DYNAMIC WEBPAGE DESIGN AND IMPLEMENTATION

Submitted to Committee Members

Dr. Kai-Hsiung Chang (Chair)
Dr. Alvin S. Lim
Dr. Gerry V. Dozier

by

Weiqin Ye

in Partial Fulfillment of the
Requirements for the
Degree of
Master of Computer Science and Engineering

March 19, 1999
DYNAMIC WEBPAGE DESIGN AND IMPLEMENTATION

Submitted to Committee Members

Dr. Kai-Hsiung Chang (Chair)
Dr. Alvin S. Lim
Dr. Gerry V. Dozier

by

Weiqin Ye

in Partial Fulfillment of the
Requirements for the
Degree of
Master of Computer Science and Engineering

March 19, 1999
ABSTRACT

DYNAMIC WEBPAGE DESIGN AND IMPLEMENTATION

Weiqin Ye

Master of Computer Science and Engineering, June, 1999

Directed by Dr. Kai-Hsiung Chang

Internet and World Wide Web browsers are not new to most people nowadays. Many wondrous things can be found in the World Wide Web. Hypertext documents with hyperlinks, colorful graphics, and multimedia technology allow people to tour around the world on the Internet by just clicking mouse buttons.

Web technology can provide front ends for users to access commercial database applications as well. Customized application programs can be wrapped around Web server, and a user just has to click buttons to activate applications. A user can also collaborate with others on work-related projects and share scientific data and information.

This project studies how to develop dynamic web pages, data communication and interface between users and a database server. Also a concrete example, Online Test System, will be provided to demonstrate how to implement useful web features using Miva, CGI programming, and Oracle.

Using HTML, Perl and Oracle to implement dynamic web page is more complicated and difficult than using Miva Htmlscript language. It involves several different languages and commercial database systems.
# TABLE OF CONTENTS

## CHAPTER 1 INTRODUCTION

1.1. Introduction to the World Wide Web  
1.2. HTML and dynamic web page  
1.3. Dynamic web page and Database package  

## CHAPTER 2 DYNAMIC WEB PAGE IMPLEMENTED BY USING CGI SCRIPT LANGUAGE AND ORACLE DATABASE

2.1. Form implementation  
2.2. CGI language and CGI programming  
2.3. Database, Oracle DBMS, and database programming  
2.3.1. Data model and relational data model  
2.3.2. Oracle RDBMS and SQL language  
2.3.3. Database application programming  
2.4. A sample dynamic web page  
2.4.1. HTML files  
2.4.2. CGI program and variable passing between web-page and external program  
2.4.2.1. C shell program
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.2.2. Input stream encoding and decoding</td>
<td>20</td>
</tr>
<tr>
<td>2.4.2.3. Perl programming</td>
<td>22</td>
</tr>
<tr>
<td>2.4.3. Database</td>
<td>24</td>
</tr>
<tr>
<td>2.4.4. Embedded SQL and Pro*C</td>
<td>25</td>
</tr>
<tr>
<td>CHAPTER 3 DYNAMIC WEB PAGE IMPLEMENTED IN MIVA</td>
<td>27</td>
</tr>
<tr>
<td>3.1. Introduction of the Miva</td>
<td>27</td>
</tr>
<tr>
<td>3.2. Miva tags and attributes</td>
<td>28</td>
</tr>
<tr>
<td>3.3. Miva script variables and variable scope</td>
<td>29</td>
</tr>
<tr>
<td>3.4. Miva script function</td>
<td>30</td>
</tr>
<tr>
<td>3.5. Miva database and data file</td>
<td>31</td>
</tr>
<tr>
<td>3.6. A sample dynamic web page</td>
<td>32</td>
</tr>
<tr>
<td>3.6.1. The procedure of the program</td>
<td>32</td>
</tr>
<tr>
<td>3.6.2. Data input and output</td>
<td>34</td>
</tr>
<tr>
<td>3.6.3. Running Miva program in Auburn University</td>
<td>35</td>
</tr>
<tr>
<td>CHAPTER 4 DISCUSSION AND CONCLUSION</td>
<td>36</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>37</td>
</tr>
<tr>
<td>APPENDIXES</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1. Online Test System’s cover page 14
Figure 2. Test subject selection page 15
Figure 3. Questions answering page 17
Figure 4. Test grade page 18
CHAPTER 1 INTRODUCTION

Netscape, Mosaic, or other World Wide Web browsers are very familiar to most people. Many wondrous things can be found in the World Wide Web. Hypertext documents with hyperlinks, colorful graphics, and multimedia technology allow people to tour around the world on the Internet. Just following your interests and impulses by clicking mouse buttons, vast information resources are at your fingertips. With a Web browser's intuitive interface, all these resources are accessible even to novice computer users.

By now, it is hard to imagine how someone could have missed learning at least something about the World Wide Web and Internet. People can see Web page's address everywhere, such as in television commercials, newspapers, magazines, even hear them from broadcasting. With Web pages, universities, businesses, and individuals can distribute information about their organizations and/or people. People can shop on the Internet for merchandises, read newspapers and magazines online, or download free software. Even students can fill in the application form online to apply for colleges or universities these days. Web technology can provide front ends for users to access commercial database applications. Customized application programs can be wrapped around Web server, a user just has to click buttons to activate applications. A user can also collaborate with others on work-related projects and share scientific data and information.
This project studies how to develop dynamic web pages, data communication and interface between users and a database server. Also a concrete example, Online Test System, will be provided to demonstrate how to implement useful web features using Miva, CGI programming, and Oracle. Comparison between Miva and CGI will also be provided.

1.1. Introduction to the World Wide Web

In 1993, Tim Berners-Lee and other researchers at the European Particle Physics Lab (CERN) in Geneva, Switzerland, developed a means of sharing data among their colleagues using something they called hypertext [1]. CERN users could view documents on their computer screens using new browser software. Special codes embedded in these electronic documents allowed users to jump from one document to another on screen just by selecting a hyperlink. Internet capabilities were built into these browsers. CERN’s breakthrough work is the basis of today’s World Wide Web. Its web server and browser software (now being maintained by the World Wide Web Consortium with the current versions) were the first of their kinds.

Marc Andreesen [2], a graduate student working part time at the University of Illinois National Center for Supercomputer Applications, picked up CERN’s work and turned it into what would become today’s NCSA Mosaic, the first graphical Web Browser with the point-and-click applications. Just as Mosaic descended from the work at CERN, all subsequent graphical Web browsers come from this common ancestor.
The World Wide Web is a hypertext information system on the Internet. It operates based on the Client/Server model. A web client is the Web browser, such as Netscape or Internet Explorer. The Web server is any machine with the Web server software running on it.

When a hyperlink is clicked, the browser reads a document written in HTML and displays it, interpreting all the markup codes in the document. Then, the browser uses the HyperText Transfer Protocol (HTTP) to send a network request to a Web server to access the new document or service specified by the hyperlink. The Web server responds to the request with the document or other data. Finally, the browser software then reads and interprets that information and presents it in the correct format.

1.2. HTML and dynamic web page

The programming language for the World Wide Web is the Hypertext Markup Language (HTML). The Web actually contains a huge amount of interconnected hypertext documents on the Internet written in HTML. A hypertext document is a document that contains hot links to other documents.

Dynamic web page is a web page in which user can enter text information, e.g., name, email address, or a page that contains choices to check off and options to select from. It is almost like a form in real life. While a paper form must be turned in or mailed, a Web form is instantly submitted. The power of dynamic web page is that it can accept and organize the user's input. Businesses and organizations use forms for conducting surveys, allowing online shopping and product ordering, as well as service registration.
Many personal home pages have guest books that users can “sign” by simply filling in a form.

In technical terms, a web form is the interactive front end to the Web. It is where the user enters text and makes selections, while the back end, the actual processing of data, is done on the Web server by CGI scripts.

1.3. Dynamic web page and Database package

Users can perform queries and data entry using fill-in web form, in which users can enter query keywords or other search criteria through menu selections, click buttons, free-form text blocks, or fill-in-the-blanks. CGI-bin scripts take the information entered in the form and wrap it up into valid SQL queries or data-entry updates. They then pass it off to the database back end. The same CGI-bin scripts receive the results back from the database engine after processing. They format the report in HTML and pass it back to the customer’s Web browser for display.

Besides using CGI script to access the commercial database package, Miva (HTML script 3.0) also has a build-in database, which is based on the xBase file format. The index structure has been designed with data portability, multi-user, and performance as the primary goals. Miva can also use file input and output to handle simple database application.
CHAPTER 2

DYNAMIC WEB PAGE IMPLEMENTATION BY USING CGI SCRIPT

LANGUAGE AND ORACLE DATABASE

There are several issues that must be considered to implement a dynamic Web page, such as form implementation, CGI programming language choice, database system and database application programming.

2.1. Form implementation

In HTML, the FORM tag specifies a fill-out form within an html document. More than one fill-out form can exist in a single document, but forms cannot be nested. For example,

<FORM METHOD = post ACTION = http://roadrunner.eng.auburn.edu/cgi-bin/weiqye/star1.sh >

</FORM>
The FORM tag has three attributes: ACTION, METHOD, and ENCTYPE. ACTION is the URL of the query server to which the form contents will be submitted. "star1.sh" is a shell script file, which will set the database path and call Perl file. The query server URL is http://roadrunner.eng.auburn. There are two METHODS, GET and POST. Which method to use depends on how the particular server works. ENCTYPE specifies the encoding for the fill-out form contents. This attribute applies only if a method is set to post.

Inside a FORM, anything except another FORM can exist. Specifically, INPUT, SELECT, and TEXTAREA tags are used to specify interface elements within the form.

The SELECT tag has both opening and closing tags. Inside a SELECT tag, only a sequence of OPTION tags, each followed by an arbitrary amount of plain text, is allowed.

The TEXTAREA tag can be used to place a multi-line text entry field with optional default contents in a fill-out form. TEXTAREA fields automatically would have scrollbars, any amount of text can be entered in them.

The INPUT type might be one of “text”, “password”, “checkbox”, “radio”, “submit”, and “reset”. A “text” element is a text entry field. A “password” element is also a text entry field except the entered characters are echoed as asterisks. A “checkbox” element is a single toggle button, on or off. A “radio” element is a single toggle button, on or off, other toggles with the same NAME are grouped into “one of many” behavior. A “reset” element is a pushbutton that causes the various input elements in the form to be reset to their default values.

A “submit” element is also a pushbutton that causes the current form to be packaged into a query URL and sent to a remote server. It is necessary that all forms
should have a "submit" element, except those containing only a single INPUT element of type TEXT, that a RETURN in the text entry area will submit the form. If at least one IMAGE type INPUT element is in the form, a click on the image will also submit the form.

2.2. CGI language and CGI programming

The Common Gateway Interface (CGI) has emerged as the first way to present dynamically generated information on the World Wide Web. CGI allows computers to generate Web pages instantly at the user's request rather than being written by someone in advance. CGI is the part of the Web server that can communicate with other programs running on the server. With CGI, the Web server can call up a program, while passing user-specific data to the program (such as what host the user is connecting from, or input the user has supplied using HTML form syntax). The program then processes that data, passes results to the server, and the server passes the results back to the Web browser.

Any language could be a CGI programming language, although certain languages are more suited for CGI programming than others. The following features must be considered when choosing a language [3]:

- Ease of text manipulation
- Ability to interface with other software libraries and utilities such as database engines or graphic manipulation libraries
- Ability to access environment variables (in UNIX)
Some of the more popular languages for CGI programming include AppleScript, C/C++, C shell, Perl, TCL and Visual Basic. Perl is by far the most widely used language for CGI programming! It contains many powerful features, and is very easy for the novice programmer to learn.

Perl's author, Larry Wall, describes Perl this way[^4]:

Perl is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). It combines (in the author's opinion, anyway) some of the best features of C, sed, awk, and sh, so people familiar with those languages should have little difficulty with it. Expression syntax corresponds quite closely to C expression syntax.

Unlike most Unix utilities, Perl does not arbitrarily limit the data size—if there is enough memory, Perl can slurp in your whole file as a single string. The depth of recursion is unlimited. Perl uses sophisticated pattern matching techniques to scan large amounts of data very quickly. Although optimized for scanning text, Perl can also deal with binary data, and can make dbm files look like associative arrays (where dbm is available). Setuid Perl scripts are safer than C programs through a data-flow tracing mechanism which prevents many stupid security holes[^4].
2.3. Database, Oracle DBMS, and database programming

Databases and database technology are very important to computer application. From large tracking databases, such as airline reservation systems to a child’s baseball card collection, database systems are used to store and distribute information. Large database systems could be run only on large mainframe-based computers few years ago. These machines are expensive to design, purchase, and maintain. Due to the increased power of personal computer hardware, critical database information can be stored on a relatively inexpensive stand-alone server. Programmers now can design software to maintain and distribute data quickly and inexpensively.

