Online Multiple Choice Assessment Tool: User Documentation

Claire Hewson[1]**,**
University of Bolton,

1st July, 2005

[1] The author wishes to acknowledge the valuable assistence and expertise of Peter Yule, who worked on this project to help in developing the tool described here.

Contents

Description, Functionality and Requirements

This document describes a tool to implement online MCQ assessments. The tool, which consists of a number of customisable files, is provided for download at: http://www.clairehewson.co.uk/LTSNproject/MCQtool. A demonstration version, which was used at the University of Bolton as part of an evaluation study, can be viewed at: http://www.clairehewson.co.uk/LTSNproject/example1.html. Since it was initially developed, the tool has evolved in a number of ways to meet a range of needs. Here three different implementations are described (each with slightly different functions) ranging from a procedure which can be implemented with minimal levels of technical equipment and expertise, to a more sophisticated implementation which requires access to a server on which CGI scripts can be run.

1. Formative Assessment

This version of the tool presents respondents with a set of twenty MCQs and provides automatic feedback in terms of an overall score, but does not store answers, thus making it suitable for *formative* type assessments (e.g. practice quizzes). Implementation involves downloading and customising the HTML code as described, and having access to space on a web server on which this HTML form can be placed so it is available via the world wide web.

2. Summative Assessment 1: Using email to store responses

This implementation also presents respondents with twenty MCQs, but instead of providing automatic feedback it stores respondents' answers, including the total score, thus making it more suitable for *summative* types of assessment. This version stores responses by sending them to a specified email address. It also includes an ID-checking function, such that respondents are not able to submit answers unless they input a valid ID. Implementation involves downloading and customising the relevant HTML code, and requires access to space on a web server, as well as an email account.

3. Summative Assessment 2: Using a CGI script to process submitted data

This implementation performs like that above, in that answers are stored but no automatic feedback is provided. However, here, a CGI script is used to process data when the respondent submits their answers. A CGI script is a program that executes and processes the data submitted by a user through an HTML form. This approach has a number of advantages over the previous implementation, including greater levels of reliability, and heightened security over the correct answers to questions. This version also stores information about the respondent's browser type and IP address, as well as time and date of submission. In order to implement this system, access is needed to a web server which allows CGI scripts to be executed. While not all educational establishments will allow ordinary users such access, support technicians may be available to help install the CGI script in an appropriate location. Otherwise, it is possible to purchase space on a commercial web server which will allow users to publish web pages and run CGI scripts, for a cost of around £50 a year. In this implementation both the HTML form and CGI script need to be customised as described in the relevant section below.

Formative Assessment

A working version of this assessment can be viewed by clicking on MCQquiz.html on the MCQ tool webpage: http://www.clairehewson.co.uk/LTSNproject/MCQtool.

Implementation[2]

1. Download the MCQquiz.txt file by right-clicking on 'MCQquiz.txt' and selecting 'Save Target As…'. Save to a location on your computer. Make sure the file is saved as a *text* file, by saving with the name 'MCQquiz.txt' (not 'MCQquiz.html'). Simply specify this in the 'Filename' box; if there is a 'File type' pull down menu, then select type 'text' or 'txt' if available, otherwise select 'All files'.

2. You should now have a file 'MCQquiz.txt' stored on your computer; open this file using a text editor such as Microsoft's *Notepad*. The file should open to display the text (this is the "HTML code") shown in Table 1 (except that Table 1 displays a shortened version containing only the first MCQ; the downloaded file will contain all twenty MCQs). If your text file displays without line breaks and is thus difficult to read, opening in a different application such as *Wordpad* should resolve this. In this case you will need to save again (as a text file) from within this application. The file should now view okay when opened in Notepad.

3. In order to get the code to display correctly in a web browser (i.e. as a web page) it needs to be saved with the extension '.html', i.e. 'MCQquiz.html'. One way to do this is simply rename the 'MCQtest.txt' file as 'MCQtest.html'. Try this and check that the file opens and displays correctly in a web browser. It should look identical to MCQquiz.html on the MCQ tool web page (the way the shortened code in Table 1 displays in a web browser is shown in Figure 1).

