Lesson 29

Objectives: -

- C-To-HTML
- Programming Issues

C-To-HTML

Function

This program translates one or more C source files into a well-formed, valid XHTML1.0 document. It was used to create all the C program listings on this site.

The program works in the following way:

- Output basic HTML header information, such as <head> and <title> information, and the main page heading.
- Copy the specified C source files onto the page, surrounded by <pre> tags to preserve spacing.
- "Markup" keywords, standard library functions, preprocessig directives and comments with appropriate tags for formatting. Note that this assumes you have a separate cascading style sheet file which actually defines the specific format to use for keywords and comments (e.g to display keywords in blue, comments in red etc). If you do not, it is a fairly trivial matter to add style sheet commands to the top of the document, or to explicitly specify background images, or to search and replace the tags with more explicit formatting tags.
- Output basic HTML footer information, such as closing <body> and <html> tags.

The program will currently only correctly translate C89 source files, and does not support any additional keywords, library functions or preprocessing directives

from the new C99 standard, although C++ style single line comments are supported.

Programming Issues

String handling like this is fairly complex in C, and most of the issues involved revolve around this. The following are the most significant:

- Retrieving the command line arguments, and opening/closing the specified files in an orderly manner.
- Replacing characters which won't output properly in HTML (such as "<") with suitable alternatives.
- Finding and checking each word, and determining whether or not it is a keyword, and whether or not it is inside a comment or a string (in which case it should not be marked up)
- If it is a keyword, comment, standard library function or comment, inserting markup commands at the appropriate points.

Usage

Use the following syntax:

```
ctohtml file1.c file2.c file3.c > output.html
```

main.c

```
/*
  MAIN.C
  ======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  C-to-XHTML converter
*/


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

#include "file.h"
```

```
#include "process.h"
#include "html.h"


int main(int argc, char *argv[]) {

    FILE * infile;
    char filename[MAX_FILENAME_LEN] = {0};
    char message[MAX_FILENAME_LEN + 256];


    OutputHTMLHeader();


    /*  Loop through input files and process  */

    while ( (infile = GetNextInputFile(argc,
argv, filename)) ) {
        ProcessFile(infile, filename);

        if ( fclose(infile) == EOF ) {
            sprintf(message, "couldn't close
input file %s", filename);
            perror(message);
        }
    }


    OutputHTMLFooter();

    return EXIT_SUCCESS;
}
```

---

file.h

```
/*
  FILE.H
  ======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Interface to input and output file operations
*/


#ifndef PG_FILE_H
#define PG_FILE_H

#define MAX_FILENAME_LEN 256

FILE * GetNextInputFile(int argc, char *argv[],
char * filename);
```

```
#endif  /* PG_FILE_H */
```

---

file.c

```
/*
  FILE.C
  ======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Input and output file operations
*/


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

#include "file.h"


/*  Gets the next input file from the command
line and returns
    a FILE pointer to it, or NULL if no more
input files.       */

FILE * GetNextInputFile(int argc, char *argv[],
char * filename) {
    static int n = 1;
    FILE * fp;
    char message[MAX_FILENAME_LEN + 256];

    while ( n < argc ) {
        if ( (fp = fopen(argv[n++], "r")) == NULL
) {
            sprintf(message, "couldn't open input
file %s", argv[n-1]);
            perror(message);
            continue;
        }
        else {
            strncpy(filename, argv[n-1],
MAX_FILENAME_LEN - 1);
            return fp;
        }
    }

    return NULL;
}
```

---

process.h

```
/*
  PROCESS.H
  =========
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Interface to input file processing routines
*/


#ifndef PG_PROCESS_H
#define PG_PROCESS_H

#include <stdio.h>

#define MAX_TOKEN_LEN 30
enum tokens { NOTOKEN, COMMENT, CPP_COMMENT,
KEYWORD, PREPROC, STDFUNC };

void ProcessFile(FILE * infile, const char *
filename);

#endif  /* PG_PROCESS_H */
```

