# A Hyper-Text Tutorial Markup Language

Brian L. Stuart
Math/Computer Science Department, Rhodes College
Memphis, TN
stuart@mathcs.rhodes.edu

## Abstract

Due to experience with a Web-based C programming tutorial, we have learned of the need for an authoring system for such tutorials. Here we describe an extension of HTML we call Hyper-Text Tutorial Markup Language (HTTML) and a translator that takes an HTTML file and produces the corresponding HTML files and CGI script sources.

## 1 Background

The CS1 course at Rhodes takes a breadth-first approach with a lab component that focuses on programming such as discussed in [1,4]. Teaching the details of the programming language (C in our case) has always been problematic. Lecturing about where the semicolons go has never been productive nor has throwing them in and telling them to swim by writing programs from day one. Learning the language just from reading examples doesn't work well either since students tend to gloss over important details.

To better convey the details of the language, we developed an on-line tutorial for the C programming language as reported in [9]. Studies of the impact of such Computer-Assisted Instruction approaches are reported in [2,8,11]. This tutorial operates across the Web. The student uses a Web browser to access it. HTML pages present features of the C language, and most end in a problem to be solved. These problems are presented using HTML forms [3]. When the student submits a form, a CGI script on the host computer checks the answer given by the student. If the answer is correct, the CGI script returns a page reporting that with a pointer to the next page in the tutorial. Otherwise, the CGI script returns a page reporting that the answer is incorrect, and for some combinations of question and answer, it includes an explanation of why the answer is wrong. The tutorial has been successful and is now in its second year of use at Rhodes.

While the tutorial has been successful as a tool in teaching the C programming language, its development has little to recommend it for those who wish to write similar tutorials. Coordinating the HTML forms for questions, the CGI scripts (which were written in Lex [6]) and the HTML response files proved to be a tedious task at best and somewhat error-prone. Also the use of Lex for CGI scripts is not easily portable to some platforms. Therefore, we have embarked on a project to develop an authoring system for such tutorials.

## 2 The Markup Language

The authoring system is based on a script file that describes the tutorial in the form of lessons divided into parts. Each part may contain a question to be answered by the student. Along with the question itself, the script file describes what happens for both right and wrong answers to the question. This script file is given in the Hyper-Text Tutorial Markup Language (HTTML). HTTML is an extension of the Hyper-Text Markup Language (HTML) [7] intended to facilitate authoring interactive tutorials on the World-Wide Web (WWW). It adds, to HTML, tags that delineate parts within a lesson and tags that describe short answer, one-of-many multiple choice and many-of-many multiple choice questions.

### 2.1 HTTML Specific Tags

The new tags which we have added to HTML describe the structure of a tutorial with most describing the question and answer part of a tutorial. These new tags include:

`<lesson>`

`</lesson>` Delimit a lesson. A lesson represents a single session in the tutorial. For each HTTML file, the lessons are indexed into an output index file.

`<part>`

`</part>` Delimit a part of a lesson. Parts are intended to be done in succession and are listed in the index.

`<omc-question>`

`</omc-question>` Delimit a one-of-many multiple choice question. The multiple choice questions work much like lists in HTML.

`<mmc-question>`

`</mmc-question>` Delimit a many-of-many multiple choice question.

`<sa-question>`

`</sa-question>` Delimit a short answer question.

`<choice>` Identify a potential answer to a question. This tag works like the `<li>` tag in HTML. For multiple choice questions, the choices are listed as labels beside buttons in the browser. For short answer questions, choices are regular expressions that define specifically recognized answers.

`<right>` Mark this answer as a correct one. For many-of-many multiple choice questions, all right answers must be marked for the question to be counted correct. For one-of-many multiple choice questions, any of the right answers will be accepted. The default behavior of a right tag is as if it had been given as `<right next>` indicating that if the right answer(s) are given, then the tutorial moves on to the next part. Alternatively, another page may be identified with a syntax similar to the anchor tag, i.e. `<right href=...>`. The last possibility is to specify `<right repeat>` which would indicate that the question is to be repeated if answered correctly. (This option is not very useful but is included for consistency with the `wrong` tag.) When different destinations are given for the right answers in a many-of-many multiple choice question, the actual destination is implementation dependent.

`<wrong>` Mark this answer as a wrong one. Unlike the right tag, any wrong answer given will result in the question being declared wrong regardless of the type of question. The wrong tag may also be used with the same destination specifiers as in the right tag. The default behavior for wrong tags is to repeat.