A database is a collection of related data, which has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the mini-world.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has as intended group of users and some preconceived applications in which these users are interested.

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is hence a general-purpose software system that facilitates the processes of defining, constructing, and manipulating database for various applications.
2.3.1. Data model and relational data model

A data model is a set of concepts that can be used to describe the structure of a database. By structure of the database, it means the data types, relationships, and constraints that should hold for the data. Most data models also include a set of basic operations for specifying retrievals and updates on the database.

The most popular data storage model is the relational database. Relational Database Management System (RDBMS) uses the mathematical concepts of relational algebra to break data down into sets and relate common subsets. Under the relational model, data is separated into sets that resemble a table structure. This table structure consists of individual data elements called columns or fields. A single set of a group of fields is known as a record or row.

The relational database model gives the database designer a great deal of flexibility in describing the relationships between the data elements. Data can be stored in separate tables to reduce repetition, quickly retrieved from different sets (tables), and returned to the user or program as one “joined” collection of data.

2.3.2. Oracle RDBMS and SQL language

Oracle is the most widely used database in the world. It runs on every kind of computer, from PCs and Macintoshses to minicomputers and giant mainframes. Oracle is a relational database.
The information stored in Oracle is kept in tables --- much like a weather table from the daily newspaper. It has vertical columns, horizontal rows and the table name. These are the three major characteristics of most tables. There can be many separate and independent tables in the database. Tables can be related to each other if they each have a column with a common type of information.

Structured Query Language (SQL) is a standard language used to manipulate and retrieve data from the relational databases. Through SQL, a programmer or database administrator can do the following:

- Modify a database's structure
- Change system security settings
- Add user permissions on databases or tables
- Query a database for information
- Update the contents of a database

A commonly used statement in SQL is the SELECT statement. This statement retrieves data from the database and returns this data to the user. In addition to the SELECT statement, SQL provides statements for creating new databases, tables, fields, and indexes as well as statements for inserting and deleting records.

2.3.3. Database application programming

Users can type an SQL query into the database system (a query interface), and then the system displays the resulting answer on the user's terminal screen. But sometimes queries, in fact, are posed through an application program interface. Here the
application program means some higher-level language such as C or PASCAL or sometimes a special type of *forms language* provided with the DBMS. When the program is executed, it interacts with a user at the terminal (there may be many users executing the program simultaneously) usually by displaying a menu form. The terminal user performs some menu selections, after which the program may respond by executing one or more SQL queries posed by the menu selection. The terminal user through selections from one terminal form or another controls all activity.

An SQL query (or other type of SQL statement) can be executed inside a program. However, an SQL query is not a valid statement in most types of higher-level language such as C or PASCAL, so a special form is used to designate such SQL statements. The program is then run through a preprocessor, where these special forms are turned into valid calls to C functions in the database function library, and then the C compiler produces a working program.

The practice of placing SQL statements inside a higher-level language program is known as Embedded SQL programs. A feature known as program-data independence is a very important issue. The program logic does not have to keep track of modes of access, positions in files, record structure, and so on, all of this detail is handled by the DBMS. The program logic can remain unaware of this, referencing only the structure of the tables and the column values they contain.
2.4. A sample dynamic web page

Online Test System is a sample program to show how dynamic web pages are implemented by using HTML, CGI program, and Oracle RDBMS.

This program could be used by instructors to post exams or reviewing materials online. Different exam questions can be put in a database. Users can select the subject of the exam, query the database to display the exam questions. Then the users can do the test online, and submit the answers. Online Test System can also give user the answers and grade of the test, and enter the user’s name, ID, and grade into the database.

The sample test system designed for this project is for CSE100, specifically for the Microsoft Office 97 package. There are three subjects in it --- Word, Excel, and Access. First, users need to have a user ID and password to access this online test system. They then select the subject. Third, they take the test, and submit the answers of that test. Finally, the online test system will give the users correct answers and grade for the test.

Figures 1 to 4 show the screen displays. Figure 1 shows the first page of the Online Test System, which is a login page. The user ID and password text fields allow the user to access the Online Test System. After the user enters the correct user ID and password, and clicks the submit button, he/she can access to the test system. If the user enters the wrong password, it will display an error message.

Figure 2 shows the choice page of the Online Test System, the user can select the subject of the test. In this example, the system is designed for CSE100, which is about Microsoft Office 97 package. There are three radio buttons, Word, Excel and Access, one submit button and one reset button. After the user selects one subject and submits,
CSE100 Introduction to Microsoft Office 97

On Line Test

Please enter your user ID and password

UserID: [Redacted]
Password: [Redacted]

Submit  Reset

Figure 1. Online Test System's cover page
CSE100 Introduction to Microsoft Office 97

On Line Test

Please select the test subject

- Word
- Excel
- Access

Run test Reset

Figure 2. Test subject selection page
the system will display the questions and four answer choices for each question (see Figure 3).

Figure 4 shows the last page of the Online Test System, it display the correct answers and grade of the test. The Online Test System will also use user’s ID to track the user’s grade.

The program languages used here are HTML, C Shell script language, Perl, and Pro*C. The database used in this project is Oracle 7 RDBMS.

2.4.1. HTML files

The first page of the Online Test System is implemented directly by HTML fill-in form. A segment of the code looks like follow:

```html
<FORM METHOD = POST ACTION = "http://roadrunner.eng.auburn.edu/cgi-bin/weiqye/start.sh">
     UserID: <input type = “text” name = “userid” size=”10”><br>
     Password: <input type = “password” name = “pass” size = “8”><br>
     <input type = “submit” value = “Submit” >
     <input type = “reset” value = “Reset” >
</FORM>
```

After the user enters the user ID and password and clicks the submit button, the form collects the variable’s values that the user has entered, and calls the shell script
Figure 3. Question answering page
Figure 4. Test grade page
program (\texttt{start.sh}). The C shell program is on the \textit{"roadrunner"} CGI test server in the Auburn University Engineering Network, and \texttt{start.sh} C shell program will call the Perl program (\texttt{project.pl}) to activate the external executable C program (\texttt{pass}).

The C program will test if the user has entered the correct password. If yes, it will display the second page (see Figure 2) using \textit{"printf"} statement. If not, it will display the error message page.

Most forms will send their data using the POST method. POST is more secure than GET, since the data is not sent along with the URL, and also more data can be sent with POST. The browser, web server, or proxy server may cache GET queries, but POSTed data is sent each time. However, since data posted via most forms is often more complex than a single word or two, decoding posted data takes a little more effort.

If GET method is used, the input values from the form are sent as part of the URL, and saved in the QUERY\_STRING environment variable after the question mark in the URL itself, like

\begin{verbatim}
\end{verbatim}

here \texttt{test1} is an input variable from the form. The POST method and data decoding will be explained in next section.
2.4.2. CGI program and variable passing between web-page and external program

2.4.2.1. C shell program

The shell script program file sets the database environment variable, Pro*C preprocessor path, and calls the Perl program file. Perl program decodes the query information and passes variables to the external program.

The segment of the shell program file provides such services follows:

```bash
... some code ....
setenv TWO_TASK "T:oracle7csedb"
set path = ($path /opt/SUNWspro/bin)
set path = ($path /opt/oracle7/product/7.2.2/bin)
set path = ($path /opt/oracle7/product/7.2.2/proc/bin)
set path = ($path /usr/ucb)
set path = ($path /usr/ccs/bin)
set path = ($path /ccs/bin)
/home/cse_h2/weiqye/project/Proj-ye/proj.pl
... some code ...
```

Because the FORM selects the POST method, the variables are passed as standard input into the variable $buffer of the Perl program. Then Perl decodes the input stream and puts them into a hash %FORM, whose values are passed as command line variables into an external program by Perl's system( ) function.
2.4.2.2. Input stream encoding and decoding

When a web server sends form data to the Perl script, it encodes the data being sent. Alphanumeric characters are sent as themselves; spaces are converted to plus signs (+); other characters – like tabs, quotes, etc. – are converted to “%HH” – a percent sign and two hexadecimal digits representing the ASCII code of the character. This is called URL encoding.

Perl has powerful pattern matching and replacement capabilities along with substitute and translate commands. It can match the most complex patterns in a string, using regular expressions. It is also quite capable of the simplest replacements. The basic syntax for substitutions is:

```perl
$mystring =~ s/pattern/replacement/;
```

This command substitutes “pattern” for ‘replacement’ in the scalar variable “$mystring”. The operator is a =~ (an equal sign followed by a tilde) – this is a special operator for Perl, telling it that it is about to do a pattern matching or replacement.

Two changes must be made to the data sent from the form:

```perl
$value =~ tr/+//;
$value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
```
The first line translates every instance of “+” back to a space. The second line is a rather complicated regular expression that substitutes every %HH hex pair back to its equivalent ASCII character, using the pack( ) function.

After decoding the input stream, the values of the hash $FORM are passed as command line variables into external program with Perl’s system( ) function.

2.4.2.3. Perl programming

Following is an example of Perl program named proj.pl, which passes the FORM variables into program executable file “proj”.

```perl
#!/opt/local/perl5
print "Content-type:text/html\n\n";
print "<head><title> Project – Online test system </title></head>";
read(STDIN, $buffer, $ENV{’CONTENT_LENGTH’});
@pairs = split(/&/, $buffer);
foreach $pair (@pairs){
    ($name, $value) = split(/=/, $pair);
    $value =~ s/+/ /;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])pack(’C’, hex($1))/g;
    $value =~ s/~/!/~/g;
    $FORM{$name} = $value;
}
system("home/cse_h2/weiqye/project/Proj-ye/proj $FORM{’action’} ");
```
Statement 1 in the above Perl program tells the server that this is a Perl script, and where to fine the Perl interpreter. Every Perl program has to have this statement in the first line.

Statement 2 generates an HTML page, it should be included somewhere in the script, before anything else being printed out. This is a MIME header that tells the receiving web browser what sort of data it is going to get; in this case, an HTML document. If this statement is not included, or something else is printed before this header, "Internal Server Error" would occur.

Statement 3 prints out an HTML code.

With POST method, the input string is read as standard input STDIN into a scalar $buffer (see statement 4), multiple values are separated by an "&". Each value has two parts, for example,

\[
\text{action} = \text{Word}
\]

the value on the left is the actual name of the form field. The value on the right is whatever you have selected from the radio box. To parse this, the buffer is split into an array of pairs with Perl's split function (see statement 5) when it meets an "&".

Statements 6 to 12 contain a for loop. If there are multiple variables in a string, it splits each pair, translates it back to plain text, and stores the values into a hash table called %FORM. The keys of %FORM are the form field names.

Statement 13 calls the Perl's system function, "home/cse_h2/weiqye/project/Projyme" which is the path to the external program. "proj" is the external program name. "action" is the variable name.
2.4.3. Database

Oracle 7 is available on the Auburn University Engineering Network. This project uses three tables. One table is named QUESTIONS which has eight columns --- Q#, QUESTION, ANSWER_A, ANSWER_B, ANSWER_C, ANSWER_D, ANSWER and SUBJECT.

For example, one row in this table might look like:

Q#: 1

QUESTION: “To save two versions of a document, the original and the revised version, use the ________ command.”

ANSWER_A: “a. Save”

ANSWER_B: “b. Save as”

ANSWER_C: “c. Copy”

ANSWER_D: “d. New”

ANSWER: “b”

SUBJECT: “Word”

Second table is named Grade with two columns --- USER_ID, and GRADE. Every time the user submits his answer, the test system will update the data, put the new record into the database.

The last table is named LOG with two columns --- USER_ID and PASSWORD. Every time the user enter user ID and password to access the system, the test system will
compare the values the user has entered with the data in the table LOG to let the right user take the test.

2.4.4. Embedded SQL and Pro*C

Pro * C includes embedded SQL using C language, which is available in Oracle 7 RDBMS. General syntax of an embedded SQL database statement looks like

```sql
EXEC SQL sql_statement;
```

The key words can be in capitals or lower cases. Declaration and use of C language variables must follow the C language syntax and SQL must be informed before they are used in EXEC SQL statements. For example,

```sql
EXEC SQL BEGIN DECLARE SECTION;
...
EXEC SQL END DECLARE SECTION;
```

Host variables in the INTO clause of a SELECT or FETCH statement are called output host variables because they hold attribute values output by Oracle. The system assigns the column values to the corresponding output host variables in the INTO clause.
For example,

EXEC SQL SELECT Question, Answer_a, Answer_b, Answer_c, Answer_d, INTO :questions.question_pc, :questions.answer_a, : questions.answer_b, questions.answer_c, questions.answer_d, 
FROM Questions 
WHERE Subject = :str1

Pro * C accepts the command line variable from Perl program, e.g., if the user clicks the "Word" radio button, "Word" will be passed to Pro * C. Then it will query the database (table questions), collect questions, and display them on the web. Also it generates four radio buttons associated with four answers to let user select the answer. One Submit button to submit current form, and one Reset button to let user redo the exam before they submit.