4. The next two steps describe how to customise the HTML code to provide MCQs with appropriate content, and to code the correct answers for scoring purposes. Changes should be made to the code within the text editor; as these changes are made, continue to save updated versions as 'MCQquiz.html' and reload these versions into the web browser to view resulting changes to the web page[3]. (Hint: to edit the code just open the file 'MCQquiz.html' in a text editor, make the changes, then re-save as 'MCQquiz.html').

5. Customising the content of the questions simply involves replacing the relevant text, i.e. 'Insert question 1 here', and so on, with whatever content is desired

6. Scoring requires a little deeper explanation. Scoring is carried out by a javascript routine in the HTML form ("function score()"). The correct answers are coded in lines like this one: **if(document.myform[1].checked==true){x = x+1}**, which need to be edited by changing the number in square brackets. Basically, each

[2] Browser details provided in these instructions refer to Internet Explorer, version 6.0. Note that alternative browsers may present slightly different menu options.

[3] To reload the web page, hold down the shift key on the keyboard and click the 'reload' or 'refresh' button on the web browser.

possible respondent input option in the HTML form – in this case these are 'radio button' inputs corresponding to the answers 'a', 'b', etc. for each of the twenty questions – is represented by a number, indicating its position within the array of possible inputs. Thus with twenty questions, each allowing for four possible answer choices, there are eighty (twenty times four) possible inputs from which the respondent may select their answers. Since radio buttons are utilised here for the four answer options for each question, only one of these can be selected at any one time. In order to allow respondents to be able to select more than one answer per question, check boxes could be used. These inputs can be represented within the HTML form as "document.myform[1]" (question 1, answer 'a'), "document.myform[2]" (question 1, answer 'b'), "document.myform[3]" (question 1, answer 'c'), and so on. Thus question 8, answer 'c' would be represented as "document.myform[31]", it's position within this array of eighty possible inputs. In the MCQquiz.txt file, the correct answers have all been coded as 'a' (i.e. [1], [5], [9], [13], [17], and so on – try answering 'a' to every question and a score of twenty should be returned). Thus, coding the correct answers is simply a matter of changing the numbers in square brackets in these lines so they refer to the appropriate inputs[4].

7. Having successfully completed the last two steps, you should have an MCQ quiz with customised questions and answers, and the correct answers coded so that scoring functions correctly. If this file is opened up in a web browser it should work as does the example on the MCQ tool page, providing the correct total score when the 'Score' button is clicked[5]. The final stage involves making this MCQ quiz accessible via the world wide web, which requires placing it on a web server. This requires access to space on a web server, either provided by an educational institution or a commercial provider. For example, at the University of Bolton staff are able to place their webpage documents in their own dedicated folder on the University's web server, and these pages are then available at: http://www.bolton.ac.uk/staff/username (where username is replaced with the individual's own personal username). Instructions on how to place files in the relevant web server directory should be requested from your web server administrator, who often will be able to refer you to documentation available online which contains all the necessary information.

Evaluation

The automatic feedback version just described could be useful to provide online practice quizzes, and is fairly easily implemented. For summative assessments, however, answers will need to be stored. The two implementations described below enable this function; the first involves a simple adaption of the technique described above, the second involves a more sophisticated procedure using *CGI scripting*.

---

[4] Essentially, the code checks whether any of these correct answers have been selected, and for each that has adds a value of 1 to the variable x, where x started at zero. In this way x will end up with a value equal to the number of correct answers selected, and thus provide the respondent's total score.

[5] For scoring to work, the browser's 'Internet options' settings need to be configured so that javascript is enabled; for most browsers javascript will be enabled by default. It is also possible that certain security settings may cause the scoring routine to fail, in which case these settings will need to be adjusted as appropriate.

Summative Assessment 1: *Using Email to Store Responses*

Implementation

1. Download the file 'MCQtest-email.txt' from the MCQ tool webpage (refer to steps 1 to 3 above). The HTML code (with one instead of twenty questions) is shown in Table 2. Customise the question content and answer codings as required (as described in steps 5 and 6 above). In coding the correct answers, note that because two additional respondent inputs have been added at the start of the form – asking for the respondent's user ID and name – these are assigned as inputs [1] and [2] respectively; question 1 answer 'a' thus now becomes input [3].

