Lesson 32

Objectives: -

- Form Echo

Form Echo

Function

This echoes the keys and fields from the HTML form which sends to it. It will accept both the POST and GET request methods.

Programming Issues

- Unencoding and parsing url-encoded input
- Accepting input from the server, and sending output back to the server

Usage

Use this program's name as the ACTION attribute in an HTML form. Place the executable in the cgi-bin directory of your web server.

formecho.c

```
/*

  FORMECHO.C
  ==========
  (c) Paul Griffiths 1999
  Email: mail@paulgriffiths.net

  Simple CGI program to echo the keys and values
  returned from an HTML form.

*/


#include <stdlib.h>
#include <stdio.h>
#include "cgihelp.h"
```

```
/*  Forward function declaratios  */

void DisplayErrorHTML();


/*  main() function  */

int main() {
    int   n = 1;
    char *key;
    char *value;

    /*  Get key/value pairs from input  */

    if ( !InitInput(1000) )
      DisplayErrorHTML();


    /*  Output HTML header  */

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

    printf("<!DOCTYPE HTML PUBLIC \"-//IETF//DTD
HTML 4.0//EN\">\n\n");
    printf("<HTML>\n\n<HEAD>\n");
    printf("    <TITLE>Formecho
Output</TITLE>\n");
    printf("</HEAD>\n\n");
    printf("<BODY>\n\n");


    /*  Output table of key/value pairs  */

    printf("<TABLE BORDER=1
CELLPADDING=10px>\n\n");
    printf("    <TR>\n");
    printf("        <TH>Key</TH>\n\n");
    printf("        <TH>Value</HT>\n\n");
    printf("    </TR>\n\n");

    while ( (key = GetIndexedKey(n)) && (value =
GetIndexedValue(n)) ) {
      printf("    <TR>\n");
      printf("        <TD>%s</TD>\n", key);
      printf("        <TD>%s</TD>\n", value);
      printf("    </TR>\n\n");
      ++n;
    }

    printf("</TABLE>\n\n");


    /*  Output HTML footer  */

    printf("</BODY>\n\n</HTML>\n\n");
```

```
    /*  Clean up and exit  */

    FreeCGI();
    return 0;
}


/*  Outputs HTML indicating error if CGI
functions fail  */

void DisplayErrorHTML() {
    printf("Content-Type: text/html\n\n");

    printf("<!DOCTYPE HTML PUBLIC \"-//IETF//DTD
HTML 4.0//EN\">\n\n");
    printf("<HTML>\n\n<HEAD>\n");
    printf("    <TITLE>Formecho
Output</TITLE>\n");
    printf("</HEAD>\n\n");
    printf("<BODY>\n\n");
    printf("<H2>Error!</H2>\n\n");
    printf("<P>An error has occurred. Please
contact the ");
    printf("site
administrator.\n\n<BR><HR><BR>\n\n");
    printf("</BODY>\n\n</HTML>\n");

    exit(EXIT_FAILURE);
}
```

---

cgihelp.h

```
/*

  CGIHELP.H
  =========
  (c) Paul Griffiths 1999
  Email: mail@paulgriffiths.net

  Interface to CGI helper functions

*/


#ifndef PG_CGIHELP
#define PG_CGIHELP
```

```
/*  Global macros  */

#define ERR_BADREQUEST          (1)
#define ERR_BADMALLOC           (2)


/*  Function declarations  */

int    InitInput      (int   maxbuffer);
char * GetValue       (char *key);
char * GetIndexedValue(int   index);
char * GetIndexedKey  (int   index);
void   FreeCGI        ();


#endif    /*  PG_CGIHELP  */
```

