-- 1--	Tom
-- 2 --	Neil
…	…
-- 100 --	Tom

Tabella visualizzata dal browser

```
<table border=1 width=75%>
<tr>
<td>-- 1 --</td>
<td>Tom</td>
</tr>

<tr>
<td>-- 2 --</td>
<td>Neil</td>
</tr>
…
<tr>
<td>-- 100 --</td>
<td>Tom</td>
</tr>
</table>
```

Codice html

9

Tabelle (2)

- **Le parti costanti sono**

```
<table border=1 width=75%>
<tr>
<td>-- 1 --</td>
<td>Tom</td>
</tr>

<tr>
<td>-- 2 --</td>
<td>Neil</td>
</tr>
…
<tr>
<td>-- 100 --</td>
<td>Tom</td>
</tr>
</table>
```

10

Tabelle (3)

- **La parte che deve essere ripetuta 100 volte è**

```
<table border=1 width=75%>
<tr>
<td>-- Numero --</td>
<td> Nome </td>
</tr>

<tr>
<td>-- 2 --</td>
<td>Neil</td>
</tr>
…
<tr>
<td>-- 100 --</td>
<td>Tom</td>
</tr>
</table>
```

11

Codice C per la generazione di Tabelle

- **Il codice C che genera automaticamente la tabella di 100 righe e 2 colonne potrebbe essere simile al seguente:**

```
fprintf(Fp1, "%s\n", "<table border=1 width=20%>");
for (i=0; i< 100; i++)
{
   fprintf(Fp1, "<tr>\n");
   fprintf(Fp1, "<td>-- %d --</td>\n", i );
   fprintf(Fp1, "<td>-- %d --</td>\n", Nomi[i] );
   fprintf(Fp1, "</tr>\n\n");
}
fprintf(Fp1, "%s\n", "</table>\n");
```

Codice C che genera una pagina HTML contenente una tabella

12

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 17 e 18

Input/Output formattato

Obiettivo:

- Comprendere il significato della programmazione strutturata ed il suo utilizzo per creare programmi riusabili, facilmente comprensibili e correggibili.

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
  - la funzione **`fprintf()`** su file
- **In questa lezione approfondiamo l'aspetto che riguarda l'input**
- **L'obiettivo è quello di capire come sia possibile estrarre correttamente le informazioni da un ingresso del programma**

Input formattato

- **Le tecniche che esporremo funzionano ugualmente sia che l'input venga:**
  - dalla tastiera
  - da un file
  - da un socket
  - 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"**

Semplici esempi di i/o formattato

- **Lettura da tastiera dei dati immessi da un utente:**
  - `scanf("%f", &f) ;`
  - `scanf("%d", &d) ;`
- **Scrittura di campi (record) su file di testo:**
  - `fprintf(Fp1,"%s\t%s\t%s\n",nome,cognome,tel);`
- **Recuperiamo le informazioni contenute in ogni riga con una chiamata del tipo:**

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

Esempi avanzati di i/o formattato

- **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

17

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' più corretto immaginare che**
  - ogni dispositivo di ingresso rispetto al programma C abbia un propria memoria tampone (buffer)
  - Il dispositivo riempie il buffer di caratteri
  - le chiamate alla scanf o fscanf lo svuotano prendendo i caratteri che servono al programma.

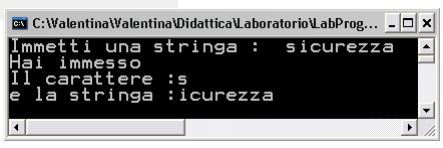
18

Un ingresso ed una uscita per ogni blocco

- **Per esempio, cosa accade se l'utente immette da tastiera più caratteri dei quelli che la scanf legge e memorizza?**
  - rimangono 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"

Esempio (1)

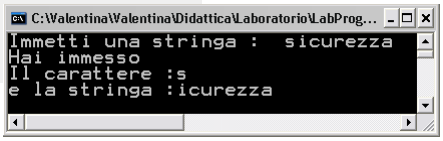
```
#include <stdio.h>

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);

}
```



Esempio (2)

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

```
#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);

}
```



21

Un errore molto frequente

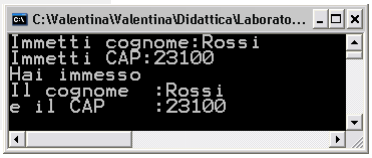
- **Per evitare errori di difficile correzione è buona norma:**
  - usare la funzione **`fflush(stdin)`** che pulisce il buffer da tastiera dopo aver usato una **`scanf`** per leggere da terminale
- **Dimenticandosi di fare ciò**
  - se per errore leggiamo un numero di caratteri mediante **`scanf`** 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**

22

Esempio (1)

- **Vogliamo chiedere all'utente di immettere il cognome ed il CAP di residenza. La soluzione apparentemente corretta potrebbe sembrare:**

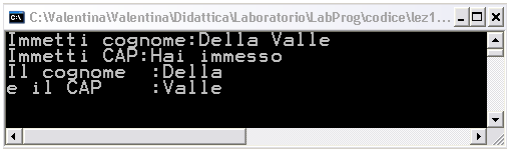
```
#include <stdio.h>

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);

}
```



23

Esempio (2)