Pro * C can also generate HTML statements using ‘printf’ statement like:

```c
printf("<title> Online Test System </title>" );
printf(" Student Name:" );
printf("<input type = "text" name = "student_name" size = 25">" );
```
CHAPTER 3 DYNAMIC WEB PAGE IMPLEMENTED IN MIVA

3.1. Introduction of Miva

Miva (formerly called htmlscript) is an HTML-based web development language which provides the power of CGI-scripting via simple HTML-like tags. These new tags support variables, operators and Boolean logic statements, allowing anyone to create Web sites that interactively process data. Miva can also import and export data to and from files, process HTML forms, display information generated from other Web pages, pass information on to other pages and even call other scripts.

Miva runs as a “CGI” application and is compatible with all browsers that support the CGI standard. Miva processes Miva commands and then outputs the results in standard HTML code to the browser. A “View Page Source” from the browser will only show the HTML statements.

Miva Script is a ‘server-side’ scripting language that is implemented in Miva Empresa and Miva Mia. In contrast, a “client-side” language such as JavaScript is implemented by the browser. Miva Script tags correspond to typical programming language constructs such as assignment statements, conditional expressions, loops, and input/output statements, as well as Miva Script’s special database, mail, and commerce functionality.
Miva Script tags have the same format as HTML tags. The element names indicate their function, and the attributes specify values that the tags operate on. Miva Script tags can be freely mixed with HTML tags. Miva Script programs use the following kinds of auxiliary files:

- Delimited (Flat) Data files – Text files that contain one record per line, with fields separated by one or more ASCII characters such as “|” or “:”. The last line should be blank.
- Miva Database files – Databases that can be organized by multiple index files. A database may have a memo file (.dbt file) associated with it.
- Miva Index files – Files that create an ordered view of the database files.

When a browser requests a Miva Script document, Miva Empresa or Miva Mia will pre-process the document before passing it on to the browser in the following order.

1. The Miva Script tags (commands) are executed, and replaced by the results of the execution, which is generally a mixture of HTML tags and text (there may also be input to and output from external files).

2. The pre-processed document, consisting of the original HTML tags and text, together with the tags and text generated by Miva Empresa or Miva Mia, is served to the browser and displayed.

3.2. Miva tags and attributes

All Miva Script tags (also called elements) consist of a start-tag, such as <MvIF>, and sometimes an end tag, such as </MvIF>. <IF> and </IF> can also be used. End-tags
are either required or prohibited: tags that can have end-tags must have them in order for
the program to be correct; for all other tags, end-tags are prohibited. Tags for which end-
tags are prohibited are called empty tags. Not only do empty elements not have end-tags,
you cannot enter any ‘content’ for these elements, except for attribute values in the start-
tag.

For example, the \texttt{<IF>} tag requires an end-tag:

\begin{verbatim}
<IF EXPR = "\{age GT 6\}">
... some code and/or text
</IF>
\end{verbatim}

\texttt{<MvEVAL>} is an empty tag that has an attribute:

\begin{verbatim}
<MvEVAL EXPR = "\{age +1\}"
\end{verbatim}

3.3. Miva Script variables and variable scope

In Miva Script, a variable name can consist of letters ‘a’ through ‘z’ (upper or lower case), the underscore, ‘_’, and the digits ‘0’ through ‘9’. Other characters are permitted if they are ‘escaped’ with a backslash, such as:

\begin{verbatim}
<MvEVAL EXPR ="\{age\-old\}"
\end{verbatim}

This evaluates the variable \textit{age-old}; without the backslash, this expression would be interpreted as the value of age minus the value of old.

The period (dot), ‘.’, character can appear in a variable name, but only when it is following a prefix that indicates the scope in which the variable is understood.
Miva Script variable names are not case-sensitive: the variable age is the same as AGE and Age. Unlike in some programming languages, it is not necessary to declare or define variables --- just go ahead and use them. Using a variable in an expression without first assigning it a value also does not cause an error (although it may, of course, be responsible for logic errors in your program). Miva Script variables are typeless: the same variable can be assigned a numerical, text string, or Boolean (true/false) values.

To display the value of variable, use the <MvEVAL> tag or a macro:

<MvEVAL EXPR="\{var\}" or &var;

The scope of a variable is the context in which it exists and has meaning. The variable scopes available in Miva Script are local, database, system, and global.

3.4. Miva Script functions

Miva Script uses <MvFUNCTION> and </MvFUNCTION> tags to define functions. Following is an example of a Miva function:

<MvFUNCTION NAME = "myfunc" PARAMETER = "var1, var2,..."

STANDARDOUTPUTLEVEL = "html, text"

ERROROUTPUTLEVEL = "syntax, expression, runtime">

... code

<MvFUNCTIONRETURN VALUE ="\{expression\}">

... code

</MvFUNCTION>
myfunc(var1, var2, ...)

The function name in this example is 'myfunc'. Parameters var1, var2, etc., can optionally be defined with the PARAMETERS attribute. They are assigned values var1, var2, ..., in the function call. The function can optionally terminate with a return value specified with <MvFUNCRETURN>. Function can use locally defined variables whose scope is the function body. STANDARDOUTPUTLEVEL and ERROROUTPUTLEVEL are optional and are interpreted in the same way as in the <MIVA> tag. Functions are called in expressions. They can also be forward referenced, i.e., Miva Script code can make references to functions that are defined after the reference in question.

3.5. Miva database and data file.

Miva Script supports two kinds of databases: xbase3 (dBaseIII compatible, which is supported on all platforms), and ODBC (which is supported only on Microsoft Windows).

The full Miva database feature sets is currently supported only for xbase3 databases. Access to ODBC data sources is currently supported only with Miva Mia. A Miva xbase3 database organizes data into a simple tabular format that consists of records (which can be thought of as the rows of the table) that each has one or (usually) more fields (which can be thought of as the columns of the table). Here ‘table’ refers to the
ordering of the data—the way that the Miva Script language enables people to manipulate the data.

Miva Script has tags that let you perform database operations, such as: creating, opening, and closing a database; adding, deleting, and updating records; searching for records; and creating and using database indexes. Any database files or database indexes must be located in the data directory.

Miva allows user input or output data file. Every record in the data file is in one line. Fields are separated by one or more ASCII characters. Examples will be shown in next section.

3.6. A sample dynamic web page

3.6.1. The program

The same application program, Online Test System, was also implemented in Miva (HTL script 3.0). At first, the SYSTEM CONFIGURATION VARIABLES should be changed, i.e.,

```html
<LET server_name = "http://htmlscript.duc.auburn.edu/"/>
<LET script_name = "cgi-bin/htmlscript?”>
<LET pathfile = "-yeweiqi/project.hts”>
<LET fileurl = “&[server_name]&[script_name]&[pathfile]”>
<LET question_file = “questions.dat”>
```

In the fill-in form, METHOD is set equal to POST and ACTION is set equal to the URL of this script (as variable fileurl). When a user submits a fill-in form, it causes the script to be executed again. If there is no variable from fill-in form, e.g., the user
loads this script for the first time, the browser will display the same first page of the online test system as in Figure 1.

In the Miva program, the variable *function* works as a selection point that tells the script where to execute. It is passed as the FORM ACTION’s second command line argument, such as follow:

```html
<FORM METHOD = POST ACTION = "&[fileurl]+getfile">
```

here "&[fileurl]" is the first command line argument, "getfile" is the second command line argument.

The whole procedure would look like:

```html
<IF arg2 EQ “passwd”>
    <LET function = “passwd”>
        <IF arg2 EQ “getfile”>
            <LET function = “getfile”>
                </IF>
            </IF>
        </LET function = “getfile”>
    </LET function = “passwd”>
</IF>

<IF arg2 EQ “score”>
    <LET function = “score”>
</IF>

<IF function EQ “score”>
    ... codes to display the answer and grade of the test and update the grade ...
</IF>

<IF function EQ “getfile”>
    ... codes to display the questions and text fields ...
</IF>

<IF function EQ “ passwd”>
    ... codes to display subject selection page ...
</IF>
```
3.6.2. Data input and output

The flat delimited data file is used in this application. Each question is one line in the data file, and each field is delimited by a ‘|’. For example,

1 | To save two versions of a document, the original and the revised version, use the command. | a. Save | b. Save as | c. Copy | d. New | word

2 | If you select too much text, you can cancel the current selection by choosing Undo | b. choosing Undelete | c. clicking the mouse pointer inside the document window | d. pressing Esc | word

When data is needed, the <IMPORT> is used to retrieve the data and set the fields.

<IMPORT FILE = question_file>

<IMPORT DELIMITER = “|”>

<IMPORT ORDER = question_number, question, question_a, question_b, question_c, question_d question_answer, subjects>

<IMPORT DATA WHERE sub EQ subjects>
3.7. Running Miva program in Auburn University

Miva is available on the Auburn University Network. "htmlscript.auburn.edu" is a virtual host configured for executing Miva program. Miva program file has a .hts or .mv extension. The Miva file should be placed in a public_html directory or a subdirectory in Auburn University Network account. Data files should be placed in the home directory or a subdirectory of the home directory.
CHAPTER 4 DISCUSSION AND CONCLUSION

Using HTML, Perl and Oracle to implement dynamic web page is more complicated and difficult than using Miva Htmlscript language. It involves several different languages and commercial database systems.

Using Miva to implement the dynamic web page has many advantages. Miva tags can be interspersed with HTML tags, it is very easy for non-programmers to learn. If one already knows HTML, he does not have to learn an entirely new language. Also Miva can input data from files and write output to files. The most important thing is Miva has built-in database, which can be queried directly. People do not have to worry about using several different languages to write a database application program.

However, Miva also has several disadvantages, such as no typing, only basic programming capability, and only pass-by-value is allowed. It may not be flexible enough to implement large and complicated applications.
REFERENCES

<form method="post" action="http://roadrunner.eng.auburn.edu/cgi-bin/weiyou/start.sh">
  UserID : <input type="text" name="userid" size="10">  
  Password: <input type="password" name="pass" size="8">  
  <input type="submit" value="Submit">
  <input type="reset" value="Reset">
</form>

Please enter your user ID and password.
#!/bin/csh

# Name: Wei Qin Ye
# File name: start.sh
# This C shell file called by HTML file named index.html
# It also calls Perl file named project.pl

echo "Content-Type: text/html"
echo ""
echo "<head>"
echo "<title> CSE100 Introduction to Macrosoft Office 97 - online test</title>"
echo "</head>"
echo "<body>

/home/cse_h2/weiqye/project/Proj-ye/project.pl
echo "</body>"
#!/opt/local/bin/perl

print "<Head><Title>Project- On line test system</Title></Head>";
print"<BODY background="http://www.eng.auburn.edu/~lwp/back.jpg" text="#000000" link="#0000ee">
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});

##to split pairs of strings with '//' inside.
@pairs = split(/&/, $buffer);

foreach $pair (@pairs)
{
  ($name, $value) = split(/=/, $pair);
  $value =~ tr/+/ /;
  $value =~ s/\{a-fA-F0-9]/[a-fA-F0-9])/pack("C", hex($1))/eg;
  $value =~ s/!/-!/g;

  ##If value is empty, assign "null" to it.
  if(!$value)
  {
    $value="null";
  }

  $FORM($name) = $value;
}

system("/home/cse_h2/weiqye/project/Proj-ye/pro $FORM('userid') $FORM('pass')");
#!/bin/csh

# This C shell file is called by proc.c within the HTML format. It sets the database and Pro*C preprocessor path. It will also call the Perl program named proj.pl

echo "Content-Type: text/html"
echo ""
echo ""<head>"
echo ""<title> CSE100 Introduction to MicrosofOffice 97 - online test</title>"
echo ""</head>"
echo ""<body >"
echo ""<center>"
echo ""<h1>CSE100 Introduction to MicrosofOffice 97</h1>""<br>
echo ""<h1><i> On line Test </i></h1>"
echo ""</center>"
echo ""<h3>"
setenv TWO_TASK "T:oracle7:csedb"
set path = ($path /opt/SUNWspro/bin)
set path = ($path /opt/oracle7/product/7.2.2/proc/bin)
set path = ($path /usr/ucb)
set path = ($path /usr/ccs/bin)
setenv LD_LIBRARY_PATH /usr/uclib

echo ""<br>
/home/cse_h2/weiqye/project/Proj-ye/proj.pl
echo ""</h3>"
echo ""</body>""
#! /opt/local/bin/perl

print "<Head><Title>Project- On line test system </Title></Head>";

print "<body background="http://www.eng.auburn.edu/~lwp/back.jpg" text="#000000" link="#0000ee">

read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});

