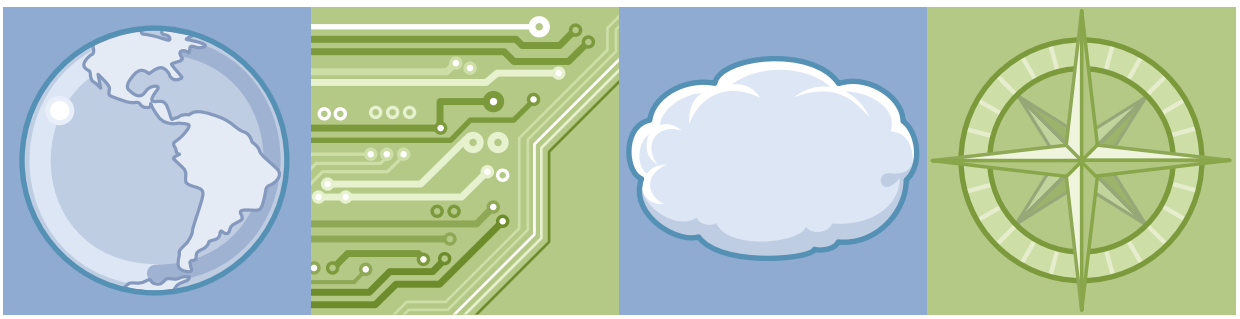
IBM Training

Student Exercises

PERL Programming for Open Systems

Course code AN203G ERC 1.0



Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AIX 5L™	AIX®	DB™
DB2®	Informix®	Lotus Notes®
Lotus®	Notes®	PartnerWorld®
Power®	PowerVM®	POWER6®
Tivoli®	WebSphere®	

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

October 2015 edition

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AIX 5L™	AIX®	DB™
DB2®	Informix®	Lotus Notes®
Lotus®	Notes®	PartnerWorld®
Power®	PowerVM®	POWER6®
Tivoli®	WebSphere®	

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

Exercises description

There are no definitive details on how to perform the tasks. You are given the opportunity to work through the exercise given what you have learned in the unit presentation, utilizing the unit Student Notebook, your past experience, and maybe a little intuition.

You can choose to:

- Use the files in /home/perl to see the completed files.
- Type and debug the programs from scratch.

Exercise 1. Overview

What this exercise is about

This exercise provides an opportunity to begin programming in Perl.

What you should be able to do

After completing this exercise, you should be able to:

- Retrieve basic output from a Perl program

Exercise instructions

__ 1. Construct and execute a program to: Print out the famous "Hello World"to your display.

__ 2. Construct and execute a program to: Print out the famous "Hello World"to your display with an initial newline, tab, and an ending newline.

End of exercise

Exercise 2. Simple data types

What this exercise is about

This exercise provides an opportunity to generate scalar data to the display.

What you should be able to do

After completing this exercise, you should be able to:

- Develop and print scalar data

Exercise instructions

__ 1. Construct and execute a program to:

  __ a. Print out the sum of the numbers nine and one and five added together.

__ 2. Construct and execute a program to:

  __ a. Compute and print the area of a rectangle with sides of 12 and 24.

__ 3. Construct and execute a program to:

  __ a. Compute and print the square root of two.

  __ b. Compute and print the square of three and then five.

  __ c. Compute and print the sum of the previous line.

__ 4. Construct and execute a program to:

  __ a. Print out the result of concatenating two strings `"Every good boy"` and `"does fine"` together with the period (`.`) operator.

End of exercise

Exercise 3. I/O using standard input and output

What this exercise is about

This exercise provides an opportunity to read input into a program.

What you should be able to do

After completing this exercise, you should be able to:

- Give a program input from the keyboard

Exercise instructions

__ 1. Construct and execute a program to:

  __ a. Change the rectangle program in the previous exercise on scalar data to accept the sides as input from the keyboard.

__ 2. Construct and execute a program to:

  __ a. Read in a string from standard input and also a number. Then, print the resulting string after using the repetition (x) operator.

__ 3. Construct and execute a program to:

  __ a. Print out the result of 20 new-line characters using the repetition (x) operator.

End of exercise

Exercise 4. Flow control

What this exercise is about

This exercise provides an opportunity to investigate the application of control structures.