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



- **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!**

24

Uso della `fflush(stdin)`

- **E' necessario usare la funzione `fflush(stdin)` dopo una `scanf` per pulire il buffer di tastiera**
- **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

25

Gestione dei nomi con spazio

- **La gestione dei nomi con gli spazi può essere fatta in moltissimi 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 '_'
  - con la `scanf("%[^\n]", stringa);` che legge tutti i caratteri dati in input da tastiera fino all'accapo (\n)

26

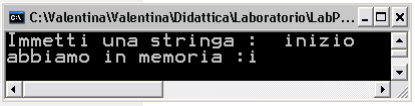
Gruppo di scansione

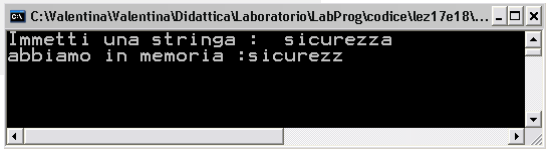
- **E' possibile prelevare dal flusso di dati (file, tastiera, ecc.)**
  - solo alcuni caratteri che appartengono ad un gruppo che specifichiamo (tra le parentesi [ ])
  - gli altri caratteri non appartenenti al gruppo che vi erano nel buffer vengono scartati e la scansione finisce
- **Il gruppo di scansione si definisce nella stringa di controllo della `scanf` impiegando**
  - un carattere '%'
  - seguito da il gruppo di caratteri chiuso da parentesi quadre
  - es: %[aeiou]

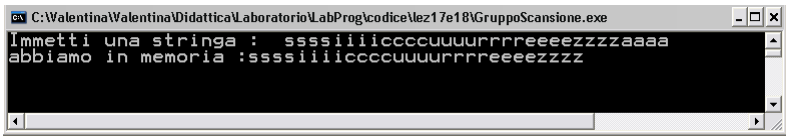
27

Esempio (1)

```
#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);

}
```







28

Esempio (2)

- **Quando abbiamo scritto la stringa "inizio":**
  - abbiamo memorizzato solo il carattere 'i'
  - poiché il carattere successivo 'n' non appartiene al gruppo di scansione [sicurez]
  - la memorizzazione dei caratteri si è quindi arrestata
  - anche se successivamente nella parola "inizio" troviamo il carattere 'z' che appartiene al gruppo di scansione
- **Il gruppo di scansione è come un setaccio**
  - che fa passare solo i caratteri appartenenti al gruppo
  - che arresta la scansione NON APPENA incontra un carattere non appartenente al gruppo

29

Gruppo di scansione invertito (1)

- **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:**
  - si estraggono i caratteri che non sono nel gruppo
  - ci si ferma quando si arriva ad un carattere nel gruppo

30

Gruppo di scansione invertito (2)

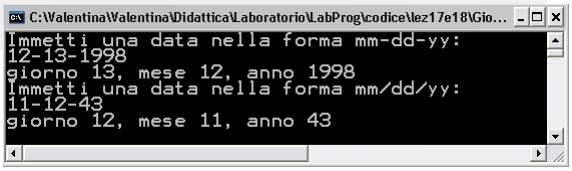
- **Ad esempio partendo dal programma mostrato:**
  - si potrebbero estrarre solo le vocali dalla stringa e fermarsi alla prima consonante impiegando il gruppo di scansione [aeiou].
  - usando un gruppo di scansione invertito [^aeiou] è invece possibile estrarre solo le consonanti dalla stringa e fermarsi alla prima vocale.

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**
- **Esempio: memorizzare solo i numeri del**
  - mese
  - giorno
  - 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" (funziona anche con altri tipi di dati: "%*d" salta un intero, "%*s" salta una stringa, ecc.)**

Esempio

```
#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);
}
```



33

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 17 e 18

Esempio di scansione di un file di log

Obiettivo:

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

Composizione di un file di log

```
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.
```


Riga di un file log (1)

- **Riga di un file log:**

  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

Riga di un file log (1)

- **Vediamo la riga nel dettaglio:**

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

Esempio

- **Avendo a disposizione un file composto da righe di log di questo tipo**

```
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, la data, il tipo di operazione e l'oggetto che hanno richiesto al server stampandoli con delle `printf`**

Commentiamo il codice (1)

- **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 )
{
```

39

Commentiamo il codice (2)

- **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 rimanenti 3 interi che lo compongono usando i punti come separatori**
- **Nel nostro esempio:**

159.149.67.22

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

40

Commentiamo il codice (3)

- **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

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

41

Commentiamo il codice (4)

- **Usando la maschera**
  - tutti i caratteri che appartenevano alla stringa di controllo vengono tolti dal buffer di lettura
  - non solo data ed operation (però saranno letti)
  - in questo modo il prossimo carattere che verrà letto sarà il primo dopo il "GET

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

```
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..
```

42

Commentiamo il codice (5)

- **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);
```


Commentiamo il codice (6)

- **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  );
```

Commentiamo il codice (7)

- **E' 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'

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

45

Codice (1)

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

46

Codice (2)

```
  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
```

47

Risultato di esecuzione

```
84.11.41.177
  data:      02/Apr/2004:10:40:18
  operation: "GET
  page:      /js/menu.php


84.11.41.177
  data:      02/Apr/2004:10:40:18
  operation: "GET
  page:      /js/cerca.php


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


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


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

Una parte della finestra di output

48