# to split pairs of strings with '&' inside.
@pairs = split(/&/, $buffer);

foreach $pair (@pairs)
{
    ($name, $value) = split(/=/, $pair);
    $value =~ tr/^[a-fA-F0-9][a-fA-F0-9]//pc('C', hex($1))/g;
    $value =~ s/-/!/g;

    if(!$value)
    {
        $value="null";
    }

    $FORM($name) = $value;
}

system("/home/cse_h2/weiqye/project/Proj-ye/test $FORM('action') $FORM('user')");
#include <stdio.h>
#include <string.h>

struct record
{
    int q_num;
    char question_pc[200];
    char a_pc[200];
    char b_pc[200];
    char c_pc[200];
    char d_pc[200];
    char answer[20];
    char subject_pc[20];
} questions;

char* str2="*
char* str3="*
char* str4="*
int count,i;

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

    /* Declare host and indicator variables*/
    exec sql begin declare section;

        char* username = "ops$weiye";
        char* password = "project";

    exec sql end declare section;

    /* include SQL "communication area" */
    exec sql include sqlca;

    /* register the routine for processing errors */
    exec sql whenever sqlerror do exit();

    /* login to ORACLE database */
    exec sql connect :username identified by :password;
                str1="Word";
                str2="Access";
                str3="Excel";

    /* Form output states here */
    printf("<form method = post action = \"http://roadrunner.eng.auburn.edu/cgi-bin/weiye/star2.sh\"">\n";
    printf("<font size = \"2\"">\n")

    if(!strcmp(argv[1], "word"))
    {
        printf(" The subject of this test is \s\",str1);
        printf("<input type="\"radio\" name="\"action\" value="\"word\" checked">")
    }
printf("<br><br>");

/* declare and open cursor */

EXEC SQL DECLARE c1 CURSOR FOR
SELECT Question, Answer_a, Answer_b, Answer_c, Answer_d
FROM Questions
WHERE Subject =:str1;

EXEC SQL OPEN c1;
EXEC SQL WHENEVER NOT FOUND DO BREAK;

for(i=1; i<21; i++) {
EXEC SQL FETCH c1 INTO :questions.question_pc, :questions.a_pc, :questions.b_pc, :questions.c_pc, :questions.d_pc;

printf("%d. %s",i,questions.question_pc);
printf("<br>");

printf("<input type="radio" name="selection%d" value = "a" checked"",i);
printf("<br>");

printf("<input type="radio" name="selection%d" value = "b"">",i);
printf("<br>");

printf("<input type="radio" name="selection%d" value = "c"">",i);
printf("<br>");

printf("<input type="radio" name="selection%d" value = "d"">",i);
printf("<br>");

}

/* close the cursor */
EXEC SQL CLOSE c1;

}

if(!strcmp(argv[1], "access"))
{
printf("The subject of this test is %s",str2);
printf("<input type="radio" name="action" value="access" checked"">");
printf("<br>");

/* declare and open cursor */

EXEC SQL DECLARE c2 CURSOR FOR
SELECT Question, Answer_a, Answer_b, Answer_c, Answer_d
FROM Questions
WHERE Subject =:str2;

EXEC SQL OPEN c2;
EXEC SQL WHENEVER NOT FOUND DO BREAK;

for(i=1; i<21; i++) {
EXEC SQL FETCH c2 INTO :questions.question_pc, :questions.a_pc, :questions.b_pc, :questions.c_pc, :questions.d_pc;

printf("%d. %s",i,questions.question_pc);
printf("<br>");

printf("<input type="radio" name="selection%d" value = "a" checked"",i);
printf("<br>");

printf("<input type="radio" name="selection%d" value = "b"">",i);
printf("<br>");

printf("<input type="radio" name="selection%d" value = "c"">",i);
printf("<br>");

printf("<input type="radio" name="selection%d" value = "d"">",i);
printf("<br>");
}
printf("<input type="radio" name="selection$\d\" value = \"c\"">",i);
printf("<br*>;
printf("<input type="radio" name="selection$\d\" value= \"d\"">",i);
printf("<br*>;
}

EXEC SQL CLOSE c2;
}
else if(!strcmp(argv[1], "excel"))
{
printf("The subject of this test is %s",str3);
printf("<input type="radio" name="action" value="\excel\" checked="">");
printf("<br*>;

/* declare and open cursor */

EXEC SQL DECLARE c3 CURSOR FOR
SELECT Question, Answer_a, Answer_b, Answer_c, Answer_d
FROM Questions
WHERE Subject = :str3;

EXEC SQL OPEN c3;
EXEC SQL WHENEVER NOT FOUND DO BREAK;

for(i=1; i<21; i++) {
EXEC SQL FETCH c3 INTO :questions.question_pc, :questions.a_pc, :questions.b_pc, :questions.
   c_pc, :questions.d_pc;

printf("$d. %s",i,questions.question_pc);
printf("<br*>");

printf("<input type="radio" name="selection$\d\" value = \"a\" checked="">",i);
printf("<br*>;

printf("<input type="radio" name="selection$\d\" value = \"b\"">",i);
printf("<br*>;

printf("<input type="radio" name="selection$\d\" value = \"c\"">",i);
printf("<br*>;

printf("<input type="radio" name="selection$\d\" value = \"d\"">",i);
printf("<br*>;
}

EXEC SQL CLOSE c3;
}
printf("<br*>");
printf("User %s has taken the test",argv[2]);
printf("<input type = \"radio\" name = \"user\" value=\"%s\" checked="">",argv[2]);
printf("<br*>;

printf("<input type = \"submit\" value=\"Grade\">");
printf("<input type = \"reset\" value=\"Reset\">");
printf("</form*>");
printf("</font*>");

/*Commit and pending changes and disconnect from Oracle*/
EXEC SQL COMMIT RELEASE;
exit(0);
/********************************************
* Name: Wengin Ye
* File name: test.c
* This file is created by Pro*C preprocessor after compiling
* the Pro*C program proj1.pc.
********************************************/

/* Result Sets Interface */
 ifndef SQL_CRSR
 # define SQL_CRSR
 struct sql_cursor
 {
   unsigned int currowcn;
   void *ptr1;
   void *ptr2;
   unsigned long magic;

   typedef struct sql_cursor sql_cursor;
   typedef struct sql_cursor SQL_CURSOR;
 # endif /* SQL_CRSR */

 /* File name & Package Name */
 struct sqlcxp
 {
   unsigned short fillen;
   char fillnam[8];

   static struct sqlcxp sqlfpn =
   {
     7,
     "test.pc"
   }

   static unsigned long sqlctx = 0;

   static struct sqlxed
   {
     unsigned int sqlvsn;
     unsigned int arrsz;
     unsigned int iters;
     unsigned int offset;
     unsigned short selerr;
     unsigned short sqlety;
     unsigned int unused;
     short *cd;
     unsigned char *sqlset;
     char *stmt;
     unsigned char **sqlhsv;
     unsigned int *sqlhs1;
     short **sqlpind;
     unsigned int *sqlparm;
     unsigned int **sqlparc;
     unsigned char *sqlstv[5];
     unsigned int sqlstrl[5];
     short *sqlindv[5];
     unsigned int sqlarm[5];
     unsigned int *sqlarc;
   } sql stm = {8,5};

   /* Prototypes */
   extern sqlcxp(*_unsigned long *, struct sqlxed *, struct sqlcxp * _*/);
   extern sqlcx2(*_unsigned long *, struct sqlxed *, struct sqlcxp * _*/);
   extern sqlcte(*_unsigned long *, struct sqlxed *, struct sqlcxp * _*/);
   extern sqlbuf(*_char * _*/);
   extern sqlgs2(*_char * _*/);
   extern sqlora(*_unsigned long *, void * _*/);

   /* Forms Interface */
   static int TAPSUCC = 0;
   static int TAPFAIL = 1403;
   static int TAPFTTL = 535;
   extern sqlerr(*_char *, int * _*/);

   static char *sq00002 =

"select Question ,Answer_a ,Answer_b ,Answer_c ,Answer_d from Questions where Subject=0 ;"
static char *sq0003 = "select Question ,Answer_a ,Answer_b ,Answer_c ,Answer_d from Questions where Subject=0 ;"
static char *sq0004 = "select Question ,Answer_a ,Answer_b ,Answer_c ,Answer_d from Questions where Subject=0 ;"
typedef struct { unsigned short len; unsigned char arr[1]; } VARCHAR;
typedef struct { unsigned short len; unsigned char arr[1]; } varchar;

/* cud (compilation unit data) array */
static short sqlcud0[] =
{8,34,
 2,0,0,1,0,0,27,58,0,3,3,0,1,0,1,97,0,0,1,97,0,0,0,1,10,0,0,
 28,0,0,2,100,0,9,84,0,1,1,0,1,0,1,97,0,0,
 46,0,0,2,0,0,13,88,0,5,0,0,1,0,2,97,0,0,2,97,0,0,2,97,0,0,2,97,0,0,2,97,0,0,
 80,0,0,2,0,0,15,113,0,0,0,0,0,1,0,
 94,0,0,3,100,0,9,131,0,1,1,0,1,0,1,97,0,0,
 112,0,0,3,0,0,13,135,0,5,0,0,1,0,2,97,0,0,2,97,0,0,2,97,0,0,2,97,0,0,
 146,0,0,3,0,0,15,158,0,0,0,0,1,0,
 160,0,0,4,100,0,9,174,0,1,1,0,1,0,1,97,0,0,
 178,0,0,4,0,0,13,178,0,5,0,0,1,0,2,97,0,0,2,97,0,0,2,97,0,0,2,97,0,0,
 212,0,0,4,0,0,15,201,0,0,0,0,0,1,0,
 226,0,0,5,7,0,30,216,0,0,0,0,0,1,0,
};

/******************************
* Name: Weiqin Ye
* File name: test.pc
* 
* This Pro*C program called by Perl program. It will querys
* table Questions, selects the questions with right subject.
* It will also display the test web page using 'printf'
* statement.
******************************/

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

struct record{
    int q_num;
    char question_pc[200];
    char a_pc[200];
    char b_pc[200];
    char c_pc[200];
    char d_pc[200];
    char answer[20];
    char subject_pc[20];
} questions;

char* str2="";
char* str3="";
char* str1="";
int count,i;

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

/* Declare host and indicator variables*/
/* exec sql begin declare section; */
    char* username = 'ops$weiqy' ;
    char* password = 'project';
execute SQL end declare section; /*
execute SQL include SQLCA; /*

$Header: sqlca.h 7020200.2 95/03/10 21:36:20 cli Generic<base> $ sqlca.h */

/* Copyright (c) 1985,1986 by Oracle Corporation. */

NAME
SQLCA : SQL Communications Area.

FUNCTION
Contains no code. Oracle fills in the SQLCA with status info
during the execution of a SQL stmt.

NOTES
******************************************************************************************************************************************
***
*** This file is SOD. Porters must change the data types ***
*** appropriately on their platform. See notes/pport.doc ***
*** for more information. ***
***
******************************************************************************************************************************************

If the symbol SQLCA_STORAGE_CLASS is defined, then the SQLCA
will be defined to have this storage class. For example:

#define SQLCA_STORAGE_CLASS extern

will define the SQLCA as an extern.

If the symbol SQLCA_INIT is defined, then the SQLCA will be
statically initialized. Although this is not necessary in order
to use the SQLCA, it is a good programming practice not to have
uninitialized variables. However, some C compilers/OS's don't
allow automatic variables to be init'd in this manner. Therefore,
if you are including the SQLCA in a place where it would be
an automatic AND your C compiler/OS doesn't allow this style
of initialization, then SQLCA_INIT should be left undefined --
all others can define SQLCA_INIT if they wish.

If the symbol SQLCA_NONE is defined, then the SQLCA variable will
not be defined at all. The symbol SQLCA_NONE should not be defined
in source modules that have embedded SQL. However, source modules
that have no embedded SQL, but need to manipulate a sqlca struct
passed in as a parameter, can set the SQLCA_NONE symbol to avoid
creation of an extraneous sqlca variable.