---

process.c

```
/*
  PROCESS.C
  =========
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Input file processing routines
*/


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

#include "process.h"
#include "html.h"
#include "token.h"


#define M_COMMENT  0
#define M_KEYWORD  1
#define M_PREPROC  2
#define M_STDFUNC  3

static int GetNextToken(FILE * infile);
static void OutputToken(FILE * infile, int
tokentype);
```

```
/*  Processes an input file  */

void ProcessFile(FILE * infile, const char *
filename) {
    int tokentype;

    OutputFileHeader(filename);

    while ( (tokentype = GetNextToken(infile)) >=
0 ) {
        OutputToken(infile, tokentype);
    }

    OutputFileFooter();
}


/*  Finds the next token and returns its type  */

int GetNextToken(FILE * infile) {
    enum tokens tokentype = NOTOKEN;
    char buffer[MAX_TOKEN_LEN];
    static int quote = 0;
    static int escape = 0;
    fpos_t pos;
    int c;


    /*  Output characters until we hit a markup
token candidate  */

    for ( ;; ) {
        c = getc(infile);

        if ( c == EOF )
            return -1;


        /*  See if next character is consistent
with the start of
            a keyword, library function, comment
or preprocessing
            directive. If it is, and it isn't
quoted, exit loop.   */

        if ( (islower(c) || c == '/' || c == '#')
&& quote == 0 )
            break;
        else {

            /*  Logic to tell whether we are in a
quote  */

            if ( quote == 0 && !(escape % 2) &&
(c == '"' || c == '\'') )
```

```
                quote = c;
            else if ( quote == c && !(escape % 2)
)
                quote = 0;


            /*  Quotation marks can be escaped,
so avoid confusion  */

            if ( quote && c == '\\' )
                ++escape;
            else
                escape = 0;
        }
        TranslateToken(c);
    }


    /*  Place first character of token candidate
back onto stream  */

    ungetc(c, infile);


    /*  Save file position  */

    if ( fgetpos(infile, &pos) ) {
        perror("Error getting file position");
        exit(EXIT_FAILURE);
    }


    /*  Read biggest possible token size into
buffer  */

    fgets(buffer, MAX_TOKEN_LEN, infile);


    /*  Find out what type of token we have, if
any  */

    if ( !strncmp(buffer, "/*", 2) )
        tokentype = COMMENT;
    else if ( !strncmp(buffer, "//", 2) )
        tokentype = CPP_COMMENT;
    else {
        if ( IsToken(buffer, KEYWORD) )
            tokentype = KEYWORD;
        else if ( IsToken(buffer, PREPROC) )
            tokentype = PREPROC;
        else if ( IsToken(buffer, STDFUNC) )
            tokentype = STDFUNC;
    }


    /*  Reset file position to beginning of
token,
```

```
        because we will be outputting it soon
*/

    if ( fsetpos(infile, &pos) ) {
        perror("Error setting file position");
        exit(EXIT_FAILURE);
    }

    return tokentype;
}


/*  Outputs the token of the specified
    type at the current file position   */

void OutputToken(FILE * infile, int tokentype) {
    static char * class[] = { "comment",
"keyword", "preproc", "stdfunc" };
    int c;


    /*  Print token & markup according to what
type it is  */

    switch ( tokentype ) {

    case COMMENT:
        printf("<span class=\"%s\">",
class[M_COMMENT]);
        putchar(getc(infile));
        putchar(getc(infile));
        {
            /*  Don't markup anything else till
end of comment  */

            int c[2];
            TranslateToken((c[0] =
getc(infile)));
            for ( ;; ) {
                c[1] = getc(infile);

                if ( c[1] == EOF )
                    break;

                TranslateToken(c[1]);

                if ( c[0] == '*' && c[1] == '/' )
                    break;

                c[0] = c[1];
            }
        }
        printf("</span>");
        break;

    case CPP_COMMENT:
```

```
        printf("<span class=\"%s\">",
class[M_COMMENT]);
        for ( ;; ) {
            c = getc(infile);

            /*  Don't markup anything else till
end of line or file  */

            if ( c == '\n' || c == EOF ) {
                ungetc(c, infile);
                printf("</span>");
                break;
            }

            TranslateToken(c);
        }
        break;

    case KEYWORD:
    case STDFUNC:
        printf("<span class=\"%s\">",
               class[tokentype == KEYWORD ?
M_KEYWORD : M_STDFUNC]);

        /*  Rest of token consists of lower case
chars and underscore  */

        c = getc(infile);
        while ( islower(c) || c == '_' ) {
            putchar(c);
            c = getc(infile);
        }

        ungetc(c, infile);
        printf("</span>");
        break;

    case PREPROC:
        printf("<span class=\"%s\">",
class[M_PREPROC]);

        putchar(getc(infile));
        while ( islower((c = getc(infile))) )
            putchar(c);

        ungetc(c, infile);
        printf("</span>");
        break;

    case NOTOKEN:

        /*  First char is lower case, or a '#' or
a '/', but
            isn't recognised as a valid token for
markup, so
            just output it without marking it up.
*/
```

```
        c = getc(infile);
        if ( islower(c) || c == '#' ) {
            putchar(c);
            c = getc(infile);
            while ( islower(c) || c == '_' ) {
                putchar(c);
                c = getc(infile);
            }
            ungetc(c, infile);
        }
        else
            putchar(c);

        break;

    default:
        break;
    }
}
```