2. To customise the range of IDs which will be accepted, simply insert these in place of "userID1", "userID2", and "userID3". The text within the quotes specifies exactly what the user must enter, and is case sensitive. If you test MCQtest-email.html, inputting userID1, userID2 or userID3 into the ID box will result in "ID verified" being returned when you click the button as instructed. Any other value will lead to "You have not entered a valid ID. Please try again and if the problem persists contact the module tutor" being returned. This message can be altered, as desired, simply by editing the text. As many IDs as required can be included, simply by adding more identical lines of code as appropriate, e.g.

   && (document.myform.ID.value !="userID4")

3. The final change that needs to be made is to specify an email address to which the responses should be sent. This is done in the **action="mailto:** command, which here contains the fictitious address "myemail@myserver.ac.uk". Try substituting this with a real email address (i.e. yours) and submitting the assignment, and it should send an email to that address[6]. The email message will contain the user's data in the following format: ID=userID1&name=&Q1=a&totalscore=1. In this case the user entered 'userID1', did not enter a name, answered 'a' to question 1 (but did not answer any other questions), and obtained a total score of 1. Obviously this data will need some tidying up before entering into a spreadsheet or data analysis package.

   All the above will work by placing the MCQtest-email.html file on a local computer, as long as an email client (e.g. Outlook, Pegasus, etc.) is correctly installed and configured, and there is an Internet connection active (needed in order for the email to send). To make the MCQ assessment accessible via the world wide web, it will need to be placed on a web server, as described in step 7 above.

Evaluation

The above implementation provides a relatively straightforward and effective way of administering MCQ assessments which includes ID-checking, automated scoring, and storing of respondents' answers within the body of an email message sent to an appropriate email account.

---

[6] The email client may first open up a mail message window which allows the user to view the contents of the email prior to sending it, which has obvious implications.

However, there are some drawbacks to this approach. Firstly, the data is not stored in an easily readable format, and would need to be tidied up somewhat before being imported into a data processing package. Secondly, and more crucially, some mailers will open up a window displaying the contents of the email message to be sent – in this case, the respondent's answers and total score – thus allowing the user to view the contents of the email and consider whether they really want to send it. Obviously, this is problematic in this context, and may lead to attempts to either edit the score value directly (which of course can be detected if the tutor compares the total score against responses to individual questions), or to make multiple attempts, perhaps by trial and error, until a satisfactory score is achieved and sent. This makes the approach unworkable for summative assessments unless the tutor either is able to maintain control over the range of browsers/email clients[7] which students may use (e.g. by requiring the assessment to be taken in-class), or removes the automatic scoring function (which can be easily done[8]). Having the scoring key within the HTML form itself is less than ideal anyway, whether or not the mailer displays answers before sending, as respondents with programming experience may be able to detect the correct answers by viewing the source code (done by clicking 'Page Source' under the browser's 'View' menu). A final issue concerns the need for the respondent to have an email client correctly installed and configured on their computer in order for the answers to be sent by email; while most users do have this, some (e.g. those relying on 'webmail') may not.

Given the above reliability and validity issues, an alternative implementation is now described which makes use of a *CGI script* to process and save incoming data from respondents. Since this implementation moves the scoring process from the HTML form to the CGI script, it increases the level of security over the correct answers, while maintaining the convenience of using automated scoring. It also increases reliability by not relying on the respondent having an email client appropriately configured on their machine. Hence this approach is strongly recommended whenever possible.

---

[7] On testing, Mozilla's (version 1.6) email client did open up an email window displaying the answers and total score, when configured with a POP3 mail server account. When configured for an IMAP mail server account, this same mailer failed to include any data in the body of the email message, with only an empty email message being sent. Outlook express's email client included the answers and total score as a file attachment to an email message, which could be opened and viewed from within the user's outbox prior to sending (thus generating the same problem as described previously of the user being able to view their total score prior to sending).

[8] To do this, remove all the code from **function score()** down to and including the **}** just before **/script**. Then edit the line below **/script**, i.e. **<form method=post name=myform action="mailto:myemail@myserver.ac.uk" onsubmit="return score()">** so that **onsubmit="return score()">** becomes *onsubmit="return checkform()">*. Finally, in the third line from the bottom of the code, i.e. **<input type=submit value="submit assignment"><input type=hidden name=totalscore value="">**, remove the latter part within < > so that this line becomes: *<input type=submit value="submit assignment">*. Now open this new file as a webpage (html document) and try completing and sending the MCQ assessment; you will notice that the "totalscore=" line in the email message no longer appears.