MODIFIED
xxxxxxx 12/12/94 - Bug 217878: note this is an SOD file
xxxxxxx 08/11/92 - No sqlca var if SQLCA_NONE macro set
xxxxxxx 12/06/94 - Ch SQLCA to not be an extern.
xxxxxxx 10/21/85 - Add initialization.
xxxxxxx 01/05/86 - Only initialize when SQLCA_INIT set
xxxxxxx 06/12/86 - Add SQLCA_STORAGE_CLASS option.
*/

#ifndef SQLCA
#define SQLCA 1

struct sqlca
{
  /* ub1 */ char sqlcaid[8];
  /* b4 */ long sqlabc;
  /* b4 */ long sqlcode;
  struct
  {
    /* ub2 */ unsigned short sqlerrml;
    /* ub1 */ char sqlerrmc[70];
    } sqlerrm;
    /* ub1 */ char sqlerrp[8];
    /* b4 */ long sqlerrd[6];
}
/* ub1 */ char sqlwarn[8];
/* ub1 */ char sqlext[8];
}

#ifndef SQLCA_NONE
#else SQLCA_STORAGE_CLASS struct sqlca sqlca
#endif

/* SQLCA_INIT */

*/

/* register the routine for processing errors */
/* exec sql whenever sqlext do exit(); */

/* login to ORACLE database */
/* exec sql connect :username identified by :password; */
{
  sqlstm.items = (unsigned int )10;
  sqlstm.offset = (unsigned int )2;
  sqlstm.cud = sqlcud0;
  sqlstm.sqlset = (unsigned char *)&sqlca;
  sqlstm.sqltype = (unsigned short )0;
  sqlstm.sqhtv[0] = (unsigned char *)username;
  sqlstm.sqhtl[0] = (unsigned int )0;
  sqlstm.sqindv[0] = (short * )0;
  sqlstm.sqind[0] = (unsigned int )0;
  sqlstm.sqhtv[1] = (unsigned char *)password;
  sqlstm.sqhtl[1] = (unsigned int )0;
  sqlstm.sqindv[1] = (short * )0;
  sqlstm.sqind[1] = (unsigned int )0;
  sqlstm.sqhsv = sqlstm.sqhtv;
  sqlstm.sqhs1 = sqlstm.sqhtl;
  sqlstm.sqpin = sqlstm.sqind;
  sqlstm.sqharm = sqlstm.sqharm;
  sqlstm.sqlarc = sqlstm.sqlarc;
sqlcxt(&sqlctx, &sqlstm, &sqlfpn);
if (sqlca.sqlcode < 0) exit();
}

str1="Word";
str2="Access";
str3="Excel";

/* Form output states here */
printf("<form method = post action = "http://roadrunner.eng.auburn.edu/cgi-bin/weigye/star2.sh"/>");
printf("<font size = "2"/>");

if(strcmp(argv[1], *word*))
{
  printf(" The subject of this test is %s",str1);
  printf("<input type="radio" name="action" value="word" checked>");
printf("<br>");

/* declare and open cursor */

/* EXEC SQL DECLARE c1 CURSOR FOR
SELECT Question, Answer_a, Answer_b, Answer_c, Answer_d
FROM Questions
WHERE Subject = :strl; */

/* EXEC SQL OPEN c1; */
{
    sqlstmt.stmt = sq0002;
    sqlstmt.iter = (unsigned int)1;
    sqlstmt.offset = (unsigned int)28;
    sqlstmt.cud = sqcud0;
    sqlstmt.sqlset = (unsigned char *)&sqlca;
    sqlstmt.sqlty = (unsigned short)0;
    sqlstmt.sqlhsv[0] = (unsigned char *)&strl;
    sqlstmt.sqlhstl[0] = (unsigned int)0;
    sqlstmt.sqlhsv[0] = (short *)0;
    sqlstmt.sqlharm[0] = (unsigned int)0;
    sqlstmt.sqlhsv = sqlstmt.sqlhstv;
    sqlstmt.sqlhshl = sqlstmt.sqlhstl;
    sqlstmt.sqlind = sqlstmt.sqlhsv;
    sqlstmt.sqlparm = sqlstmt.sqlharm;
    sqlstmt.sqlparc = sqlstmt.sqlharc;
    sqlctx(&sqlctx, &sqlstmt, &sqlfpn);
    if (sqlca.sqlcode < 0) exit();
}

/* EXEC SQL WHENEVER NOT FOUND DO BREAK; */

for(i=1; i<21; i++) {
    /* EXEC SQL FETCH c1 INTO :questions.question_pc, :questions.a_pc, :questions.b_pc, :questions.c_pc, :questions.d_pc; */
    {
        sqlstmt.iter = (unsigned int)1;
        sqlstmt.offset = (unsigned int)46;
        sqlstmt.cud = sqcud0;
        sqlstmt.sqlset = (unsigned char *)&sqlca;
        sqlstmt.sqlty = (unsigned short)0;
        sqlstmt.sqlhstv[0] = (unsigned char *)&(questions.question_pc);
        sqlstmt.sqlhstl[0] = (unsigned int)200;
        sqlstmt.sqlhsv[0] = (short *)0;
        sqlstmt.sqlharm[0] = (unsigned int)0;
        sqlstmt.sqlhstv[1] = (unsigned char *)&(questions.a_pc);
        sqlstmt.sqlhstl[1] = (unsigned int)200;
        sqlstmt.sqlhsv[1] = (short *)0;
        sqlstmt.sqlharm[1] = (unsigned int)0;
        sqlstmt.sqlhstv[2] = (unsigned char *)&(questions.b_pc);
        sqlstmt.sqlhstl[2] = (unsigned int)200;
        sqlstmt.sqlhsv[2] = (short *)0;
        sqlstmt.sqlharm[2] = (unsigned int)0;
        sqlstmt.sqlhstv[3] = (unsigned char *)&(questions.c_pc);
        sqlstmt.sqlhstl[3] = (unsigned int)200;
        sqlstmt.sqlhsv[3] = (short *)0;
        sqlstmt.sqlharm[3] = (unsigned int)0;
        sqlstmt.sqlhstv[4] = (unsigned char *)&(questions.d_pc);
        sqlstmt.sqlhstl[4] = (unsigned int)200;
        sqlstmt.sqlhsv[4] = (short *)0;
        sqlstmt.sqlharm[4] = (unsigned int)0;
        sqlstmt.sqlhsv = sqlstmt.sqlhstv;
        sqlstmt.sqlhshl = sqlstmt.sqlhstl;
        sqlstmt.sqlind = sqlstmt.sqlhsv;
        sqlstmt.sqlparm = sqlstmt.sqlharm;
        sqlstmt.sqlparc = sqlstmt.sqlharc;
        sqlctx(&sqlctx, &sqlstmt, &sqlfpn);
        if (sqlca.sqlcode == 1403) break;
        if (sqlca.sqlcode < 0) exit();
    }
}

printf("%d. %s", i, questions.question_pc);
printf("<br>");
```c
printf("<input type="radio" name="\$selection\" value = \"a\" checked="i>",i);
printf("%s", questions.a_pc);
printf("<br>");

printf("<input type="radio" name="\$selection\" value = \"b\"">",i);
printf("%s", questions.b_pc);
printf("<br>");

printf("<input type="radio" name="\$selection\" value = \"c\"">",i);
printf("%s", questions.c_pc);
printf("<br>");

printf("<input type="radio" name="\$selection\" value = \"d\"">",i);
printf("%s", questions.d_pc);
printf("<br>\<br><br>");
}

/* close the cursor */
/* EXEC SQL CLOSE c1; */
{
sqlstm.itors = (unsigned int )1;
sqlstm.offset = (unsigned int )80;
sqlstm.cud = sqlcud0;
sqlstm.sqlst = (unsigned char *)sqlca;
sqlstm.sqlty = (unsigned short)0;
sqlcex(&sqlctx, &sqlstm, &sqlfpn);
if (sqlca.sqlcode < 0) exit();
}

if(!strcmp(argv[1], "access"))
{
printf("The subject of this test is %s", str2);
printf("<input type="radio" name="\$action\" value="\"access\" checked="i>");
printf("<br><br>");

/* declare and open cursor */
/* EXEC SQL DECLARE c2 CURSOR FOR
SELECT Question, Answer_a, Answer_b, Answer_c, Answer_d
FROM Questions
WHERE Subject = :str2; */
/* EXEC SQL OPEN c2; */
{
sqlstm.stmt = sq0003;
sqlstm.itors = (unsigned int )1;
sqlstm.offset = (unsigned int )94;
sqlstm.cud = sqlcud0;
sqlstm.sqlst = (unsigned char *)sqlca;
sqlstm.sqlty = (unsigned short)0;
sqlstm.sqhtv[0] = (unsigned char *)str2;
sqlstm.sqhtl[0] = (unsigned int )0;
sqlstm.sqindv[0] = (short *)0;
sqlstm.sqarm[0] = (unsigned int )0;
sqlstm.sqphsv = sqlstm.sqhtv;
sqlstm.sqphal = sqlstm.sqhtl;
sqlstm.sqind = sqlstm.sqindv;
sqlstm.sqparm = sqlstm.sqarm;
sqlstm.sqparc = sqlstm.sqpharc;
sqlcex(&sqlctx, &sqlstm, &sqlfpn);
if (sqlca.sqlcode < 0) exit();
}

/* EXEC SQL WHENEVER NOT FOUND DO BREAK; */

for(i=1; i<21; i++) {
/* EXEC SQL FETCH c2 INTO :questions.question_pc, :questions.a_pc, :questions.b_pc, :questions.c_pc, :questions.d_pc; */
{
```
sqlstmt.iters = (unsigned int )1;
sqlstmt.offset = (unsigned int )112;
sqlstmt.cud = sqlcud0;
sqlstmt.sqlset = (unsigned char *) &sqlca;
sqlstmt.sqlyty = (unsigned short)0;
sqlstmt.sqshtr[0] = (unsigned char *)(questions.question_pc);
sqlstmt.sqshtr[1] = (unsigned int )200;
sqlstmt.sqindv[0] = (short *)0;
sqlstmt.sqharm[0] = (unsigned int )0;
sqlstmt.sqhstv[1] = (unsigned char *)(questions.a_pc);
sqlstmt.sqshtr[1] = (unsigned int )200;
sqlstmt.sqindv[1] = (short *)0;
sqlstmt.sqharm[1] = (unsigned int )0;
sqlstmt.sqhstv[2] = (unsigned char *)(questions.b_pc);
sqlstmt.sqshtr[2] = (unsigned int )200;
sqlstmt.sqindv[2] = (short *)0;
sqlstmt.sqharm[2] = (unsigned int )0;
sqlstmt.sqhstv[3] = (unsigned char *)(questions.c_pc);
sqlstmt.sqshtr[3] = (unsigned int )200;
sqlstmt.sqindv[3] = (short *)0;
sqlstmt.sqharm[3] = (unsigned int )0;
sqlstmt.sqhstv[4] = (unsigned char *)(questions.d_pc);
sqlstmt.sqshtr[4] = (unsigned int )200;
sqlstmt.sqindv[4] = (short *)0;
sqlstmt.sqharm[4] = (unsigned int )0;
sqlstmt.sqphsv = sqlstmt.sqhstv;
sqlstmt.sqphsl = sqlstmt.sqshtr;
sqlstmt.sqphsv = sqlstmt.sqindv;
sqlstmt.sqpharm = sqlstmt.sqharm;
sqlstmt.sqparc = sqlstmt.sqharm;
sqlctx(&sqlctx, &sqlstmt, &sqlfpn);
if (sqlca.sqlcode == 1403) break;
if (sqlca.sqlcode < 0) exit();
}

printf("%d. %s", i, questions.question_pc);
printf("<br>");

printf("<input type="radio" name="selection%d" value = \\
"a\" checked">", i);
printf("%s", questions.a_pc);
printf("<br>");

printf("<input type="radio" name="selection%d" value = \\
"b\"">", i);
printf("%s", questions.b_pc);
printf("<br>");

printf("<input type="radio" name="selection%d" value = \\
"c\"">", i);
printf("%s", questions.c_pc);
printf("<br>");

printf("<input type="radio" name="selection%d" value = \\
"d\"">", i);
printf("%s", questions.d_pc);
printf("<br><br><br>");