---

cgihelp.c

```
/*

  CGIHELP.C
  =========
  (c) Paul Griffiths 1999
  Email: mail@paulgriffiths.net

  CGI helper functions

*/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "cgihelp.h"


/*  Forward function declarations  */

void cleanurl(char *str);


/*  Linked list node to hold key/value pairs  */

typedef struct varlist {
    char *key;
    char *value;
    struct varlist *next;
```

```
} varlist;

varlist* varstart;


/*  Get all key/value pairs from input  */

int InitInput(int maxbuffer) {
    varlist *var;
    char    *input;


    /*  Allocate memory for input buffer  */

    if ( !(input = malloc(sizeof(char) *
maxbuffer)) )
      return ERR_BADMALLOC;
    memset(input, '\0', sizeof(char) *
maxbuffer);


    /*  Determine request method and fill input
buffer  */

    if ( !strcmp(getenv("REQUEST_METHOD"), "GET")
)
      strncpy(input, getenv("QUERY_STRING"),
maxbuffer - 1);
    else if ( !strcmp(getenv("REQUEST_METHOD"),
"POST") )
      fgets(input, maxbuffer, stdin);
    else
      return ERR_BADREQUEST;


    /*  Allocate memory for first key/value pair
*/

    if ( ! (varstart = malloc(sizeof(varlist)) )
)
      return ERR_BADMALLOC;
    var = varstart;


    /*  Populate key/value pairs  */

    if ( (input = strtok(input, "=")) ) {
      while ( var ) {
          if ( !(var->key = malloc(sizeof(char) *
strlen(input))) )
              return ERR_BADMALLOC;
          strcpy(var->key, input);
          cleanurl(var->key);

          input = strtok(NULL, "&");
          if ( !(var->value = malloc(sizeof(char)
* strlen(input))) )
```

```
            return ERR_BADMALLOC;
          strcpy(var->value, input);
          cleanurl(var->value);

          if ( (input = strtok(NULL, "=")) ) {
              if ( !(var->next =
malloc(sizeof(varlist))) )
                  return ERR_BADMALLOC;
          }
          else {
              var->next = NULL;
          }
          var = var->next;
      }
    }
    else {

      /*  No key/value pairs in input if we get
here  */

      free(varstart);
      varstart = NULL;
    }

    free(input);
    return 0;
}


/*  Retrieves a value from a key  */

char * GetValue(char *key) {
    varlist *var = varstart;
    while ( var ) {
      if ( !strcmp(key, var->key) )
          return var->value;
      var = var->next;
    }
    return NULL;
}


/*  Retrieves a key using a 1-based index  */

char * GetIndexedKey(int index) {
    int n;
    varlist *var = varstart;
    for ( n = 1; n < index; ++n )
      if ( var->next == NULL )
          return NULL;
      else
          var = var->next;
    return var->key;
}


/*  Retrieves a value using a 1-based index  */
```

```
char * GetIndexedValue(int index) {
    int n;
    varlist *var = varstart;
    for ( n = 1; n < index; ++n )
      if ( var->next == NULL )
          return NULL;
      else
          var = var->next;
    return var->value;
}


/*  Frees memory used by key/value pairs  */

void FreeCGI() {
    varlist *var = varstart;
    while ( var ) {
      free(var->key);
      free(var->value);
      var = var->next;
      free(varstart);
      varstart = var;
    }
}


/*  Cleans up url-encoded string  */

void cleanurl(char *str) {
    char asciinum[3] = {0};
    int i = 0, c;

    while ( str[i] ) {
      if ( str[i] == '+' )
          str[i] = ' ';
      else if ( str[i] == '%' ) {
          asciinum[0] = str[i+1];
          asciinum[1] = str[i+2];
          str[i] = strtol(asciinum, NULL, 16);
          c = i+1;
          do {
              str[c] = str[c+2];
          } while ( str[2+(c++)] );
      }
      ++i;
    }
}
```

CGI Calculator

Function

This program implements a simple, web-based CGI calculator. The user can input two numbers, and either:

- add them, or
- subtract one from the other, or
- multiply them, or
- divide one by the other, or
- raise one to the power of the other

The program does not depend on an existing HTML interface; calling the program without supplying any input will cause it to generate its own HTML interface, which can then be used to enter calculations.

This program uses the same CGI helper functions as the other programs in the CGI section of this site.

Programming Issues

The CGI programming issues involved are fairly basic:

- Getting and verifying user input
- Handling bad input cleanly
- Outputting the desired HTML