---

token.h

```
/*
  TOKEN.H
  =======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Interface to utilities for finding a string
token
*/


#ifndef PG_TOKEN_H
#define PG_TOKEN_H

#include "process.h"

int IsToken(const char * token, enum tokens
tokentype);

#endif  /* PG_TOKEN_H */
```

---

token.c

```
/*
  TOKEN.C
  =======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net
```

```
  Utilities for testing for a string token
*/


#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "token.h"
#include "process.h"


int cmptoken(const void * s1, const void * s2);


/*  Returns 1 if the first argument is a
    token of the specified type, 0 otherwise  */

int IsToken(const char * token, enum tokens
tokentype) {
    static const char * stdfuncs[] = { "abort",
"abs", "acos", "asctime",
        "asin", "assert", "atan", "atan2",
"atexit", "atof", "atoi", "atol",
        "bsearch", "calloc", "ceil", "clearerr",
"clock", "cos", "cosh",
        "ctime", "difftime", "div", "exit",
"exp", "fabs", "fclose", "feof",
        "ferror", "fflush", "fgetc", "fgetpos",
"fgets", "floor", "fmod",
        "fopen", "fprintf", "fputc", "fputs",
"fread", "free", "freopen",
        "frexp", "fscanf", "fseek", "fsetpos",
"ftell", "fwrite", "getc",
        "getchar", "getenv", "gets", "gmtime",
"isalnum", "isalpha", "iscntrl",
        "isdigit", "isgraph", "islower",
"isprint", "ispunct", "isspace",
        "isupper""isxdigit", "labs", "ldexp",
"ldiv", "localtime", "log",
        "log10", "longjmp", "malloc", "memchr",
"memcmp", "memcpy", "memmove",
        "memset", "mktime", "modf", "perror",
"pow", "printf", "putc",
        "putchar", "puts", "qsort", "raise",
"rand", "realloc", "remove",
        "rename", "rewind", "scanf", "setbuf",
"setjmp", "setvbuf", "signal",
        "sin", "sinh", "sprintf", "sqrt",
"srand", "sscanf", "strcat",
        "strchr", "strcmp", "strcpy", "strcspn",
"strerror", "strftime",
        "strlen", "strncat", "strncmp",
"strncpy", "strpbrk", "strrchr",
        "strspn", "strstr", "strtod", "strtok",
"strtol", "strtoul", "system",
```

```
        "tan", "tanh", "time", "tmpfile",
"tmpnam", "tolower", "toupper",
        "ungetc", "va_arg", "va_list",
"va_start", "vfprintf", "vprintf",
        "vsprintf" };

    static const char * keywords[] = { "auto",
"break", "case", "char",
        "const", "continue", "default", "do",
"double", "else", "enum",
        "extern", "float", "for", "goto", "if",
"int", "long", "register",
        "return", "short", "signed", "sizeof",
"static", "struct", "switch",
        "typedef", "union", "unsigned", "void",
"volatile", "while" };

    static const char * preprocs[] = { "#define",
"#elif", "#else", "#endif",
        "#error", "#if", "#ifdef", "#ifndef",
"#include", "#line", "#pragma" };

    static size_t numfunc = sizeof stdfuncs /
sizeof *stdfuncs;
    static size_t numkey  = sizeof keywords /
sizeof *keywords;
    static size_t numproc = sizeof preprocs /
sizeof *preprocs;

    int i = 1;
    void * found;
    char temp[MAX_TOKEN_LEN];


    /*  Make copy of token candidate and trim it
*/

    strcpy(temp, token);
    while ( temp[i] ) {
        if ( !islower(temp[i]) && temp[i] != '_'
) {
            temp[i] = 0;
            break;
        }
        ++i;
    }


    /*  See if it is in the token lists  */

    switch ( tokentype ) {
    case STDFUNC:
        found = bsearch(temp, stdfuncs, numfunc,
sizeof *stdfuncs, cmptoken);
        break;

    case KEYWORD:
```

```
        found = bsearch(temp, keywords, numkey,
sizeof *keywords, cmptoken);
        break;

    case PREPROC:
        found = bsearch(temp, preprocs, numproc,
sizeof *preprocs, cmptoken);
        break;

    default:
        found = NULL;
    }

    if ( found )
        return 1;
    else
        return 0;
}


/*  Compare function used in above calls to
bsearch()  */

int cmptoken(const void * s1, const void * s2) {
    return strcmp(s1, *((char **)s2));
}
```