/* EXEC SQL CLOSE c2; */
{
sqlstmt.iters = (unsigned int )1;
sqlstmt.offset = (unsigned int )146;
sqlstmt.cud = sqlcud0;
sqlstmt.sqlset = (unsigned char *) &sqlca;
sqlstmt.sqlyty = (unsigned short)0;
sqlctx(&sqlctx, &sqlstmt, &sqlfpn);
if (sqlca.sqlcode < 0) exit();
}
else if(!strcmp(argv[1], 'excel'))
{
printf("The subject of this test is %s", str3);
printf("<input type="radio" name="action" value="excel" checked>");
printf("<br><br>");
/* declare and open cursor */

/* EXEC SQL DECLARE c3 CURSOR FOR
SELECT Question, Answer_a, Answer_b, Answer_c, Answer_d
FROM Questions
WHERE Subject = :str3; */

/* EXEC SQL OPEN c3; */
{
    sqlstm.stmt = sq0004;
    sqlstm.stmt = (unsigned int )1;
    sqlstm.offset = (unsigned int )160;
    sqlstm.cud = sqlcud0;
    sqlstm.sqlcst = (unsigned char *)&sqlca;
    sqlstm.sqlcsty = (unsigned short)0;
    sqlstm.sqshstv[0] = (unsigned char *)str3;
    sqlstm.sqshstl[0] = (unsigned int )0;
    sqlstm.sqshindv[0] = (short *)0;
    sqlstm.sqsharm[0] = (unsigned int )0;
    sqlstm.sqshsclv = sqlstm.sqshstv;
    sqlstm.sqshscl = sqlstm.sqshstl;
    sqlstm.sqshind = sqlstm.sqshindv;
    sqlstmt.sqsharm = sqlstm.sqsharm;
    sqlstm.sqsharc = sqlstm.sqsharm;
    sqlcex(&sqlctx, &sqlstm, &sqlfpn);
    if (sqlca.sqlcode < 0) exit();
}

/* EXEC SQL WHENEVER NOT FOUND DO BREAK; */

for(i=1; i<21; i++) {
    /* EXEC SQL FETCH c3 INTO :questions.question_pc, :questions.a_pc, :questions.b_pc, :questions.c_pc, :questions.d_pc; */
    
    sqlstm.stmt = (unsigned int )1;
    sqlstm.offset = (unsigned int )178;
    sqlstm.cud = sqlcud0;
    sqlstm.sqlcst = (unsigned char *)&sqlca;
    sqlstm.sqlcsty = (unsigned short)0;
    sqlstm.sqshstv[0] = (unsigned char *) (questions.question_pc);
    sqlstm.sqshstl[0] = (unsigned int )200;
    sqlstm.sqshindv[0] = (short *)0;
    sqlstm.sqsharm[0] = (unsigned int )0;
    sqlstm.sqshstv[1] = (unsigned char *) (questions.a_pc);
    sqlstm.sqshstl[1] = (unsigned int )200;
    sqlstm.sqshindv[1] = (short *)0;
    sqlstm.sqsharm[1] = (unsigned int )0;
    sqlstm.sqshstv[2] = (unsigned char *) (questions.b_pc);
    sqlstm.sqshstl[2] = (unsigned int )200;
    sqlstm.sqshindv[2] = (short *)0;
    sqlstm.sqsharm[2] = (unsigned int )0;
    sqlstm.sqshstv[3] = (unsigned char *) (questions.c_pc);
    sqlstm.sqshstl[3] = (unsigned int )200;
    sqlstm.sqshindv[3] = (short *)0;
    sqlstm.sqsharm[3] = (unsigned int )0;
    sqlstm.sqshstv[4] = (unsigned char *) (questions.d_pc);
    sqlstm.sqshstl[4] = (unsigned int )200;
    sqlstm.sqshindv[4] = (short *)0;
    sqlstm.sqsharm[4] = (unsigned int )0;
    sqlstm.sqshsclv = sqlstm.sqshstv;
    sqlstm.sqshscl = sqlstm.sqshstl;
    sqlstm.sqshind = sqlstm.sqshindv;
    sqlstmt.sqsharm = sqlstm.sqsharm;
    sqlstm.sqsharc = sqlstm.sqsharm;
    sqlcex(&sqlctx, &sqlstm, &sqlfpn);
    if (sqlca.sqlcode == 1403) break;
    if (sqlca.sqlcode < 0) exit();
}

printf("%d. %s", i, questions.question_pc);
printf("<br>");
printf("<input type="radio" name="selection\d" value = "a" checked="",i); printf("%s", questions.a_pc);
printf("<br>");

printf("<input type="radio" name="selection\d" value = "b">",i); printf("%s", questions.b_pc);
printf("<br>");

printf("<input type="radio" name="selection\d" value = "c">",i); printf("%s", questions.c_pc);
printf("<br>");

printf("<input type="radio" name="selection\d" value = "d">",i); printf("%s", questions.d_pc);
printf("<br><br>");

} /* EXEC SQL CLOSE c3; */
{
stmt.iters = (unsigned int )1;
stmt.offset = (unsigned int )212;
stmt.cud = sqlcud0;
stmt.sqlnest = (unsigned char *)&sqlca;
stmt.sqlety = (unsigned short)0;
sqlcx(&sqlctx, &stmt, &sqlfpn);
if (sqlca.sqlcode < 0) exit();
}
printf("<br>");
printf("User %s has taken the test",argv[2]);
printf("<input type = "radio" name="user" value="%s" checked="",argv[2]);
printf("<br>");

printf("<input type = "submit" value="Grade">");
printf("<input type = "reset" value="Reset">");
printf("</form>");
printf("</font>");

/*Commit and pending changes and disconnect from Oracle*/
/* EXEC SQL COMMIT RELEASE; */
{
stmt.iters = (unsigned int )1;
stmt.offset = (unsigned int )226;
stmt.cud = sqlcud0;
stmt.sqlnest = (unsigned char *)&sqlca;
stmt.sqlety = (unsigned short)0;
sqlcx(&sqlctx, &stmt, &sqlfpn);
if (sqlca.sqlcode < 0) exit();
}
exit(0);
#!/bin/csh

# Name: Weiqin Ye
# File name: star2.sh
# This file is called by proj.pc file

#-----------------------------------------------------------------
echo "Content-Type: text/html"
echo ""
echo "<head>"
echo "<title> CSE100 Introduction to Macrosoft Office 97 - online test</title>"
echo "</head>"
echo "<body>"
echo "<center>"
echo "<h1>CSE100 Introduction to Macrosoft Office 97 </h1><br>""
echo "<h1><i> Result of this test </i></h1>"
echo "</center>"
echo "<h3>"
setenv TWO_TASK "T:oracle7:cse6b"

set path = ($path /opt/SUNWspro/bin)
set path = ($path /opt/oracle7/product/7.2.2/proc/bin)
set path = ($path /usr/ucb)
set path = ($path /usr/ccs/bin)
set path = ($path /ccs/bin)
setenv LD_LIBRARY_PATH /usr/ucb/lib

echo "<br>" /
/home/cse_h2/weiqye/project/Proj-ye/proj1.pl
echo "</h3>"
echo "</body>"
#!/opt/local/bin/perl

print "<Head><Title>Project- On line test system </Title></Head>

print "<body background="http://www.eng.auburn.edu/~lwp/back.jpg" text="#000000" link="#0000ee">";

read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});

# to split pairs of strings with '&' inside.
$pairs = split(/&/, $buffer);

foreach $pair (@pairs)
{
    ($name, $value) = split(/=/, $pair);
    $value =~ tr/+/ /;
    $value =~ s/@(~a-fA-F0-9)[a-fA-F0-9]@/pack("C", hex($1))/eg;
    $value =~ s/-/-/g;
    if ($value eq "null")
    {
        ($value = "null");
    }
    $FORM{$name} = $value;

    system("/home/cse_h2/weiqye/project/Proj-ye/test1 $FORM{action} $FORM{selection1} $FORM{selection2} $FORM{selection3} $FORM{selection4} $FORM{selection5} $FORM{selection6} $FORM{selection7} $FORM{selection8} $FORM{selection9} $FORM{selection10} $FORM{selection11} $FORM{selection12} $FORM{selection13} $FORM{selection14} $FORM{selection15} $FORM{selection16} $FORM{selection17} $FORM{selection18} $FORM{selection19} $FORM{selection20} "$FORM{user}" "");

    print "<center><hr><hr><hr><hr>
    print "If you have any comments please sent mail to <a href="http://www.eng.auburn.edu/cgi-bin/mailto.pl?22474">weiqye@eng.auburn.edu</a>";
    print "</center>
";}
#include <stdio.h>
#include <string.h>

struct record{
    int q_num;
    char question_pc[200];
    char a_pc[200];
    char b_pc[200];
    char c_pc[200];
    char d_pc[200];
    char answer_pc[10];
    char subject_pc[20];
} questions;

char* str1;
char* str2;
char* str3;
char* str4;
char* str5;
int count,i;
float grade,j;

int argc, char* argv[]){

    /***********************************************************/
    /* Declare host and indicator variables*/
    exec sql begin declare section;
    char* username = "ops\$weiqye";
    char* password = "project";
    exec sql end declare section;
    /***********************************************************/
    /* include SQL "communication area" */
    exec sql include sqlca;
    /***********************************************************/
    /* register the routine for processing errors */
    exec sql whenever sqlerror do exit();
    /***********************************************************/
    /* login to ORACLE database */
    exec sql connect :username identified by :password;
    str1="Word";
    str2="Access";
    str3="Excel";
    str4=argv[4];
    str5=argv[5];
    j=0.0;

    /* check if the student fill in his name and student ID */
    if (((!strcmp(argv4, "null")) | (!strcmp(argv5, "null"))){
        printf("<center>");
        printf("<font size=5><b>");
    }
printf(" Please input your name and your student ID. \n\n");
printf("</center>");
}

/***** check answers of subject word ******/
else
{
    if(!strcmp(argv[1], "word"))
    {
        printf("<center>");
        /*printf("<br><br><br><br>");
        printf("Student %s has 28 questions correct in all 30 questions", str4);
        printf("<br>");
        printf("Your grade has been sent to the instructor.\n");
        printf("<br>");/*
        printf("<br><br><br>");
        printf("The correct answers for this test are ");
        printf("</br>");
        EXEC SQL DECLARE cl CURSOR FOR
        SELECT Answer
        FROM Questions
        WHERE Subject= :str1;
        EXEC SQL OPEN cl;
        EXEC SQL WHENEVER NOT FOUND DO BREAK;

        for (i=1; i<21; i++)
        {
            EXEC SQL FETCH cl INTO: questions.answer_pc;
            printf("%d. %s", i, questions.answer_pc);
            printf("<br>");
            printf("</center>");

            if(!strcmp(argv[2], "b")) j++;
            if(!strcmp(argv[3], "c")) j++;
            if(!strcmp(argv[4], "d")) j++;
            if(!strcmp(argv[5], "a")) j++;
            if(!strcmp(argv[6], "a")) j++;
            if(!strcmp(argv[7], "a") j++;
            if(!strcmp(argv[8], "c") j++;
            if(!strcmp(argv[9], "b") j++;
            if(!strcmp(argv[10], "b") j++;
            if(!strcmp(argv[11], "b") j++;
            if(!strcmp(argv[12], "b") j++;
            if(!strcmp(argv[13], "a") j++;
            if(!strcmp(argv[14], "a") j++;
            if(!strcmp(argv[15], "a") j++;
            if(!strcmp(argv[16], "c") j++;
            if(!strcmp(argv[17], "d") j++;
            if(!strcmp(argv[18], "d") j++;
            if(!strcmp(argv[19], "d") j++;
            if(!strcmp(argv[20], "b") j++;
            if(!strcmp(argv[21], "b") j++;

            grade=(j/20)*100;
            printf("<br><br><br><br>");
            printf("Student %s has %.0f questions correct in all 20 questions", argv[22], j);
            printf("<br>");
            printf("Your grade is %.1f. It has been sent to the instructor.\n", grade);
            printf("<br>");
        }
    }
}
if(!strcmp(argv[1], "access"))
{
    printf("<center>");
    printf("<br>");
    printf("<br><br><br>");
    printf("The correct answers for this test are ");
    printf("<br>");

    EXEC SQL DECLARE c2 CURSOR FOR
    SELECT Answer
    FROM Questions
    WHERE Subject= :str2;