Summative Assessment 2: *Using a CGI Script to Process Submitted Data*

This implementation requires access to a server space which allows execution of CGI scripts. The CGI script provided here, which is written in Perl, saves the respondent's data to a specified file, in a format[9] ready for import into SPSS or a similar data analysis package, and also sends the data in an easily readable format in the body of an email message to a specified email address. Most modern servers can run Perl CGI scripts, including the popular open-source UNIX-based 'Apache', and many Windows-based servers.

Implementation

1. Save the file 'MCQtest-cgi.txt' from the MCQ tool webpage, and make the necessary changes to the question content, and ID codes, as described in steps 1 and 2 above. Note that the scoring function has been omitted in this implementation, so there is no need to specify any correct answer codings here. Other than this, the HTML code is very similar to that of 'MCQtest-email.txt' (the condensed version of which is shown in Table 2).

2. Next, the HTML code needs to be edited to specify the location and name of the CGI script which will process the data when the respondent submits their answers. This is done in the **action="** command. In the above implementation this command specified an email address; here that is replaced with the location and name of the CGI script. The code currently specifies the location of the CGI script as a (fictitious) web address: "http://www.servername.com/cgi-bin/MCQscript.pl"[10]. This needs to be replaced with the location of the CGI script on your own server[11]. The appropriate directory in which to place CGI scripts, and the full path name to the script, should be obtained from your web server administrator.

3. The next step involves downloading and customising the CGI script 'MCQscript.pl' from the MCQ tool website (following the same procedure as for 'MCQtest-cgi.txt'). Note that the CGI script has been given the file extension .pl, but you should save it initially as a text file, i.e. 'MCQscript.txt', so that it can be opened and edited within a text editor (it will need to be re-named with the extension .pl after it has been edited). The script should be customised according to the instructions in steps 4 to 6 below. The full code for MCQtest.pl is shown in Table 3.

4. First, the code needs to be edited to specify the appropriate mail server, currently "mail.bolton.ac.uk"; ask your web server administrator for this (which will typically be of the form 'mail.servername.ac.uk'). In the line below, a recipient's

---

[9] CSV, or 'comma separated values'.

[10] That is, the address of the server (http://www.servername.com), and the directory on that server (cgi-bin) in which the script (MCQscript.pl) resides.

[11] Typically, a CGI script will need to be placed in a special directory called 'bin' or 'cgi-bin', in which CGI scripts are allowed to execute. If the HTML form is also placed in this same directory then all that needs to go in the *action=*" command is the name of the script itself (e.g. 'MCQscript.pl') and the HTML form will find it. Otherwise the appropriate full path to the script will need to be specified.

email address (who you want the data to be sent to) should be provided (currently "c.hewson@bolton.ac.uk"). Below, put the name of the data file where you want the data to be stored (here "MCQtest.dat"). If you do not require data to be sent by email then simply leave the email recipient field blank and no email will be sent[12]. The data file itself, which can be simply created as an empty text file, will need to be placed somewhere on the web server; if this is in the same directory as the CGI script then just the filename needs to be specified, i.e. 'MCQtest.dat' in the current example. However, if it is in a different directory then the full path will need to be specified (if unsure, your web administrator will be able to provide this).

5. Next, the *$mail_sender* value (in this case "MCQ assignment") should be changed to whatever name you wish to appear as the sender of the email message in which data is sent. You will also need to change the *$debrief_url* value (in this case "http://www.myserver.ac.uk/debrief.html"), which is the address of a webpage to which the user is directed after the answers have been submitted. This could be any webpage address you wish, e.g. a university homepage, or one which contains a simple text message stating something such as "thank you, your answers have been successfully sent". The file 'debrief.html', which has been provided for download along with the other MCQ tool files, does just this. This file can be placed on your web server, and it's location specified as the *$debrief_url* address.

6. Finally, the correct answers for the questions need to be specified, simply by changing those that are already coded, as necessary (e.g. **"Q1" => "d"**, etc.). The rest of the script does not need to be modified (though it can, by anyone with some knowledge of Perl).