## 2.2 Grammar for HTTML Tags

In this subsection, we present a YACC-like grammar for the new tags introduced for HTTML. We will use the tokens `OTHER` and `OTHER_TAG` to represent text outside of tags and non-HTTML tags respectively. The grammar as actually used includes productions that absorb any errors and attempts to move through them making a best-guess attempt at output rather than issuing error messages to the user. Those error productions are not included here. The grammar as actually used also contains some singleton productions which have been removed here for clarity

We begin by defining a grammar that is idempotent. That is, we want one of the output files (which contain only HTML code without any HTTML tags) to produce itself as output. Alternatively, the file may contain one or more lessons.

```
file:   /* nothing */
   | file OTHER
   | file OTHER_TAG
   | file lesson
   ;
```

Each lesson in the file contains zero or more parts. The actual grammar as implemented also includes the possibility of quizzes and tests, but currently they do exactly the same thing as lessons and have been removed here.

```
lesson: LESSON parts END_LESSON
   ;
```

The parts that make up a lesson may in part or in whole be normal HTML code not delimited by the `<part>` and `</part>` tags. Such code is included in the index file that is created from the HTTML code.

```
parts: /* nothing */
   | parts part
   | parts OTHER
   | parts OTHER_TAG
   ;

part: PART part_body END_PART
   ;
```

Each part will typically consist of some HTML code which teaches some particular concept followed by a question. We do provide limited support for multiple questions per part, but the basic model calls for one.

```
part_body: /* nothing */
   | part_body OTHER
   | part_body OTHER_TAG
   | part_body omc_question
   | part_body mmc_question
   | part_body sa_question
   ;
```

The questions all have pretty much the same structure, namely the text of the question followed by zero or more choices for the answer. (In practice, of course, zero choices wouldn't behave in a useful way.) The question text is given in normal HTML code.

```
omc_question: OMC_QUESTION question_head
   omc_body END_OMC_QUESTION
   ;

mmc_question: MMC_QUESTION question_head
   mmc_body END_MMC_QUESTION
   ;

sa_question: SA_QUESTION question_head
   sa_body END_SA_QUESTION
   ;

question_head: /* nothing */
   | question_head OTHER
   | question_head OTHER_TAG
   ;

omc_body: /* nothing */
   | omc_body choice
   ;

mmc_body: /* nothing */
   | mmc_body choice
   ;

sa_body: /* nothing */
   | sa_body choice
   ;
```

Each choice for an answer is labeled as either right or wrong, and each has a response associated with it. For each choice, there is also an indication of what page we should point to next. By default, right answers point to the page for the next part and wrong answers point back to the page with the question that the student got wrong. These can be overridden by the use of the `next` and `repeat` keywords. Using an `href=` in the tag, we can also specify an arbitrary next page.

```
choice: CHOICE choice_body RIGHT response
   | CHOICE choice_body RIGHT_NEXT response
   | CHOICE choice_body RIGHT_REPEAT response
   | CHOICE choice_body RIGHT_HREF response
   | CHOICE choice_body WRONG response
   | CHOICE choice_body WRONG_NEXT response
   | CHOICE choice_body WRONG_REPEAT response
   | CHOICE choice_body WRONG_HREF response
   ;

choice_body: /* nothing */
   | choice_body OTHER
   | choice_body OTHER_TAG
   ;

response: /* nothing */
   | response OTHER
   | response OTHER_TAG
   ;
```

## 3 HTTML Processor

A file of HTTML code is processed by a translator that produces a collection of HTML files and CGI script files. The translator is much like a simple compiler. It has a parser implemented in YACC [5]. The parser collects the HTML portions of the file and writes them to several HTML output files. For the entire tutorial script, an index file is written that is divided into lessons and parts within each lesson. Each part of a lesson is written to its own file and many of the responses are written to files.

For each question, an HTML form is written to the output for the file that contains the question. In addition to the HTML form, the translator produces the source code for a CGI script that processes the form. The translator component that generates the CGI source code forms a back-end analogous to a code generator in a compiler. At the present time, we have implemented two such back-ends for the HTTML translator. The first back-end produces CGI scripts in Lex following the form used in our original C programming tutorial. We have also written a back-end that produces AppleScript code for the CGI scripts so that they may be run on the Apple Macintosh.

## 4 Example

The language is illustrated here by way of a short example.

```
<lesson>
<part>
<omc-question>
What color is the sky?
<choice>red <wrong>Maybe at sunset
<choice>blue <right>Congratulations
<choice>green <wrong>If the sky's green on your
planet, I'd love to visit.
<choice>yellow <wrong>Have you been staring at
the sun again?
</omc-question>
</part>
<part>
<mmc-question>
What computers did Seymore Cray design?
<choice>CDC 6600 <right>The 6600 is considered
one of his landmark designs.
<choice>Apple II <wrong>The Apple II was
designed by Steve Wozniak.
```

```
<choice>Cray 1 <right>Who do you think did
design the Cray 1?
<choice>Cray 2 <right>All of the Cray 1 through
4 series were designed by him.
<choice>ENIAC <wrong>The ENIAC came before his
time.
</mmc-question>
</part>
<part>
<sa-question>
What is the name of the primary CS professional
society?
<choice>AOL <wrong>Look again
<choice>ACM <right>By George, you've got it.
</sa-question>
</part>
</lesson>
```

Assuming this code is contained in the file `example.httml`, it is processed with the command `httml example.httml`. The translator then creates an index file, `example.html` and the HTML files `example-1-1.html`, `example-1-2.html` and `example-1-3.html` which contain HTML code for each of the three parts. A fourth HTML file, `example-1-4.html` is written which announces completion of the lesson and provides a pointer back to the index.