    EXEC SQL OPEN c2;
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    for (i=1; i<21; i++) {
        EXEC SQL FETCH c2 INTO: questions.answer_pc;
        printf("%d. %s", i, questions.answer_pc);
        printf("<br>");
    }

    if(!strcmp(argv[2], "a")) j++;
    if(!strcmp(argv[3], "b")) j++;
    if(!strcmp(argv[4], "a")) j++;
    if(!strcmp(argv[5], "b")) j++;
    if(!strcmp(argv[6], "b")) j++;

    if(!strcmp(argv[7], "d")) j++;
    if(!strcmp(argv[8], "d")) j++;
    if(!strcmp(argv[9], "c")) j++;
    if(!strcmp(argv[10], "c")) j++;
    if(!strcmp(argv[11], "c")) j++;

    if(!strcmp(argv[12], "c")) j++;
    if(!strcmp(argv[13], "d")) j++;
    if(!strcmp(argv[14], "d")) j++;
    if(!strcmp(argv[15], "c")) j++;
    if(!strcmp(argv[16], "d")) j++;

    if(!strcmp(argv[17], "d")) j++;
    if(!strcmp(argv[18], "d")) j++;
    if(!strcmp(argv[19], "a")) j++;
    if(!strcmp(argv[20], "a")) j++;
    if(!strcmp(argv[21], "c")) j++;

    grade=(j/20)*100;
    printf("<br><br><br>");
    printf("Student %s has %.0f questions correct in all 20 questions", argv[22], j);
    printf("<br>");
    printf("Your grade is %.1f. It has been sent to the instructor.\n", grade);
    printf("<br>");
    printf("</center>");
}

else if(!strcmp(argv[1], "excel"))
{
    printf("<center>");
    printf("<br>");
    printf("<br><br>");
    printf("The correct answers for this test are ");
    printf("<br>");

    EXEC SQL DECLARE c3 CURSOR FOR
    SELECT Answer
    FROM Questions
    WHERE Subject= :str3;

    EXEC SQL OPEN c3;
for (i=1; i<21; i++) { 
EXEC SQL FETCH c3 INTO: questions.answer_pc;
printf("%d. %.s", i, questions.answer_pc);
printf("<br>");
if(!strcmp(argv[2],"b")) j++;
if(!strcmp(argv[3],"a")) j++;
if(!strcmp(argv[4],"c")) j++;
if(!strcmp(argv[5],"a")) j++;
if(!strcmp(argv[6],"b")) j++;
if(!strcmp(argv[7],"d")) j++;
if(!strcmp(argv[8],"d")) j++;
if(!strcmp(argv[9],"b")) j++;
if(!strcmp(argv[10],"a")) j++;
if(!strcmp(argv[11],"a")) j++;
if(!strcmp(argv[12],"c")) j++;
if(!strcmp(argv[13],"d")) j++;
if(!strcmp(argv[14],"c")) j++;
if(!strcmp(argv[15],"d")) j++;
if(!strcmp(argv[16],"b")) j++;
if(!strcmp(argv[17],"c")) j++;
if(!strcmp(argv[18],"a")) j++;
if(!strcmp(argv[19],"b")) j++;
if(!strcmp(argv[20],"b")) j++;
if(!strcmp(argv[21],"d")) j++;

grade=(j/20)*100;
printf("<br><br><br><br>");
printf("Student %s has %.0f of questions correct in all 20 questions", argv[22],j);
printf("<br>");
printf("Your grade is %.1f. It has been sent to the instructor.
", grade);
printf("<br>");
printf("</center>");
}

EXEC SQL INSERT INTO grade
VALUES (:argv[22], 'null', :grade);
printf("<br><br><br><br>");

/*Commit and pending changes and disconnect from Oracle*/
EXEC SQL COMMIT RELEASE;
exit(0);
/* Result Sets Interface */
#ifndef SQL_CRSR
#define SQL_CRSR
struct sql_cursor
{
    unsigned int curocn;
    void *ptr1;
    void *ptr2;
    unsigned long magic;
};
typedef struct sql_cursor sql_cursor;
typedef struct sql_cursor SQL_CURSOR;
#endif /* SQL_CRSR */

/* File name & Package Name */
struct sqlcxp
{
    unsigned short llen;
    char filnam[9];
};
static struct sqlcxp sqlfpn =
{
    8,
    "test1.pc"
};

static unsigned long sqlctx = 0;

static struct sqlxed {
    unsigned int sqlver;
    unsigned int arrsz;
    unsigned int iters;
    unsigned int offset;
    unsigned short selerr;
    unsigned short sqlalty;
    unsigned int unused;
    short *cud;
    unsigned char *sqlcst;
    char *stmt;
    unsigned char *sqlhsv;
    unsigned int *sqlhs1;
    short **sqlind;
    unsigned int *sqlarm;
    unsigned int **sqlarc;
    unsigned char *sqltv[3];
    unsigned int sqlts1[3];
    short *sqlndv[3];
    unsigned int sqlarr[3];
    unsigned int *sqlarc[3];
} sqlstm = (8, 3);

/* Prototypes */
extern sqlcxp(/* _unsigned long *, struct sqlxed *, struct sqlcxp */);
extern sqlcx2(/* _unsigned long *, struct sqlxed *, struct sqlcxp */);
extern sqlcxe(/* _unsigned long *, struct sqlxed *, struct sqlcxp */);
extern sqlbuf(/* _char *, int */);
extern sqls2(/* _char *, int */);
extern sqlora(/* _unsigned long *, void */);

/* Forms Interface */
static int TAPSUCC = 0;
static int TAPFAIL = 1403;
static int TAPTL = 535;
extern sqliem(/* _char *, int */);
/* compilation unit data */
static short sqlcud0[] =
{8, 34,
 2, 0, 0, 1, 0, 0, 27, 60, 0, 3, 3, 0, 1, 0, 1, 97, 0, 0, 1, 97, 0, 0, 1, 10, 0, 0,
 28, 0, 0, 2, 5, 0, 9, 104, 0, 1, 1, 0, 1, 0, 1, 97, 0, 0,
 46, 0, 0, 2, 0, 0, 13, 108, 0, 1, 0, 0, 1, 0, 2, 97, 0, 0,
 64, 0, 0, 3, 5, 0, 9, 165, 0, 1, 1, 0, 1, 0, 1, 97, 0, 0,
 82, 0, 0, 3, 0, 0, 13, 165, 0, 1, 0, 0, 1, 0, 2, 97, 0, 0,
 100, 0, 0, 0, 4, 5, 0, 9, 225, 0, 1, 1, 0, 1, 0, 1, 97, 0, 0,
 118, 0, 0, 0, 4, 0, 0, 13, 229, 0, 1, 0, 1, 0, 2, 97, 0, 0,
 136, 0, 0, 5, 4, 1, 0, 3, 274, 0, 2, 2, 0, 1, 0, 1, 97, 0, 0, 1, 4, 0, 0,
 158, 0, 0, 0, 0, 0, 7, 0, 30, 280, 0, 0, 0, 0, 1, 1, 0,
};

/*******************************************************************************/
* Name: Weiqi Ye
* File name: proj1.c
* Title: This program is called by proj1.pl Perl program.
* It will query the database, compare the answers users input
* and those right answers in database, calculate the grade
* and also enter the student name, ID and his grade in to the database.
*******************************************************************************/

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

typedef struct { unsigned short len; unsigned char arr[1]; } VARCHAR;
typedef struct { unsigned short len; unsigned char arr[1]; } varchar;

#define _config

static char *sq0002 =
"select Answer from Questions where Subject=:b0 ";
static char *sq0003 =
"select Answer from Questions where Subject=:b0 ";
static char *sq0004 =
"select Answer from Questions where Subject=:b0 ";
typedef struct { unsigned short len; unsigned char arr[1]; } VARCHAR;
typedef struct { unsigned short len; unsigned char arr[1]; } varchar;

/* compilation unit data */
static short sqlcud0[] =
{8, 34,
 2, 0, 0, 1, 0, 0, 27, 60, 0, 3, 3, 0, 1, 0, 1, 97, 0, 0, 1, 97, 0, 0, 1, 10, 0, 0,
 28, 0, 0, 2, 5, 0, 9, 104, 0, 1, 1, 0, 1, 0, 1, 97, 0, 0,
 46, 0, 0, 2, 0, 0, 13, 108, 0, 1, 0, 0, 1, 0, 2, 97, 0, 0,
 64, 0, 0, 3, 5, 0, 9, 165, 0, 1, 1, 0, 1, 0, 1, 97, 0, 0,
 82, 0, 0, 3, 0, 0, 13, 165, 0, 1, 0, 0, 1, 0, 2, 97, 0, 0,
 100, 0, 0, 0, 4, 5, 0, 9, 225, 0, 1, 1, 0, 1, 0, 1, 97, 0, 0,
 118, 0, 0, 0, 4, 0, 0, 13, 229, 0, 1, 0, 1, 0, 2, 97, 0, 0,
 136, 0, 0, 5, 4, 1, 0, 3, 274, 0, 2, 2, 0, 1, 0, 1, 97, 0, 0, 1, 4, 0, 0,
 158, 0, 0, 0, 0, 0, 7, 0, 30, 280, 0, 0, 0, 0, 1, 1, 0,
};

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

struct record{
  int q_num;
  char question_pc[200];
  char a_pc[200];
  char b_pc[200];
  char c_pc[200];
  char d_pc[200];
  char answer_pc[10];
  char subject_pc[20];
} questions;

char* str1;
char* str2;
char* str3;
char* str4;
char* str5;
int count,i;
float grade,j;

main(int argc, char* argv[])
{
    /* Declare host and indicator variables*/
    /* exec sql begin declare section; */
    char* username = "ops\weiqi#";
    char* password = "project";
    /* exec sql end declare section; */
    /* include SQL "communication area" */
}
/* exec sql include sqlca; */

/* $Header: sqlca.h 7020200.2 95/03/10 21:36:20 cli Generic<base> $ sqlca.h */
/* Copyright (c) 1985,1986 by Oracle Corporation. */
/

NAME
SQLCA : SQL Communications Area.

FUNCTION
Contains no code. Oracle fills in the SQLCA with status info
during the execution of a SQL stmt.

NOTES
******************************************************************************
*** This file is SODS. Porters must change the data types ***
*** appropriately on their platform. See notes/pcport.doc ***
*** for more information. ***
******************************************************************************

If the symbol SQLCA_STORAGE_CLASS is defined, then the SQLCA
will be defined to have this storage class. For example:

#define SQLCA_STORAGE_CLASS extern

will define the SQLCA as an extern.

If the symbol SQLCA_INIT is defined, then the SQLCA will be
statically initialized. Although this is not necessary in order
to use the SQLCA, it is a good programming practice not to have
uninitialized variables. However, some C compilers/OS's don't
allow automatic variables to be init'd in this manner. Therefore,
if you are INCLUDE'ing the SQLCA in a place where it would be
an automatic AND your C compiler/OS doesn't allow this style
of initialization, then SQLCA_INIT should be left undefined --
all others can define SQLCA_INIT if they wish.

If the symbol SQLCA_NONE is defined, then the SQLCA variable will
not be defined at all. The symbol SQLCA_NONE should not be defined
in source modules that have embedded SQL. However, source modules
that have no embedded SQL, but need to manipulate a sqlca struct
passed in as a parameter, can set the SQLCA_NONE symbol to avoid
creation of an extraneous sqlca variable.