What you should be able to do

After completing this exercise, you should be able to:

- Code if and while statements

Exercise instructions

__ 1. Construct and execute a program to:

  __ a. Ask for the elevation above mean sea level. Print `"above mean sea level"` if the number entered is above zero and `"below mean sea level"` if below zero.

__ 2. Modify the previous program to:

  __ a. Include a test for the input being exactly zero and print `"exactly sea level"`.

__ 3. Construct and execute a program to build up one long line from keyboard input:

  __ a. Read strings from the keyboard until an empty line.

  __ b. Append each input line to the previous lines.

  __ c. Print the resulting string.

Note

An empty line read from the keyboard still has a line delimiter.

End of exercise

Exercise 5. Lists and arrays

What this exercise is about

This exercise provides an opportunity to work with lists and arrays.

What you should be able to do

After completing this exercise, you should be able to:

- Input lists and arrays

Exercise instructions

__ 1. Construct and execute a program to:

__ a. Create an array variable holding the following list:

```
('diamond', 'emerald', 'amethyst', 'garnet', 'topaz',
'turquoise', 'sapphire', 'ruby', 'opal', 'peridot', 'pearl',
'aquamarine')
```

__ 2. Print the following:

__ a. The entire array inside and outside a string.

__ b. The phrase `"diamond, ruby and emerald"` using single element access.

__ c. The precious stones starting with a vowel using an array slice. Sort the array and print it one stone per line.

End of exercise

Exercise 6. Regular expressions

What this exercise is about

This exercise provides an opportunity to learn about string searching.

What you should be able to do

After completing this exercise, you should be able to:

- Code different kinds of pattern matching

Exercise instructions

__ 1. Construct and execute a program to:

__ a. Loop through standard input looking for a string matching the letters perl. Print the string when it does.

__ b. Loop through standard input looking for a string matching the following pattern: Zero or one **x**, two to four **y**, and one or more **z**. Print the string when it does. Why does the string "yyyyyz" match?

__ c. Loop through standard input looking for a string matching the letter **x** and the letter **z** in any order and ignoring case. Print the string when it does.

End of exercise

Exercise 7. String and array processing

What this exercise is about

This exercise provides an opportunity to learn about substrings.

What you should be able to do

After completing this exercise, you should be able to:

- Perform string manipulation in various ways

Exercise instructions

__ 1. Construct and execute a program to:

- __ a. Read a line of input from the keyboard.
- __ b. Print the index location of string **456**.
- __ c. Replace **456** with **abc** using the `substr` operator and print the new string.
- __ d. Replace **78** with **XY** using a regular expression and the substitution operator and print the new string.
- __ e. Using transliteration, replace **1** with **b** and **b** with **1** and print the string.

__ 2. Test the program with input containing the target strings and not containing the targets.

End of exercise

Exercise 8. Multi-dimensional and associative arrays

What this exercise is about

This exercise provides an opportunity to work with associative arrays.

What you should be able to do

After completing this exercise, you should be able to:

- Develop and print associative arrays

Exercise instructions

__ 1. Construct and execute a program to:

__ a. Create a hash using the list of precious stones from the *Lists and arrays* exercise of this course as keys and the following months as the corresponding values.

```
April
May
February
January
November
December
September
July
October
August
June
March
```

__ b. Write a loop that prompts for, and reads a key from STDIN, and then prints the corresponding value.

End of exercise

Exercise 9. User-defined subroutines

What this exercise is about

This exercise provides an opportunity to write and understand a subroutine pattern.

What you should be able to do

After completing this exercise, you should be able to:

- Execute a subroutine with parameters

Exercise instructions

__ 1. Construct and execute a program to:

- __ a. Read input from the keyboard.
- __ b. Pass the input to a subroutine that checks that the input is a number from one to 13 then prints the corresponding playing card name: **1 is ace**, **2 is two**, **13 is king**.
- __ c. Print a warning message for invalid (out of range) cards.
- __ d. Read values from standard input to test the routine.