7. In order to make the MCQ assessment available online, the HTML form, CGI script and debrief page need to be placed on a web server, and their permissions set appropriately. The HTML form (MCQtest-cgi.html) does not require any special permissions, and should be placed on a web server as described in the previous two implementations. The CGI script (MCQtest.pl: note that this file must now be renamed to have the .pl extension for it to run correctly on the web server) should be placed in the appropriate directory as specified by your web server administrator, checking that this location has been correctly specified in the HTML form code (as described in step 2 above). Permissions on this file must be set to allow execute access to all users. The data file also needs to be placed somewhere on the server (refer to step 4 above). Permissions need to be set to give write access for all users (so that data may be saved). Your web server administrator will be able to explain how to set permissions appropriately.

With all the above in place, the HTML form should now submit the data to the CGI script, which will process it and save it to the specified data file, as well as send it in the body of an email message to the address specified. Try this out by filling in the MCQ assessment and submitting data. If the process is successful the data file should now contain a line consisting of the respondent's answers, which looks something like this:

Mon Mar 28 18:52:24 2005,193.63.49.30,userID1,Hammy,a,b,a,a,d,b,a,,,,,,,,,,,,,0

[12] Note, however, that if the mail server and data file fields are left blank an error will be returned and the script will fail to execute correctly.

This data is in CSV (comma separated values) format and can therefore be imported directly into a data analysis package such as SPSS.

Check the email account to which data was also sent, and a message similar to that displayed in Table 4 should have been received. This user (Hammy) answered questions one to seven and left the remainder blank, obtaining a total score of zero.

Evaluation

The above approach has a number of advantages, as have been outlined above. It does, however, require greater levels of technological expertise and resources, compared with both of the previous approaches described in this document.

Further Refinements

Providing Immediate Feedback and Storing Responses

It may, of course, be useful to provide immediate feedback to students *and* store their responses. This can be easily achieved using the system described previously which makes use of both the HTML form (MCQtest-cgi.html) and the CGI script (MCQscript.pl). The CGI script simply needs to be modified slightly to post back a response to the respondent when they submit their data, which tells them their total score. To do this simply edit the final lines of code in the CGI script which print information back to the user and redirect their browser to another web page. For example, editing the lines which include the text stating "Your answers are being sent" as shown below will provide a receipt for the student which states their name, ID, date and time of submission, and total score:

print "<font face=\"Verdana, Arial, Helvetica, sans-serif\" color=\"#000000\"><b>Your MCQ assignment has been received. You can print out this receipt for your records. <P>Name: $FORM{name}
Student number: $FORM{ID}
Date received: $date_time.<P>
You have scored $score correct out of 20.<p>\n";

Of course, the total score may or may not be included here, as appropriate. Just delete the final sentence "You have scored $score correct out of 20." in the above code to not reveal the score. Also, it is highly unlikely, but not impossible, that the answers will look as though they have been sent successfully while in fact some server error has occurred which has caused them to be lost. Thus it might be wise to replace "your assignment has been received", with "your assignment has been sent". If using the code suggested above, the line: **print "<meta http-equiv=Refresh\ content=\"3;url=$debrief_url\">\n";** should be deleted (or commented out by placing a # in front of it) so that the student's browser is not automatically redirected to another webpage, else they may miss digesting the information provided and most certainly will not have time to print out their receipt.

Conclusions, Usage, Limitations

The tool described here, in it's various forms, provides scope for administering online MCQ assessments to students, and scoring and/or storing responses automatically. This may have the benefits of time- and cost- savings, as well as convenience for students, who can access and submit the assessment from a range of locations[13]. In it's simplest form, the formative version can be used as an online 'quiz' which provides automatic feedback and is easy to implement. If storing of responses is required, this can be done without any need for CGI scripting, using the 'mailto:' command, though there are various drawbacks to this approach, as discussed above. However, if access to a server which allows CGI scripts to run is an option, these problems can be overcome, using the tools and procedures described in the third implementation above.