With only minor differences, the output produced by the translator with the Lex back-end is essentially the same as the code that was used in our C tutorial. The CGI code for the three questions are written to the files `example-1-1.l`, `example-1-2.l` and `example-1-3.l`, and these files are compiled into the executables `example-1-1.cgi`, `example-1-2.cgi` and `example-1-3.cgi`. For the one-of-many multiple choice question the responses are written to the files `example-1-1.wrong1`, `example-1-1.right2`, `example-1-1.wrong3` and `example-1-1.wrong4`. For the many-of-many multiple choice question in Part 2, we have five choices for answers. In general if we have $n$ choices, then there are $2^n$ possible sets of answers the student may give, and we would like to produce a different response for each. Consequently, only the right answer is written to its own file, `example-1-2.right` in this case. The responses for wrong answers are encoded in the CGI script itself. The short-answer question is structured much like the one-of-many. Here we generate the HTML response files `example-1-3.wrong1` and `example-1-3.right2`. For this case, though, we also have an additional file `example-1-3.wrong` that is the response for a generic wrong answer that doesn't match any of the choices specified.

For the AppleScript back-end, we generate the same set of HTML files. Instead of the Lex source code, the translator puts the AppleScript directly into the `.cgi` files

## 5 Future Directions

In addition to the tags discussed here, the translator recognizes tags that delimit quizzes and tests as alternatives to lessons. Our plan is that these tags invoke score keeping activities in the CGI scripts and perhaps change the default behavior of `<right>` and `<wrong>` tags. For example, wrong answers would, by default in tests, go ahead to the next part rather than repeat the current part.

We are also considering implementing a Tool Command Language (TCL) [10] back-end. Since TCL has been ported to a variety of platforms including Unix and X, the Macintosh and MS-Windows, it provides the opportunity to simplify support for multiple platforms with only one back-end. TCL also provides support for regular expressions addressing the original motivation for using Lex in the first place. Also the TCL support for regular expressions is more capable than the limited string recognition that we find in AppleScript. Finally, TCL is interpreted. The compiling time for Lex output dominates the translation time for that back-end. Compiling the AppleScript into executable form has also proved problematic. It does not appear to be possible to drive this process from the translator, so we currently force the user to manually initiate the process. Since the CGI scripts have very little computational overhead, we do not expect any detrimental effect from using interpreted scripts.

Finally, we would like to simplify the use of this language by providing an editor similar to many of the HTML editors available. We would like to have buttons or menu choices that can create lesson, part and question templates to be filled in. Similar facilities could be used to simplify specifying choices and responses to questions.

## References

[1] ACM/IEEE-CS Joint Curriculum Task Force, "Computing Curricula 1991," *CACM,* June 1991, Vol. 34, No. 6, pp 69–84.

[2] Chambers, Jack & Sprecher, Jerry, *Computer-Assisted Instruction,* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[3] Davidson, Andrew, "Coding with HTML Forms," *Dr. Dobb's Journal,* June 1995, Vol. 20, No. 6, pp 70–75.

[4] Denning, Peter, et. al., "Computing as a Discipline," *CACM,* January 1989, Vol. 32, No. 1, pp 9–23.

[5] Johnson, Stephen C., "Yacc: Yet Another Compiler-Compiler," Computer Science Technical Report, No. 32, Bell Laboratories, Murray Hill, NJ, 1975. Reprinted as PS1:15 in *UNIX Programmer's Manual,* Usenix Association, (1986).

[6] Lesk, M. E. & Schmidt, E., "Lex—A Lexical Analyzer Generator," Computer Science Technical Report, No. 39, Bell Laboratories, Murray Hill, NJ, October 1975. Reprinted as PS1:16 in *UNIX Programmer's Manual,* Usenix Association, (1986).

[7] McArthur, Douglas, "World Wide Web & HTML," *Dr. Dobb's Journal,* December 1994, Vol. 19, No. 15, pp 18–26.

[8] Shlechter, Theodore, *Problems and Promises of Computer-Based Training,* Ablex Publishing Corporation, Norwood NJ, 1991.

[9] Stuart, Brian L., "Computer-Assisted Instruction via the World-Wide Web," *Proceedings of the 34th Annual Southeast ACM Conference,* 1996, pp 49–56.

[10] Welch, Brent B., *Practical Programming in Tcl and Tk,* Prentice Hall, Upper Saddle River, NJ, 1995.

[11] Wilkinson, Alex, *Classroom Computers and Cognitive Science,* Academic Press, New York, 1983.