__ 2. If time permits, construct and execute a program to:

- __ a. Validate and print a card and suit. Input should be of the form: **12S**.
- __ b. Output should be of the form: **Queen of Spades**.
- __ c. Use substrings or regular expressions to separate the card and suit. Suits are Spades, Clubs, Diamonds, or Hearts.

End of exercise

Exercise 10.File I/O

What this exercise is about

This exercise provides an opportunity to use filehandles.

What you should be able to do

After completing this exercise, you should be able to:

- Read and write filehandles and perform file tests

Exercise instructions

__ 1. Create a test text file to work with and save it as **myfilein**. It should look like:

```
Andy Andrews
A1 Aardvarks
1 Airport Avenue
Anytown, USA 12345
```

__ 2. Construct and execute a program to:

__ a. Create a new **myfileout** from **myfilein**.

__ b. Create the new file again but this time replace any **3** found with a **9**.

End of exercise

Exercise 11. Advanced flow control

What this exercise is about

This exercise provides an opportunity to experiment with advanced control structures.

What you should be able to do

After completing this exercise, you should be able to:

- Program advanced control structures with a minimum of effort on one line

Exercise instructions

__ 1. Construct and execute a program to:

__ a. Loop reading from standard input. Report if the input is greater than zero using the `&&` operator with a print statement.

__ b. Now change the print statement to print a message for positive or negative using the `?:` operator.

__ c. Then, add another `if` to check for exactly zero in which case exit out of the loop with the `last` operator.

End of exercise

Exercise 12.Dealing with files and directories

What this exercise is about

This exercise provides an opportunity to learn how Perl can manipulate directories.

What you should be able to do

At the end of this exercise, you should be able to:

- Investigate a directory

Exercise instructions

__ 1. Construct and execute a program to:

  __ a. Glob the current directory and list the files.

__ 2. Construct and execute a program to:

  __ a. Open the current directory, sort the output of `readdir` and list the files one file per line.

__ 3. Construct and execute a program to:

  __ a. Test whether `myfilein` is readable and print a message.

  __ b. Test whether `myfileout` is writable and print a message.

  __ c. Get and report the modification time on `myfileout`.

  __ d. Rename the `myfileout` file to something different.

  __ e. Change the permissions to **777** octal.

  __ f. View the results from the command line with `ls -l`.

End of exercise

Exercise 13.Running Perl

What this exercise is about

This exercise provides an opportunity to do argument passing.

What you should be able to do

At the end of this exercise, you should be able to:

- Run a one-line Perl program

Exercise instructions

__ 1. Create a one-line Perl program executed directly from the command line.

   __ a. As input, use the file called **`myfilein`** from a previous exercise. Scan for the line with the street address and print just the number part of the address.

   __ b. Debug one of your existing programs.

__ 2. Start Perl with the **`-d`** switch.

   __ a. Step through your program with the **`s`** and **`n`** commands.

   __ b. Inspect variables with the **`X`** command.

   __ c. View your code with the **`l`** and **`w`** commands.

__ 3. If time permits, add a breakpoint to your program.

End of exercise

Exercise 14.Report generation

What this exercise is about

This exercise provides an opportunity to generate formats.

What you should be able to do

After completing this exercise, you should be able to:

- Print a mail label

Exercise instructions

__ 1. Create a test text file to work with and save it as **orders**. It looks like:

```
1657814: 36: motor oil: 13:02:05: 3.49: 4111
2212468: 18: candy bars: 06:22:01: 0.55: 4214
3497524: 55: magazines: 26:06:07: 3.98: 4306
1143792: 24: wheat bread: 21:15:11: 2.29: 4186
0419665: 111: potato chips: 11:08:04: 0.99: 4186
2991276: 72: soda pop: 04:01:01: 0.55: 4214
4691324: 12: whole milk: 91:05:02: 1.29: 4299
```

__ 2. Construct and execute a program to:

__ a. Create a pick list from the **orders** file using a format like the lecture notes on the slide *A report example* and write them to an output file named **picklist**.

End of exercise

Exercise 15.Accessing operating system data

What this exercise is about

This exercise provides an opportunity to use system databases.