There is scope for developing the tool in various ways; for example one possible issue is the way IDs are stored within the HTML form itself, which leaves open the possibility of respondents gaining access to this information and selecting and using any of these codes to be able to submit the assignment (though it is difficult to think of a reason why a student might want to adopt this strategy). Nevertheless, the ID-checking function could be moved into the CGI-script with a little extra programming work; ID codes could be stored within the script itself or in a database which the script is able to read. One may also want to include further text input boxes within the HTML form, e.g. asking for an email address to be provided, or change the layout, font size, number of questions, response format (radio buttons, check boxes, text input boxes), and so on. Users already familiar with HTML will be able to customise the code to their liking; those new to HTML may find one of the many introductory guides available useful, see for example:

> Elizabeth Castro: HTML 4 the World Wide Web, 4th Edition: Visual Quick Start Guide, 1999.
>
> A Quick Guide to HTML, Cardiff University, Available: http://www.cs.cf.ac.uk/Dave/PERL/node249.html.

To conclude, the aim here has been to provide a simple yet effective resource which allows users with minimal levels of computer expertise to implement MCQ assessments online. A further aim has been to make available fully customisable, open source code, so that individual users with moderate levels of computer programming experience may adapt the tool to suit their own needs.

[13] While most students these days have ready access to a computer, some may not.

```
<html><head><basefont size=3>
<title>Mock-up Version of online MCQ assignment</title>

<script language=javascript>
   function score()
{
            var x = 0
            var y = 20

            if(document.myform[1].checked==true){x = x+1}

{           alert("Your score is " +x+ " out of " +y)}

document.myform.totalscore.value = x
            return true;
}
</script>

<form method=post name=myform onsubmit="score()">
<input type="hidden">

<P> <center><B>Multiple Choice Question Quiz<BR>
<BR><P>

</B></center>
Answer the questions below by clicking next to the correct answer. Press 'Score' below to find out how
many answers

you got correct.
<BR><P>

1. Insert question one here: <P>

a) insert answer option 'a' here          <input type=radio name="Q1" value="a"><br>
b) insert answer option 'b' here          <input type=radio name="Q1" value="b"><br>
c) insert answer option 'c' here          <input type=radio name="Q1" value="c"><br>
d) insert answer option 'd' here          <input type=radio name="Q1" value="d"><br>

<BR><P>
<P>
<input type=button value="Score" OnClick="score()">
</form>
</html>
```

Table 1: MCQquiz.txt (shortened HTML code)

