16.7 Il CGI "Sono Graditi i Vostri Commenti"

Dei tanti modi di gestire per mezzo di un CGI il FORM discusso nell'esempio, riportiamo una implementazione dovuta a Thomas Boutell, l'autore della celebre libreria grafica GD per la costruzione di grafici e del programma freeware Mapedit. Il codice è riportato nel Listato 16.4 con molti commenti.

```
/* Indicare in quale directory inserire il file dei commenti */

#define COMMENT_FILE "c:\\http\\comments.txt"

#include <stdio.h>
#include <stdlib.h>

/* Variabili globali */

/* Numero massimo di campi gestiti nel form */
#define FIELDS_MAX 100

char *names[FIELDS_MAX];
char *values[FIELDS_MAX];

int fieldsTotal = 0;

/* Controlla che la richiesta provenga davvero da un modulo */
int VerifyForm();

/* Analizza i parametri passati, riempendo gli array names[]
e values[] con informazione utile                          */
void ParseForm();

/* Libera la memoria associata con i dati del form */
void FreeForm();

/* Copia src in dst trasformando le sequenze di escape
dei caratteri speciali                                  */
void UnescapeString(char *dst, char *src);

int main(int argc, char *argv[])
{
  FILE *out;
  int i;
  int nameIndex = -1, emailIndex = -1, commentsIndex = -1;
```

```
  printf("Content-type: text/html\n\n");

  printf("<html>\n");
  printf("<head>\n");
  /* Controlla che sia un form inviato con metodo POST */
  if (!VerifyForm()) {
    printf("<title>Non è un form di tipo POST</title>\n");
    printf("</head>\n");
    printf("<h1>Non è un form di tipo POST</h1>\n");
    printf("</body></html>\n");
    return 0;
  }

  ParseForm();   /* OK, analizza il form. */

  /* Usa l'informazione */

  /* Trova l'indice di ogni campo nell'array */
  for (i = 0; (i < fieldsTotal); i++) {
    if (!strcmp(names[i], "name")) {
      nameIndex = i;
    } else if (!strcmp(names[i], "email")) {
        emailIndex = i;
    } else if (!strcmp(names[i], "comments")) {
        commentsIndex = i;
    }
  }

  /* Se manca un campo, segnalalo */
  if ((nameIndex == -1) || (emailIndex == -1) || (commentsIndex ==
-1)) {
    printf("<title></title>\n");
    printf("</head>\n");
    printf("<h1>Riempire tutti i campi</h1>\n");
    printf("Inserire nome, email e i commenti.\n");
    printf("Torna alla pagina precedente e riprova.\n");
    printf("</body></html>\n");
    return 0;
  }

/* OK, Scriviamo tutte le informazioni su un file
in modalità APPEND                                */

  out = fopen(COMMENT_FILE, "a");
  fprintf(out, "From: %s <%s>\n",
      values[nameIndex], values[emailIndex]);
  fprintf(out, "%s\n", values[commentsIndex]);

  printf("<title>Grazie, %s</title>\n", values[nameIndex]);
  printf("</head>\n");
  printf("<h1>Grazie, %s</h1>\n", values[nameIndex]);
  printf("Grazie per i Vostri Commenti.\n");
```

```
  printf("</body></html>\n");
  /* Libera la memoria usata */
  FreeForm();
  return 0;
}

int VerifyForm()
{
  char *contentType;
  char *requestMethod;
  int bad = 0;
  /* Verifica il content type dei dati ricevuti */
  contentType = getenv("CONTENT_TYPE");
  if  (strcmp(contentType,  "application/x-www-form-urlencoded")  !=
0) {
    bad = 1;
  }

  /* Controlla che si sia usato un metodo POST */
  requestMethod = getenv("REQUEST_METHOD");
  if (strcmp(requestMethod, "POST") != 0) {
    bad = 1;
  }

  return !bad;
}

/* Analizza l'informazione ricevuta e valorizza names e values[]*/

void ParseForm()
{
  char *contentLength = getenv("CONTENT_LENGTH");
  /* Numero di caratteri nei dati */
  int length;
  /* Prepara il buffer dove leggere i dati */
  char *buffer;
/* Posizione corrente nel buffer mentre si cercano i separatori */
  char *p;
  /* Determina la lunghezza dell'input */
  if (!contentLength) {
    length = 0;
  } else {
    length = atoi(contentLength);
    }
/* Alloca un buffer per memorizzare l'input. Include
uno spazio in più per semplificare l'analisi sintattica */
  buffer = (char *) malloc(length + 1);
  if (!buffer) {