What you should be able to do

After completing this exercise, you should be able to:

- Interrogate the password and group files

Exercise instructions

__ 1. Construct and execute a program to:

  __ a. Get and list the fields of information in the password file for your user name.

__ 2. Construct and execute a program to:

  __ a. Get and list the fields of information in the group file for your group name.

End of exercise

Exercise 16.Running external programs

What this exercise is about

This exercise provides an opportunity to interface with the operating system.

What you should be able to do

After completing this exercise, you should be able to:

- Manipulate system commands and explain pipes

Exercise instructions

__ 1. Construct and execute a program to:

  __ a. Get a list of who is on the computer system to the display screen with the Perl system operator.

__ 2. Construct and execute a program to:

  __ a. Get a list of the key-value pairs of the environment variables.

__ 3. Construct and execute a program to:

  __ a. Get a list of who is on the system using the **`who`** command as a filehandle and print the resulting list.

__ 4. Construct and execute a program to:

  __ a. Get the date with backquotes or **`qx`** and check to see whether today is Wednesday, Thursday, or Friday. If it is, print, respectively:
  "Mid-week!", or "Are you done yet?", or "The weekend starts here!".

Note

On Windows machines, the `date` command waits for input so it cannot be used with `qx`.

End of exercise

Exercise 17.The standard modules

What this exercise is about

This exercise enables you to see what standard modules are available and use one or more of them in a simple program.

What you should be able to do

After completing this exercise, you should be able to:

- Find and use some of Perl's standard modules

Exercise instructions

__ 1. Using the commands in the unit, list the modules installed on your system. Select one of the installed modules.

__ a. Look at the source code in the file.

__ b. Check whether the module is documented with `perldoc`, if not, find another one that is.

__ c. Write a small program that uses the selected module.

End of exercise

Optional exercises

__ 1. Copy an existing program to a new file.

__ a. Add the "strict" and "diagnostics" pragmas.

__ b. Get the program to run without errors.

End of optional exercises

Exercise 18.DBI: A database interface

What this exercise is about

This exercise provides an opportunity to try out the DBI module for database access.

What you should be able to do

After completing this exercise, you should be able to:

- Use Perl and the DBI module to write database queries and updates

Introduction

There are DBD (driver) modules for a wide range of databases. Most SQL is common between most DBMS, so the programs you write here will work with any database system. Unless the instructor specifies otherwise, the DBMS used for the exercises is MySQL.

Exercise instructions

Load data from a text file into a table

__ 1. Write a program **cd_load**. The program should:

__ a. Read the file `/home/perl/cddata.txt`. Each line of the file is a `|` separated list of values that correspond to the database table `teamXX_cd` where *XX* is your team number given by the instructor.

__ b. Connect to the database with the connect string `"DBI:mysql:ipp"`.

__ 2. Insert each row into the table `teamXX_cd` using the SQL statement:

```
Insert into teamXX_cd values (?,?,?,?,?,?,?,?,?,?,?)
```

Before inserting the data, change the format of the date in the seventh field from **DD/MM/YYYY** to **YYYY-MM-DD**. Ensure that exactly **11** values are supplied for each insert.

__ a. Report any errors and continue processing, if possible.

__ b. At the end, report the number of records successfully inserted and the number of failures. Test the program. If the program loads some data, then fails, you might want to clear existing records before rerunning your program. To do this, start the MySQL monitor and delete all rows in your table:

```
$ mysql ipp

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.23.47

Type 'help' for help.

mysql> delete from teamXX_cd;
Query OK, 0 rows affected (0.00 sec)

mysql> quit
Bye
$
```

__ 3. Show that all the data has loaded by executing a select statement in the MySQL monitor:

```
$ mysql ipp

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 3.23.47

Type 'help' for help.

mysql> select id, title from teamXX_cd;
...
542 rows in set (0.01 sec)

mysql> quit
Bye
$
```

Use a report format to generate a report on the database

__ 4. Write a program **cd_rept1**. The program should:

__ a. Connect to the database with the connect string `"DBI:mysql:ipp"`.

__ b. Prepare and execute a query of: `select * from teamXX_cd order by artist`

__ c. For each row of data returned, output the following report format.

```
format STDOUT_TOP =
CD LIST Page @##
$%
-----------------------------------
.
format STDOUT =
@<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$artist
@<<< @<<<<<<<<<<<<<<<<<<<<<<<<<<
$id, $title
```

These are the fourth, first, and second columns in the table.