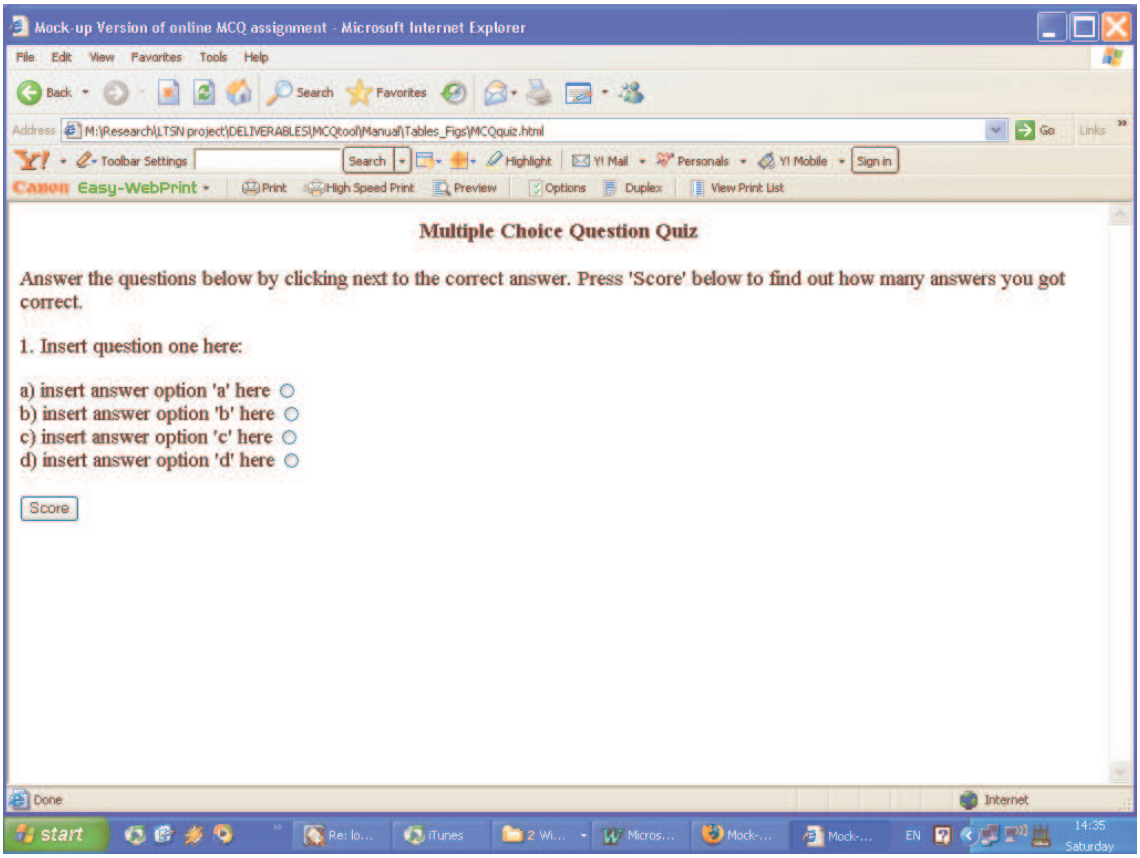

Figure 1: MCQquiz.html (shortened code) as displayed in a web browser

```
<html><head><basefont size=3>
<title>MCQ Assignment</title>

<script language=javascript>
function checkform()
{
	if ((document.myform.ID.value !="userID1")
&& (document.myform.ID.value !="userID2")
&& (document.myform.ID.value !="userID3")
		)
	{
		// something is wrong
		alert('You have not entered a valid ID. Please try again and if the problem persists
contact the module tutor');
		return false;
	}
else
{alert('ID verified')
}
	return true;
}
function score() {
	var x = 0
	var y = 3

	if(document.myform[3].checked==true){x = x+1}

document.myform.totalscore.value = x
	return checkform()
}
</script>
<form method=post name=myform action="mailto:myemail@myserver.ac.uk" onsubmit="return
score()">

<P> <center><B>MCQ Assignment<BR>
Date due: 24th March, 2005<P>
</B></center>
<P>Please type in your student ID number and click on 'Verify' <P> <input type="text" name="ID">
<input type="button" value="Verify" onclick="return checkform()">
<P><BR>
Name: <input type="text" name="name">
<BR><BR><P>

Please answer the questions below by clicking next to the correct answer. Answer all questions. <B>
Do not press the 'Enter' (Return) key on your keyboard while still completing this assignment as this
will cause your answers to be automatically sent before completion.</B>

<P> When you are sure you have completed all the questions please press the <B>'send
assignment'</B> button to send your completed answers. <BR><BR><P>

1. Insert question one here: <P>

a) insert answer option 'a' here		<input type=radio name="Q1" value="a"><br>
b) insert answer option 'b' here		<input type=radio name="Q1" value="b"><br>
c) insert answer option 'c' here		<input type=radio name="Q1" value="c"><br>
d) insert answer option 'd' here		<input type=radio name="Q1" value="d"><br>
<BR><P>
THAT IS THE END OF THE ASSIGNMENT. <P>
<P>
<input type=submit value="submit assignment"><input type=hidden name=totalscore value="">
</form>
</html>
```