    /* Uh-oh */
    return;
  }
  /* Legge tutti i dati da standard input */
```

```
  if (fread(buffer, 1, length, stdin) != length) {
    /* Uh-oh */
    return;
  }
  p = buffer;
  while (length) {

    char *name;   /* Inizio del nome corrente */
    char *value;  /* Inizio del valore corrente */

    int found;

    /* Si accerta che ci sia spazio per più campi */
    if (fieldsTotal == FIELDS_MAX) {
    /* Basta, non ne posso accettare altri */
      return;
    }

    name = p;

    /* Per prima cosa cerca il segno =. */
    found = 0;
    while (length) {
      if (*p == '=') {
       /* Termina il nome con un carattere null */
        *p = '\0';
        p++;
        found = 1;
        break;
      }
      p++;
      length--;
    }
    if (!found) {
    /* Un vuoto o una voce troncata. È strano ma potrebbe accadere
*/
      break;
    }
    value = p;
    /* Ora trova &. */
    found = 0;
    while (length) {
      if (*p == '&') {
       /* Termina il nome con un carattere null */
        *p = '\0';
        p++;
        found = 1;
        break;
      }
      p++;
      length--;
    }
```

```
    if (!found) {
      /* Suppone che sia la fine della coppia */
      *p = '\0';
    }

    names[fieldsTotal] = (char *) malloc(strlen(name) + 1);
    if (!names[fieldsTotal]) {
      /* Uh-oh, no memory.*/
      return;
    }
    values[fieldsTotal] = (char *) malloc(strlen(value) + 1);
    if (!values[fieldsTotal]) {
      /* Uh-oh, no memory. */
      free(names[fieldsTotal]);
      return;
    }

    /* Copia la stringa e risolve l'escape */
    UnescapeString(names[fieldsTotal], name);
    UnescapeString(values[fieldsTotal], value);
    fieldsTotal++;
    /* Continua con le altre coppie */
  }
  free(buffer); /* Libera il buffer. */
}

void FreeForm()
{
  int i;
  for (i=0; (i < fieldsTotal); i++) {
    free(names[i]);
    free(values[i]);
  }
}

void UnescapeString(char *dst, char *src)
{
/* Cicla sui caratteri della stringa finché non trova il null */
  while (*src) {
    char c;
    c = *src;
    /* Gestisce gli spazi rappresentati con + */
    if (c == '+') {
      c = ' ';
    } else if (c == '%') {
        /* Handle % escapes */
        char hexdigits[3];
        int ascii;
        src++;
        if (!*src) {
          /* Numeri mancanti, ignora l'escape */
          break;
```

```
        }
        hexdigits[0] = *src;
        src++;
        if (!*src) {
          /* Numeri mancanti, ignora l'escape */
          break;
        }
        hexdigits[1] = *src;
        /* Aggiunge un null finale... */
        hexdigits[2] = '\0';
        /* Usa la sscanf() per leggere il valore esadecimale */
        sscanf(hexdigits, "%x", &ascii);
        /* e lo converte nuovamente in carattere */
        c = ascii;
    }
    *dst = c;
    src++;
    dst++;
  }
  *dst = '\0';
}
```

Listato 16.4 Un CGI per gestire il FORM dei commenti

Per linee generali il programma segue il seguente flusso:

- invia (come sempre) l'intestazione della pagina di ritorno;
- verifica la correttezza del FORM che gli è stato inviato (`VerifyForm`). In questo caso specifico si assicura che sia stato scelto un metodo POST per l'invio dei parametri, altrimenti risponde con una pagina che segnala l'errore;
- analizza i parametri del FORM (`ParseForm`). In pratica con questa funzione si valorizzano due array:

```
char *names[FIELDS_MAX];
char *values[FIELDS_MAX];
```

Ogni elemento del primo array punta a un nome di parametro secondo l'ordine di trasmissione e, corrispondentemente, ogni elemento del secondo punta al valore associato a quel nome di parametro. All'interno della funzione `ParseForm` viene invocata la funzione `UnescapeString` che provvede a decodificare le eventuali sequenze di escape trasmesse nella sequenza di coppie `<nome=valore>`;

- infine, con la funzione `FreeForm` vengono liberati gli spazi di memoria allocati dinamicamente creati nel processo di acquisizione e interpretazione dei parametri.