__ d. Finish the statement handle and disconnect from the database on completion.

__ e. Report any errors, and continue or exit as appropriate. Test the program.

```
./cd_rept1 | more
```

End of exercise

Optional exercises

Add user-supplied queries to the report program

__ 1. Copy **cd_rept1** to **cd_rept2** and change it as follows:

- If there are any command-line arguments, assume they form a condition for inclusion in the SQL statement. The select statement:

  ```
  select * from teamXX_cd order by artist
  ```

  becomes

  ```
  select * from teamXX_cd where COMMAND-LINE-ARGS order by artist
  ```

__ 2. Test the program without arguments, it should perform as before.

```
./cd_rept2 | more
```

__ 3. Test the program with arguments.

```
./cd_rept2 'artist like "%bowie%"' | more
./cd_rept2 'year(purchased) = 1997' | more
./cd_rept2 'title like "%blue%"' | more
./cd_rept2 'id < 10' | more
```

End of optional exercise

Appendix A. Exercise answers

Exercise 1 answers

ex01-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
print 'Hello World!';
```

ex01-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\n\tHello World!\n";
```

Exercise 2 answers

ex02-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
$a = 9;
$b = 1;
$c = 5;
$sum = $a+$b+$c;
print "The sum is $sum for variables of $a and $b and $c.\n";
```

ex02-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
$l = 12;
$w = 24;
$area = $l*$w;
print "The area of the rectangle is $area square feet.\n";
```

ex02-3. One possible solution for the program is:

```
#!/usr/bin/perl -w
$a = 2**0.5;
print "The square root of 2 is $a.\n";
$a = 3**2;
print "The square of 3 is $a.\n";
$a = 5*5;
print "The square of 5 is $a.\n";
$sum = 3*3+5*5;
print "The sum is $sum.\n";
```

ex02-4. One possible solution for the program is:

```
#!/usr/bin/perl -w
$s1 = 'Every good boy';
$s2 = ' does fine';
$s3 = $s1.$s2;
print "The string is: $s3.\n";
print 'The string is: ', $s1.$s2, ".\n";
```

Exercise 3 answers

ex03-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "Enter length of rectangle: ";
$l = <STDIN>;
print "Enter width of rectangle: ";
$w = <STDIN>;
$area = $l * $w;
print "The area of the rectangle is $area square feet.\n";
```

ex03-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\nPlease input string: ";
chomp($string=<STDIN>);
print "\nPlease input number: ";
chomp($number=<STDIN>);
$value = $string x $number;
print "\n\nThe resultant string is $value.\n";
```

ex03-3. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\n"x 20;
```

Exercise 4 answers

ex04-1a. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\nWhat is the elevation? ";
chomp($level = <STDIN>);
if ($level > 0)
{
print "\tabove Mean Sea Level.\n";
}
else
{
print "\tbelow Mean Sea Level!\n";
}
```

ex04-1b. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\nWhat is the elevation? ";
chomp($level = <STDIN>);
if ($level > 0)
{
print "\tabove Mean Sea Level.\n";
}
elsif ($level == 0)
{
print "\texactly Mean Sea Level.\n";
}
else
{
print "\tbelow Mean Sea Level!\n";
}
```

ex04-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\nPlease input a string 'enter to end': "; chomp($str1 = <>);
while ($str1 ne "")
{
$str2 = $str2.$str1;
print "\n$str2\n";
print "\nPlease input a string 'enter to end': ";
chomp($str1 = <>);
}
print "\n\nThen final sentence is:-\n$str2.\n";
```