Table 2: MCQtest-email.txt (shortened HTML code)

```
#!/usr/bin/perl
#use CGI::Carp qw(fatalsToBrowser);
use Net::SMTP;
#use warnings;

# define parameters for your application
# Mail params
$mail_server    ='mail.bolton.ac.uk';
$mail_recipient ='my.email@myserver.ac.uk';
$data_file      ='MCQtest.dat';


$mail_sender    ='MCQ_assignment_fmethods_sem1_04-05\@bolton.ac.uk';
$debrief_url    ='http://www.servername.ac.uk/username/MCQdebrief.html';

# create array of question names, to specify the output formatting order
@itemnames = (
                "Q1", "Q2", "Q3", "Q4",
                "Q5", "Q6", "Q7", "Q8",
                "Q9", "Q10", "Q11", "Q12",
                "Q13", "Q14", "Q15", "Q16",
                "Q17", "Q18", "Q19", "Q20"
                );
# create hash of correct answers for your questions, for scoring
%itemanswers = (
                    "Q1" => "d", "Q2" => "c", "Q3" => "d", "Q4" => "b",
                    "Q5" => "c", "Q6" => "d", "Q7" => "d", "Q8" => "d",
                    "Q9" => "c", "Q10" => "a", "Q11" => "d", "Q12" => "c",
                    "Q13" => "a", "Q14" => "a", "Q15" => "d", "Q16" => "c",
                    "Q17" => "b", "Q18" => "c", "Q19" => "d", "Q20" => "a"
                    );

# this might work to avoid defining @itemnames above, but it reorders keys
#@itemnames = keys %itemanswers;

# recover time and hostname
$date_time=localtime;
$remote_address=$ENV{'REMOTE_HOST'};
if (!$remote_address) { $remote_address=$ENV{'REMOTE_ADDR'}; }
$forwarded_for=$ENV{'HTTP_X_FORWARDED_FOR'};
if ($forwarded_for) { $remote_address=$forwarded_for; }

# read the input data (method=POST)
read(STDIN, $formdat, $ENV{'CONTENT_LENGTH'});

# split the input data into individual form items
@namevals = split(/&/,$formdat);

# create a hash containing the form data
foreach $pair (@namevals) {
   # separate into name and value
   ($name,$value) = split(/=/,$pair);
   # convert + signs to spaces
   $value =~ tr/+/ /;
   # convert hex pairs (%HH) to ASCII
   $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
   # store vals in a hash called %FORM
   $FORM{$name} = $value;
}
# now, prepare the outputs (file and email)
open(SURVEY_FILE,"+>>$data_file")  or die "$^E";
print SURVEY_FILE  "$date_time,$remote_address,";
```

Table 3: The CGI script ‘MCQtest.pl’

```
# email code
$smtp = Net::SMTP->new($mail_server) || die ^E;
$smtp->mail('web_form@bolton.ac.uk');
$smtp->to($mail_recipient);
$smtp->data();
$smtp->datasend("To: $mail_recipient\n");
$smtp->datasend("From: $mail_sender\n");
$smtp->datasend("\n");
$smtp->datasend("Date\: $date_time \n");
$smtp->datasend("Remote Address\: $remote_address \n");

# print out the ID and name values to file...
print SURVEY_FILE "$FORM{ID},$FORM{name},";

# ...and to email
$smtp->datasend("ID\: $FORM{ID} \n");
$smtp->datasend("Name\: $FORM{name} \n");

# initialise the score
$score = 0;

# write one value (possibly empty) to file for each question item
# even if it doesn't appear in the form submission
# also, do scoring during the same pass
foreach $thename (@itemnames) {

   # score the answer (undef values don't get counted)
   # add 1 point for each correct answer, no penalty for incorrect
   if($itemanswers{$thename} eq $FORM{$thename}) {
         $score++;
   }

   # print comma-delimited value (possibly empty) to file...
   print SURVEY_FILE "$FORM{$thename},";
   # ...and name: value pair to email
   $smtp->datasend("$thename\: $FORM{$thename} \n");
}
# after the raw data, print out the score too
print SURVEY_FILE "$score\n";

$smtp->datasend("Score\: $score \n");

# data processing is over, so close outputs
close (SURVEY_FILE);
$smtp->dataend();
$smtp->quit;

# finally, print message page back to user
print "Content-type: text/html\n\n";
print "<html>\n";
print "<head>\n";
print "<meta http-equiv=Refresh\ content=\"3;url=$debrief_url\">\n";
print "<title>User Survey</title>\n";
print "</head>";
print "<body bgcolor=\"white\">";
print "<center>\n";
print "<br><br><br>\n";
print "<font face=\"Verdana, Arial, Helvetica, sans-serif\" color=\"#000000\"><b>Your answers are
being sent<P><BR>Please wait... <br><p>\n";
print "</center>";
print "</body>\n";
print "</html>\n";
```

Table 3 (cont): The CGI script ‘MCQtest.pl’

Date: Mon Mar 28 18:52:24 2005
Remote Address: 193.63.49.30
ID: userID1
Name: Hammy
Q1: a
Q2: b
Q3: a
Q4: a
Q5: d
Q6: b
Q7: a
Q8:
Q9:
Q10:
Q11:
Q12:
Q13:
Q14:
Q15:
Q16:
Q17:
Q18:
Q19:
Q20:
Score: 0

Table 4: Contents of the email message sent by the CGI script (MCQtest.pl)