Usage

Place the program in the cgi-bin (or equivalent) directory on your web server, and call it without input when invoking for the first time. calc.c

```
/*

  CALC.C
  ======
  (c) Paul Griffiths
  Email: mail@paulgriffiths.net
```

```
  Simple CGI web-based calculator

*/


#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cgihelp.h"


/*  Global macros  */

#define ERR_NO_VALUE              (1)
#define ERR_BAD_VALUE             (2)
#define ERR_ZERO_DIVIDE           (3)
#define ERR_BAD_OPERATOR          (4)


/*  main() function  */

int main() {
    int   a, b, result, error = 0;
    char *num1, *num2, *op, *endptr = NULL;


    /*  Output HTML header  */

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

    printf("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD
HTML 4.0//EN\">\n\n");
    printf("<HTML>\n\n");

    printf("<HEAD>\n");
    printf("    <TITLE>CGI
Calculator</TITLE>\n");
    printf("</HEAD>\n\n");

    printf("<BODY>\n\n");
    printf("<H1>CGI Calculator</H1>\n\n");


    /*  Get input and process it  */

    if ( InitInput(100) ) {
      printf("<P>ERROR: Problem getting
input</P>\n\n");
    }
    else {
      printf("<P>Last result: ");

      num1 = GetValue("number1");
      num2 = GetValue("number2");
      op   = GetValue("operator");
```

```
     if ( !num1 && !num2 && !op )
         printf("No previous
calculation.</P>\n\n");
     else {

         /*  Get first number  */

         if ( !num1 )
             error = ERR_NO_VALUE;
         else {
             a = strtol(num1, &endptr, 0);
             if ( *endptr )
                 error = ERR_BAD_VALUE;
         }


         /*  Get second number  */

         if ( !num2 )
             error = ERR_NO_VALUE;
         else {
             b = strtol(num2, &endptr, 0);
             if ( *endptr )
                 error = ERR_BAD_VALUE;
         }


         /*  Get operator and calculate result
*/

         if ( !op )
             error = ERR_NO_VALUE;
         else {
             if ( !strcmp(op, "plus") )
                 result = a + b;
             else if ( !strcmp(op, "minus") )
                 result = a - b;
             else if ( !strcmp(op, "multiplied
by") )
                 result = a * b;
             else if ( !strcmp(op, "divided by")
)
                 if ( b )
                     result = a / b;
                 else
                     error = ERR_ZERO_DIVIDE;
             else if ( !strcmp(op, "to the power
of") )
                 result = pow(a, b);
             else
                 error = ERR_BAD_OPERATOR;
         }


         /*  Check for input errors and output
result  */
```

```
         switch ( error ) {
         case ERR_NO_VALUE:
            printf("Error - one or more values
are missing.</P>\n\n");
            break;

         case ERR_BAD_VALUE:
            printf("Error - one or more values
was invalid.</P>\n\n");
            break;

         case ERR_ZERO_DIVIDE:
            printf("Error - can't divide by
zero.</P>\n\n");
            break;

         case ERR_BAD_OPERATOR:
            printf("Error - invalid
operator.</P>\n\n");
            break;

         default:
            printf("%d %s %d = %d</P>\n\n", a,
op, b, result);
            break;
         }

      }

      printf("<BR><HR><BR>\n\n");


      /*  Output calculator form  */

      printf("<FORM ACTION=\"/cgi-bin/calc\"
METHOD=\"GET\">\n");
      printf("<TABLE BORDER=\"1\"
CELLPADDING=\"10\">\n");
      printf("    <TR>\n");
      printf("        <TH>First number</TH>\n");
      printf("        <TH>Operator</TH>\n");
      printf("        <TH>Second number</TH>\n");
      printf("    </TR>\n\n");
      printf("    <TR>\n");
      printf("        <TD><INPUT TYPE=\"text\"
NAME=\"number1\" SIZE=\"8\"></TD>\n");
      printf("        <TD><SELECT
NAME=\"operator\">\n");
      printf("
<OPTION>plus</OPTION>\n");
      printf("
<OPTION>minus</OPTION>\n");
      printf("            <OPTION>multiplied
by</OPTION>\n");
      printf("            <OPTION>divided
by</OPTION>\n");
```

```
      printf("            <OPTION>to the power
of</OPTION>\n");
      printf("        </SELECT></TD>\n");
      printf("        <TD><INPUT TYPE=\"text\"
NAME=\"number2\" SIZE=\"8\"></TD>\n");
      printf("    </TR>\n\n");
      printf("    <TR>\n");
      printf("        <TD
COLSPAN=\"3\"><CENTER>");
      printf("<INPUT TYPE=\"submit\"
VALUE=\"Calculate!\">");
      printf("</CENTER></TD>\n");
      printf("    </TR>\n");
      printf("</TABLE>\n");
      printf("</FORM>\n\n");

      printf("</BODY>\n\n</HTML>\n");
    }


    /*  Clean up and exit  */

    FreeCGI();
    return EXIT_SUCCESS;
}
```