Exercise 5 answers

ex05-1a. One possible solution for the program is:

```
#!/usr/bin/perl -w

@stones = ('diamaond', 'emerald', 'amethyst', 'garnet', 'topaz',
'turquoise', 'saphire', 'ruby', 'opal', 'peridot', 'pearl',
'aquamarine');
```

ex05-1b. One possible solution for the program is:

```
#!/usr/bin/perl -w

@stones = ('diamaond', 'emerald', 'amethyst', 'garnet', 'topaz',
'turquoise', 'saphire', 'ruby', 'opal', 'peridot', 'pearl',
'aquamarine');

print "\n".'@stones unquoted:'."\n";
print @stones;

print "\n".'@stones quoted:'."\n";
print "@stones";

# To print the elements of @stones with spaces seperating use the
# following:
# print join(' ',@stones), "\n";

print "\n\nSingle element access:\n";
print "$stones[0], $stones[7] and $stones[10] \n";

print "\nPrecious stones beginning with vowels:\n";
print @stones[1,2,8,11], "\n\n";
```

ex05-1c. One possible solution for the program is:

```
#!/usr/bin/perl -w

@stones = ('diamaond', 'emerald', 'amethyst', 'garnet', 'topaz',
'turquoise', 'saphire', 'ruby', 'opal', 'peridot', 'pearl',
'aquamarine');

print "\n".'@stones unquoted:'."\n";
print @stones;

print "\n".'@stones quoted:'."\n";
print "@stones";

# To print the elements of @stones with spaces seperating use the
# following:
# print join(' ',@stones), "\n";

print "\n\nSingle element access:\n";
print "$stones[0], $stones[7] and $stones[10] \n";

print "\nPrecious stones beginning with vowels:\n";
print @stones[1,2,8,11], "\n\n";

print "Sorted list of stones on seperate lines:\n";
@sorted_stones = sort(@stones);
foreach $gem (@sorted_stones)
{
print "$gem\n";
}
```

Exercise 6 answers

ex06-1a. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\nPlease input a string: ";
while ($perl=<STDIN>)
{
chomp($perl);
if ($perl =~ /perl/)
{
print "\n\t\tString \"${perl}\"matched!";
exit;
}
print "\n\nPlease input a string: ";
}
```

ex06-1b. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\nPlease input a string: ";
while (<STDIN>)
{
chomp;
if (/x?y{2,4}z+/)
{
print "\n\t\tString $_ matched!";
exit;
}
print "\nPlease input a string: ";
}
```

ex06-1c. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\nPlease input a string: ";
while (<>)
{
if (/x/i && /z/i)
{
chomp;
print "\n\tString $_ matched!";
exit;
}
print "\n\nPlease input a string: ";
}
```

Exercise 7 answers

ex07-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\nPlease input the string \"0123456789\": ";
chomp($str = <>);
$idx = index($str,"456");
print "\n\nThe starting location of 456 is $idx.";
substr($str,$idx,3) = 'abc';
print "\n\nThe new string is $str.";
$str =~ s/78/XY/;
print "\n\nThe new string is $str.";
$str =~ tr/1b/b1/;
print "\n\nThe new string is $str.\n\n";
```

Exercise 8 answers

ex08-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
%birthstones=(diamond => 'April',
emerald => 'May',
amethyst => 'February',
garnet => 'January',
topaz => 'November',
turquoise => 'December',
sapphire => 'September',
ruby => 'July',
opal => 'October',
peridot => 'August',
pearl => 'June',
aquamarine => 'March');

print "\nPlease input a precious stone: ";
while ($stone = <>)
{
chomp($stone);
print "\nThe stone $stone corresponds to the month of
$birthstones{$stone}.\n";
print "\nPlease input a precious stone: ";
}
```

Exercise 9 answers

ex09-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\n\n\nPlease input the numeric ID for a playing card: ";
while (<>)
{
chomp;
$choice = &card($_);
if ($_ > 0 && $_ < 14)
{
print "\nCard number of $_ is idendified as $choice.";
}
else
{
print "\nCard number of $_ is out of range.";
}
print "\n\n\nPlease input the numeric ID for a playing card: ";
}

# The following subprogram takes a number and reports the name of that
# number for a deck of playing cards. If the number is out of range
# it will return the out of range value.
sub card
{
my($nmbr) = @_;
@cards = ('0','Ace','Two','Three','Four','Five','Six','Seven',
'Eight','Nine','Ten','Jack','Queen','King');
if ($cards[$nmbr])
{
$cards[$nmbr];
}
else
{
$nmbr;
}
}
```