MODIFIED
xxxxxx 12/12/94 - Bug 217878: note this is an SODS file
xxxxxxx 08/11/92 - No sqlca var if SQLCA_NONE macro set
xxxxxx 12/06/84 - Ch SQLCA to not be an extern.
xxxxxx 10/21/85 - Add initialization.
xxxxxxx 01/05/86 - Only initialize when SQLCA_INIT set.
xxxxxx 06/12/86 - Add SQLCA_STORAGE_CLASS option.
/*

ifndef SQLCA
#define SQLCA 1

struct sqlca
{
    /* ub1 */ char sqlcaid[8];
    /* b4 */ long sqlabc;
    /* b4 */ long sqlcode;
    struct
    {
        /* ub2 */ unsigned short sqlerrm;
        /* ub1 */ char sqlerrmc[70];
    } sqlerrm;
    /* ub1 */ char sqlerrp[8];
    /* b4 */ long sqlerrd[6];
    /* ub1 */ char sqlwarn[8];
    /* ub1 */ char sqlwarn[8];
};
```c
#ifndef SQLCA_NONE
#define SQLCA_STORAGE_CLASS
SQLCA_STORAGE_CLASS struct sqlca sqlca
#else
struct sqlca sqlca
#endif

#ifndef SQLCA_INIT
#define SQLCA_INIT
={
   {'S', 'Q', 'L', 'C', 'A', ' ', ' ', ' ', ' '},
   sizeof(struct sqlca),
   { 0, (0),
      { 'N', 'O', 'T', ' ', ' ', 'S', 'E', 'T', ' ' },
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0} }
}
#endif
#endif

/* end SQLCA */

/* register the routine for processing errors */
/* exec sql whenever sqlevent do exit(); */
/* login to ORACLE database */
/* exec sql connect :username identified by :password; */
{

sqlstmt_iter = (unsigned int )10;
sqldataoff = (unsigned int )2;
sqlcmd = sqlcmd0;
sqlstmt.sqlset = (unsigned char *)&sqlca;
sqlstmt.sqltype = (unsigned short)10;
sqlstmt.sqlstv[0] = (unsigned char *)username;
sqlstmt.sqlstl[0] = (unsigned int )0;
sqlstmt.sqlindv[0] = (short *)0;
sqlstmt.sqlharm[0] = (unsigned int )0;
sqlstmt.sqlstv[1] = (unsigned char *)password;
sqlstmt.sqlstl[1] = (unsigned int )0;
sqlstmt.sqlindv[1] = (short *)0;
sqlstmt.sqlharm[1] = (unsigned int )0;
sqlstmt.sqlshsv = sqlstmt.sqlstv;
sqlstmt.sqlshsl = sqlstmt.sqlstl;
sqlstmt.sqlshind = sqlstmt.sqlindv;
sqlstmt.sqlsharm = sqlstmt.sqlharm;
sqlstmt.sqlsharc = sqlstmt.sqlsharc;
sqlcex(sqlctx, sqlstmt, sqlfpn);
if (sqlca.sqlcode < 0) exit();

str1="Word";
str2="Access";
str3="Excel";
str4=argv[4];
str5=argv[5];

j=0.0;

/* check if the student fill in his name and student ID */
if (strstr(argv[4], "null") || strstr(argv[5], "null")) {
    printf("<center>");
    printf("<font size=5><b>");
    printf("Please input your name and your student ID.\n\n");
    printf("</center>");
}
```

/*
check answers of subject word
*/

else
{
  if(!strcmp(argv[1], "word"))
  {
    printf("<center>*");
    /*printf("<br><br><br><br>");
    printf("Student %s has 28 questions correct in all 30 questions", str4);
    printf("<br>");
    printf("Your grade has been sent to the instructor.\n");
    printf("<br>*");
    printf("<br><br>");
    printf("The correct answers for this test are ");
    printf("<br>*");
    */
    /* EXEC SQL DECLARE cl CURSOR FOR
SELECT Answer
FROM Questions
WHERE Subject= :str1; */

    sqlstm.stmt = sq0002;
    sqlstm.iters = (unsigned int )1;
    sqlstm.offset = (unsigned int )28;
    sqlstm.cud = sqcud0;
    sqlstm.sqlset = (unsigned char *)&sqlca;
    sqlstm.sqlgety = (unsigned short)0;
    sqlstm.sqlstv[0] = (unsigned char *)str1;
    sqlstm.sqlstl[0] = (unsigned int )0;
    sqlstm.sqlindv[0] = (short *)0;
    sqlstm.sqlarm[0] = (unsigned int )0;
    sqlstm.sqlhsv = sqlstm.sqlstv;
    sqlstm.sqlshl = sqlstm.sqlstl;
    sqlstm.sqlind = sqlstm.sqlindv;
    sqlstm.sqlparm = sqlstm.sqlarm;
    sqlstm.sqlparc = sqlstm.sqlarc;
    sqlcex(&sqlctx, &sqlstm, &sqlfpn);
    if (sqlca.sqlcode < 0) exit();

    /* EXEC SQL WHENEVER NOT FOUND DO BREAK; */
    for (i=1; i<21; i++)
    { /* EXEC SQL FETCH cl INTO: questions.answer_pc; */

      sqlstm.iters = (unsigned int )1;
      sqlstm.offset = (unsigned int )46;
      sqlstm.cud = sqcud0;
      sqlstm.sqlset = (unsigned char *)&sqlca;
      sqlstm.sqlgety = (unsigned short)0;
      sqlstm.sqlstv[0] = (unsigned char *)(questions.answer_pc);
      sqlstm.sqlstl[0] = (unsigned int )10;
      sqlstm.sqlindv[0] = (short *)0;
      sqlstm.sqlarm[0] = (unsigned int )0;
      sqlstm.sqlhsv = sqlstm.sqlstv;
      sqlstm.sqlshl = sqlstm.sqlstl;
      sqlstm.sqlind = sqlstm.sqlindv;
      sqlstm.sqlparm = sqlstm.sqlarm;
      sqlstm.sqlparc = sqlstm.sqlarc;
      sqlcex(&sqlctx, &sqlstm, &sqlfpn);
      if (sqlca.sqlcode == 1403) break;
      if (sqlca.sqlcode < 0) exit();

    }

  printf("&d. %s", i, questions.answer_pc);
  printf("<br>");
  printf("<br>");
}
if(!strcmp(argv[2],"b")) j++; 
if(!strcmp(argv[3],"c")) j++; 
if(!strcmp(argv[4],"d")) j++; 
if(!strcmp(argv[5],"d")) j++; 
if(!strcmp(argv[6],"a")) j++; 
if(!strcmp(argv[7],"a")) j++; 
if(!strcmp(argv[8],"c")) j++; 
if(!strcmp(argv[9],"b")) j++; 
if(!strcmp(argv[10],"b")) j++; 
if(!strcmp(argv[11],"b")) j++; 
if(!strcmp(argv[12],"b")) j++; 
if(!strcmp(argv[13],"a")) j++; 
if(!strcmp(argv[14],"a")) j++; 
if(!strcmp(argv[15],"a")) j++; 
if(!strcmp(argv[16],"c")) j++; 
if(!strcmp(argv[17],"d")) j++; 
if(!strcmp(argv[18],"d")) j++; 
if(!strcmp(argv[19],"d")) j++; 
if(!strcmp(argv[20],"b")) j++; 
if(!strcmp(argv[21],"b")) j++; 

grade=(j/20)*100;

printf("<br><br><br><br>*");
printf("Student %s has % .0f questions correct in all 20 questions", argv[22], j);
printf("<br><br><br><br>*");
printf("Your grade is %.1f. It has been sent to the instructor.\n", grade);
printf("<br>");

}

if(!strcmp(argv[1], "access"))
{
    printf("<center>");
    printf("<br>");
    printf("<br><br><br><br>*");
    printf("The correct answers for this test are *");
    printf("<br>");

    /* EXEC SQL DECLARE c2 CURSOR FOR 
    SELECT Answer 
    FROM Questions 
    WHERE Subject= :str2; */

    /* EXEC SQL OPEN c2; */

    sqlstmt.stmt = sq0003;
    sqlstmt.iterator = (unsigned int )1;
    sqlstmt.offset = (unsigned int )64;
    sqlstmt.cud = sqlcud0;
    sqlstmt.sqlest = (unsigned char *)sqlca;
    sqlstmt.sqlety = (unsigned short)0;
    sqlstmt.sqhtsv[0] = (unsigned char *)str2;
    sqlstmt.sqhtsl[0] = (unsigned int )0;
    sqlstmt.sqindv[0] = (short *)0;
    sqlstmt.sqharm[0] = (unsigned int )0;
    sqlstmt.sqhsrv = sqlstmt.sqhtsv;
    sqlstmt.sqphsl = sqlstmt.sqhtsl;
    sqlstmt.sqphd = sqlstmt.sqindv;
    sqlstmt.sqpham = sqlstmt.sqharm;
    sqlstmt.sqphpc = sqlstmt.sqharm;
    sqlcex(&sqlctx, &sqlstmt, &sqlfpn);
    if (sqlca.sqlcode < 0) exit();
}

    /* EXEC SQL WHENEVER NOT FOUND DO BREAK; */

    for (i=1; i<21; i++) {
        /* EXEC SQL FETCH c2 INTO: questions.answer_pc; */
        sqlstmt.sqfcty = (unsigned short)0;
    }
printf("<br><br><br>\n");
printf("Student %s has %.0f questions correct in all 20 questions", argv[22], j);
printf("<br>\n");
printf("Your grade is %.1f. It has been sent to the instructor.\n", grade);
printf("<br>\n");
printf("<center>\n");
}

/* EXEC SQL INSERT INTO grade VALUES (:argv[22], 'null', :grade); */
{
sqlstmt.stat = "insert into grade values (:b0,'null',:b1)";
sqlstmt.iter = (unsigned int )1;
sqlstmt.offset = (unsigned int )136;
sqlstmt.cud = sqlcud0;
sqlstmt.sqlest = (unsigned char *)&sqlca;
sqlstmt.sqlty = (unsigned short)0;
sqlstmt.sqhtv[0] = (unsigned char *)argv[22];
sqlstmt.sqhtl[0] = (unsigned int )0;
sqlstmt.sqindv[0] = (short *)0;
sqlstmt.sqharm[0] = (unsigned int )0;
sqlstmt.sqhtv[1] = (unsigned char *)&grade;
sqlstmt.sqhtl[1] = (unsigned int )4;
sqlstmt.sqindv[1] = (short *)0;
sqlstmt.sqharm[1] = (unsigned int )0;
sqlstmt.sqphsv = sqlstmt.sqhtv;
sqlstmt.sqphsl = sqlstmt.sqhtl;
sqlstmt.sqphid = sqlstmt.sqindv;
sqlstmt.sqparm = sqlstmt.sqharm;
sqlstmt.sqparc = sqlstmt.sqharc;
sqlcex(&sqlctx, &sqlstm, &sqlfpn);
if (sqlca.sqlcode < 0) exit();
}

printf("<br><br><br>\n");

/*Commit and pending changes and disconnect from Oracle*/
/* EXEC SQL COMMIT RELEASE; */
{
sqlstmt.iter = (unsigned int )1;
sqlstmt.offset = (unsigned int )158;
sqlstmt.cud = sqlcud0;
sqlstmt.sqlest = (unsigned char *)&sqlca;
sqlstmt.sqlty = (unsigned short)0;
sqlcex(&sqlctx, &sqlstm, &sqlfpn);
if (sqlca.sqlcode < 0) exit();
}

exit(0);
}
(sqlstmt.iter = (unsigned int)1;
sqlstmt.offset = (unsigned int)82;
sqlstmt.cud = sqlcud0;
sqlstmt.sqlstate = (unsigned char*)"sqlca";
sqlstmt.sqlopt = (unsigned short)0;
sqlstmt.sqlopt[0] = (unsigned char*)"sqlca";
sqlstmt.sqlhstl[0] = (unsigned int)10;
sqlstmt.sqlindv[0] = (short*)0;
sqlstmt.sqlarm[0] = (unsigned int)0;
sqlstmt.sqlhsv = sqlstmt.sqlopt;
sqlstmt.sqlhsv1 = sqlstmt.sqlhstl;
sqlstmt.sqlind = sqlstmt.sqlindv;
sqlstmt.sqlparm = sqlstmt.sqlarm;
sqlstmt.sqlparc = sqlstmt.sqlhsv;
sqlset(&sqlctx, &sqlstmt, &sqlfpn);
if (sqlca.sqlcode == 1403) break;
if (sqlca.sqlcode < 0) exit();
}

printf("%d. %s", i, questions.answer_pc);
printf("<br/>") ;

if(!strcmp(argv[2], "a")) j++;
if(!strcmp(argv[3], "b")) j++;
if(!strcmp(argv[4], "c")) j++;
if(!strcmp(argv[5], "d")) j++;
if(!strcmp(argv[6], "e")) j++;

if(!strcmp(argv[7], "d")) j++;
if(!strcmp(argv[8], "d")) j++;
if(!strcmp(argv[9], "d")) j++;
if(!strcmp(argv[10], "c")) j++;
if(!strcmp(argv[11], "c")) j++;

if(!strcmp(argv[12], "c")) j++;
if(!strcmp(argv[13], "d")) j++;
if(!strcmp(argv[14], "d")) j++;
if(!strcmp(argv[15], "c")) j++;
if(!strcmp(argv[16], "d")) j++;

if(!strcmp(argv[17], "d")) j++;
if(!strcmp(argv[18], "d")) j++;
if(!strcmp(argv[19], "a")) j++;
if(!strcmp(argv[20], "a")) j++;
if(!strcmp(argv[21], "c")) j++;

grade = (j/20)*100;
printf("<br><br><br><br>\n");
printf("Student %s has %.0f questions correct out of all 20 questions", argv[22], j);
printf("<br/>");
printf("Your grade is %.1f. It has been sent to the instructor.\n", grade);
printf("<br/>");
printf("</center>\n");

else if(!strcmp(argv[1], "excel") )
{
printf("<center>");
printf("<br/>");
printf("<br>");
printf("<br>");
printf("The correct answers for this test are *\n");
printf("<br>");
/* EXEC SQL DECLARE c3 CURSOR FOR SELECT Answer FROM Questions WHERE Subject = :str3 */