---

cgihelp.h

```
/*

  CGIHELP.H
  =========
  (c) Paul Griffiths 1999
  Email: mail@paulgriffiths.net

  Interface to CGI helper functions

*/


#ifndef PG_CGIHELP
#define PG_CGIHELP


/*  Global macros  */

#define ERR_BADREQUEST          (1)
#define ERR_BADMALLOC           (2)
```

```
/*  Function declarations  */

int    InitInput      (int   maxbuffer);
char * GetValue       (char *key);
char * GetIndexedValue(int   index);
char * GetIndexedKey  (int   index);
void   FreeCGI        ();


#endif    /*  PG_CGIHELP  */
```

---

cgihelp.c

```
/*

  CGIHELP.C
  =========
  (c) Paul Griffiths 1999
  Email: mail@paulgriffiths.net

  CGI helper functions

*/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "cgihelp.h"


/*  Forward function declarations  */

void cleanurl(char *str);


/*  Linked list node to hold key/value pairs  */

typedef struct varlist {
    char *key;
    char *value;
    struct varlist *next;
} varlist;

varlist* varstart;


/*  Get all key/value pairs from input  */
```

```
int InitInput(int maxbuffer) {
    varlist *var;
    char    *input;


    /*  Allocate memory for input buffer  */

    if ( !(input = malloc(sizeof(char) *
maxbuffer)) )
      return ERR_BADMALLOC;
    memset(input, '\0', sizeof(char) *
maxbuffer);


    /*  Determine request method and fill input
buffer  */

    if ( !strcmp(getenv("REQUEST_METHOD"), "GET")
)
      strncpy(input, getenv("QUERY_STRING"),
maxbuffer - 1);
    else if ( !strcmp(getenv("REQUEST_METHOD"),
"POST") )
      fgets(input, maxbuffer, stdin);
    else
      return ERR_BADREQUEST;


    /*  Allocate memory for first key/value pair
*/

    if ( ! (varstart = malloc(sizeof(varlist)) )
)
      return ERR_BADMALLOC;
    var = varstart;


    /*  Populate key/value pairs  */

    if ( (input = strtok(input, "=")) ) {
      while ( var ) {
          if ( !(var->key = malloc(sizeof(char) *
strlen(input))) )
              return ERR_BADMALLOC;
          strcpy(var->key, input);
          cleanurl(var->key);

          input = strtok(NULL, "&");
          if ( !(var->value = malloc(sizeof(char)
* strlen(input))) )
              return ERR_BADMALLOC;
          strcpy(var->value, input);
          cleanurl(var->value);

          if ( (input = strtok(NULL, "=")) ) {
```

```
            if ( !(var->next =
malloc(sizeof(varlist))) )
                return ERR_BADMALLOC;
        }
        else {
            var->next = NULL;
        }
        var = var->next;
     }
    }
    else {

     /*  No key/value pairs in input if we get
here  */

     free(varstart);
     varstart = NULL;
    }

    free(input);
    return 0;
}


/*  Retrieves a value from a key  */

char * GetValue(char *key) {
    varlist *var = varstart;
    while ( var ) {
     if ( !strcmp(key, var->key) )
         return var->value;
     var = var->next;
    }
    return NULL;
}


/*  Retrieves a key using a 1-based index  */

char * GetIndexedKey(int index) {
    int n;
    varlist *var = varstart;
    for ( n = 1; n < index; ++n )
     if ( var->next == NULL )
         return NULL;
     else
         var = var->next;
    return var->key;
}


/*  Retrieves a value using a 1-based index  */

char * GetIndexedValue(int index) {
    int n;
    varlist *var = varstart;
    for ( n = 1; n < index; ++n )
```

```
      if ( var->next == NULL )
          return NULL;
      else
          var = var->next;
    return var->value;
}


/*  Frees memory used by key/value pairs  */

void FreeCGI() {
    varlist *var = varstart;
    while ( var ) {
      free(var->key);
      free(var->value);
      var = var->next;
      free(varstart);
      varstart = var;
    }
}


/*  Cleans up url-encoded string  */

void cleanurl(char *str) {
    char asciinum[3] = {0};
    int i = 0, c;

    while ( str[i] ) {
      if ( str[i] == '+' )
          str[i] = ' ';
      else if ( str[i] == '%' ) {
          asciinum[0] = str[i+1];
          asciinum[1] = str[i+2];
          str[i] = strtol(asciinum, NULL, 16);
          c = i+1;
          do {
              str[c] = str[c+2];
          } while ( str[2+(c++)] );
      }
      ++i;
    }
}
```

Questions: -

1. Try above program and check o/p