---

html.h

```
/*
  HTML.H
  ======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Interface to HTML output functions
*/


#ifndef PG_HTML_H
#define PG_HTML_H

void OutputHTMLHeader(void);
void OutputHTMLFooter(void);
void OutputFileHeader(const char * filename);
void OutputFileFooter(void);
void TranslateToken(int c);

#endif  /* PG_HTML_H */
```

---

html.c

```
/*
  HTML.C
  ======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  HTML functions
*/


#include <stdio.h>

#include "html.h"


/*  Output XHTML prolog, head, and opening body
tag  */

void OutputHTMLHeader(void) {
    static const char * css_location =
"/css/c.css";

    puts("<?xml version=\"1.0\" encoding=\"UTF-
8\"?>");
    puts("<!DOCTYPE html PUBLIC \"-//W3C//DTD
XHTML 1.0 Strict//EN\"");
    puts("
\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd\">\n");

    puts("<html
xmlns=\"http://www.w3.org/1999/xhtml\">\n");

    puts("<head>");
    puts("    <title>CtoHTML Output</title>");
    printf("    <link rel=\"stylesheet\"
href=\"%s\" type=\"text/css\" />\n",
           css_location);
    puts("    <meta name=\"generator\"
content=\"CtoHTML\" />");
    puts("</head>\n");

    puts("<body>\n\n");
}


/*  Output email link and closing XHTML document
tags  */

void OutputHTMLFooter(void) {
    static const char * email =
"mail@paulgriffiths.net";

    puts("<!--  Email link  -->\n");
    puts("<div id=\"link\">\n");
    puts("<p class=\"email\">");
```

```
    printf("Please send any comments,
suggestions, bug reports to ");
    printf("<a href=\"mailto:%s\">%s</a>\n",
email, email);
    puts("</p>\n<hr />\n\n</div>\n");
    puts("</body>\n");
    puts("</html>");
}


/*  Output file heading and create division  */

void OutputFileHeader(const char * filename) {
    printf("<!--  Start of %s listing  -->\n\n",
filename);
    printf("<div id=\"%s\">\n", filename);
    printf("<h2>%s</h2>\n\n", filename);
    printf("<pre>");
}


/*  Close file division  */

void OutputFileFooter(void) {
    puts("</pre>");
    puts("<hr />");
    puts("</div>\n\n");
}


/*  Translate certain HTML escaped entities  */

void TranslateToken(int c) {
    switch(c) {
    case  '<':
        printf("&lt;");
        break;

    case '>':
        printf("&gt;");
        break;

    case '&':
        printf("&amp;");
        break;

    case '"':
        printf("&quot;");
        break;

    default:
        putchar(c);
        break;
    }
}
```

Questions: -

1. Try above program and check o/p