Programming Project #1
C to HTML Translator

CS 452

Due 11:59 pm, Friday the 1st of February

1 Introduction

For this project you will write a program that converts C source code into HTML that renders the lexical elements of the C code using a variety of colors, styles, and typefaces. The output is also annotated with line numbers. The actual program text is not re-formated. A true "pretty print" program that reformats the code would require some syntax analysis which we avoid here.

2 Lexical Elements

Your program must identify the following lexical elements of C source program.

1. **Directives** consist of all characters following (and including) a pound sign `#` to the end of the line (but not including the newline `\n`).
2. **Multiline comments** begin with the characters `/*` and terminate with the characters `*/`. They can include newlines and do not nest.
3. **Single line comments** consist of all characters following (and including) the characters `//` to the end of the line (but not including the newline).
4. **White space** are spaces, horizontal `\t` and vertical tabs `\v`, form-feeds `\f`, and carriage returns `\r` (newlines `\n` handled separately).
5. **Newlines** `\n` mark the end of a line and indicate when a new line begins.
6. **Keywords** are listed in Table 1.

`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`

Table 1: These identifiers are reserved as keywords and may not be used otherwise.

7. **Identifiers** begin with an alphabetic character or underscore `_` and are followed by zero or more alpha-numeric characters or underscores.

8. **Integer constants** are represented in octal, hexadecimal, or decimal. Octal constants begin with `0` and are followed by zero or more octal digits. Hexadecimal constants begin with the characters `0x` or `0X` and are followed by one or more hexadecimal digits. Decimals constants begin with a decimal digit (not including 0) and a followed by zero or more decimal digits. Integer constants may be suffixed with a `u` or `U`. They may also be suffixed with a `l` or `L`.

9. **Character constants** consist of one (single character constants) or more characters (multi-character constants) enclosed in single quotes (*e.g.,* `'x'`). A single quote character must be "escaped" `'\''`. `'\a'`, `'\007'` and `'\x07'` all represent an ASCII bell character. Character constants do not include a (literal) newline.

10. **Floating point constants** can be broken into an integer part, a decimal point `.`, a fractional part, an optional signed exponent, and an optional suffix:

    ```
    integer . fraction exponent suffix
    ```

    Either the integer or fraction part (not both) can be missing. The exponent part begins with either `e` or `E` and is followed by an (optionally signed) integer. The suffix can be `f`, `F`, `l`, or `L`.

11. **String literals** are sequences of characters surrounded by double quotes `"`. Obviously double quotes characters in strings will have to be escaped `\"` (or encoded with a multi-character constant). Literal newlines can not be included in a string.

12. **Operators and separators** are listed in Table 2.

`{`	`}`	`(`	`)`	`[`
`]`	`;`	`,`	`=`	`+=`
`-=`	`*=`	`/=`	`%=`	`&=`
`^=`	`|=`	`<<=`	`>>=`	`?`
`:`	`||`	`&&`	`|`	`^`
`&`	`==`	`!=`	`<=`	`>=`
`<`	`>`	`<<`	`>>`	`+`
`-`	`*`	`/`	`%`	`!`
`~`	`++`	`--`	`->`	`.`

Table 2: Operators and separators.

13. **Invalid lexical elements** are characters that are not part of any of the above elements.

3 HTML

The overall skeleton of the output HTML shown in Figure 1. The `<pre>` tags specify a fixed-width font and formats the content exactly as it is given (except for other html meta-characters the text formatting is unaltered). This is desirable for displaying source code. We can specify that the keyword `while` be *blue* and *bold* as follows:

```
<font color=#0000FF><b>while</b></font>
```

```
<html>
<head><title>program.c</title></head>
<body bgcolor=#FFFFFF>
<pre>

...formatted code goes here...

</pre>
</body>
</html>
```

Figure 1: Skeleton of output HTML.

symbol	*HTML*
<	<
>	>
"	"
&	&

Table 3: Special characters (left) and their corresponding HTML code (right).

The `<color>` tag argument consists of six hexadecimal characters (specifying three bytes) which encodes the red, green, and blue channels respectively. The `<b>` tag specifies bold. Other useful tags are `<i>` and `<u>` for italics and underline respectively. Some characters are interpreted as HTML meta-characters and require special HTML lingo to generate them as shown in Table 3. Consult an HTML manual for additional features.

Figure 3 show the rendered output of my program for the source code listed in Figure 2. Note that line numbers have been added. Table 4 lists suggested colors, fonts, and formats to use for the lexical elements listed in Section 2 and meant to be used with a white background.

lexical element	*format*
directives	black, regular type
comments	light green, regular type
white space	white space
newlines	start new line, add black line number, regular type
keywords	blue, bold type
identifiers	blue, regular type
integer constants	cyan, italic type
character constants	light red, italic type
floating point constants	cyan, italic type
string literals	light red, italic, underline type
operators	black, regular type
invalid lexemes	red, blinking type

Table 4: Lexical formatting.

```
/*
 * Filter that excludes bytes larger than 126.
 */
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  int c, n = 0;
  while ((c = fgetc(stdin)) != EOF)
    if (c <= 126)  /* only 7-bit ASCII allowed */
      fputc(c, stdout);
    else
      n++;
  if (n > 0)
    fprintf(stderr, "%d bytes filtered out!\n", n);
  return 0;
}
```

Figure 2: Example input C program.

4 Implementation details

I suggest using *Lex* (or similar tool) to generate your lexical analyzer, but you are free to "roll your own." Your program should read the C source file name from the command line which should be used to title the web page via the `<title>` tag. The program writes the generated HTML to `stdout`:

```
./c2html program.c > program.html
```

5 What to submit

You will archive your source code and supporting files and submit the archive electronically through the class web site. Include a `README` file that contains the following information:

- Your name and email address.
- A brief description of the project.
- A description of how to build your program from source code.
- A description, with examples, of how to use your program.
- A list of the files in the archive.

Submit your solution by midnight on the due date. Have fun.

```
1   /*
2    * Filter that excludes bytes larger than 126.
3    */
4   #include <stdio.h>
5   #include <stdlib.h>
6
7   int main(void) {
8     int c, n = 0;
9     while ((c = fgetc(stdin)) != EOF)
10      if (c <= 126)  /* only 7-bit ASCII allowed */
11        fputc(c, stdout);
12      else
13        n++;
14    if (n > 0)
15      fprintf(stderr, "%d bytes filtered out!\n", n);
16    return 0;
17  }
18
19
```

Figure 3: Rendered HTML.