ex09-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
print "\n\n\nPlease input the number and suit of a playing card: ";
while (<>)
{
chomp;
/\d+/; # \d means digit and + means 1 or more
$input_num = $&; # $& means match
$input_card = &card($input_num);
$input_lettr = uc($'); # $' is string right of match
$input_suit = &suit($input_lettr);
if ($input_num > 0 && $input_num < 14)
{
if ($input_lettr eq 'S' || $input_lettr eq 'C' ||
$input_lettr eq 'D' || $input_lettr eq 'H')
{
print "\nThe card is the $input_card of $input_suit.";
}
else
{
print "\nCard suit of $input_lettr is out of range.";
}
}
else
{
print "\nCard number of $input_num is out of range.";
}
print "\n\n\nPlease input the number and suit of a playing card: ";
}

# The following subprogram takes a number and reports the name of that
# number for a deck of playing cards. If the number is out of range
# it will return the out of range value.
sub card
{
my($nmbr) = @_;
@cards = ('0','Ace','Two','Three','Four','Five','Six','Seven',
'Eight','Nine','Ten','Jack','Queen','King');
if ($cards[$nmbr])
{
$cards[$nmbr];
}
else
{
$nmbr;
}
}
```

```
# The following subprogram takes a suit S, C, D or H and returns the
# name of the suit. ie. Spades, Clubs, Diamonds or Hearts.
sub suit
{
my($lttr) = @_;
%suit = (S => 'Spades', C => 'Clubs', D => 'Diamonds', H => 'Hearts');
$suit{$lttr};
}
```

Exercise 10 answers

ex10-1. One possible solution for the program is:

```
# This file already exists in /home/perl/data/myfilein

Andy Andrews
A1 Aardvarks
1 Airport Avenue
Anytown, USA 1234
```

ex10-2a. One possible solution for the program is:

```
#!/usr/bin/perl -w
$getfrom = "/home/perl/data/myfilein";
$sendto = "myfileout";

open(MYFILEIN, $getfrom) || die "Cannot open $getfrom to read";
open(MYFILEOUT, ">$sendto") or die "Cannot open $sendto to write";

while (<MYFILEIN>)
{
print MYFILEOUT;
}

close(MYFILEIN);
close(MYFILEOUT);
```

ex10-2b. One possible solution for the program is:

```
#!/usr/bin/perl -w
$getfrom = "/home/perl/data/myfilein";
$sendto = "myfileout2";

open(MYFILEIN, $getfrom) || die "Cannot open $getfrom to read";
open(MYFILEOUT, ">$sendto") or die "Cannot open $sendto to write";

while (<MYFILEIN>)
{
s/3/9/g;
print MYFILEOUT;
}

close(MYFILEIN);
close(MYFILEOUT);
```

Exercise 11 answers

ex11-1a. One possible solution for the program is:

```
#!/usr/bin/perl -w

while (<>)
{
chomp;
$_ > 0 && print "$_ is greater than zero.\n";
}
```

ex11-1b. One possible solution for the program is:

```
#!/usr/bin/perl -w

while (<>)
{
chomp;
$_ == 0 && last;
$_ > 0 ? print "$_ is greater than 0.\n": print "$_ is less than 0.\n";
}
```

Exercise 12 answers

ex12-1. One possible solution for the program is:

```
#!/usr/bin/perl -w

foreach (<*>)
{
print "$_\n";
}
```

ex12-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
opendir(DIR,".") || die "Could not open current directory.\n";

foreach (sort (readdir(DIR)) )
{
print "$_\n";
}

closedir(DIR);
```

ex12-3a. One possible solution for the program is:

```
#!/usr/bin/perl -w
$getfrom = "/home/perl/data/myfilein";
-r $getfrom && print "File $getfrom is readable\n";
ex12-3b. One possible solution for the program is:
#!/usr/bin/perl -w
$sendto = "myfileout";
-w $sendto and print "File $sendto is writable\n";
```

ex12-3c. One possible solution for the program is:

```
#!/usr/bin/perl -w
$sendto = "myfileout";
$modtime = ( stat($sendto) )[9];
print "Modification time is $modtime\n";
ex12-3d. One possible solution for the program is:
#!/usr/bin/perl -w
$old = 'myfileout';
$new = 'myfileout_old';
rename($old, $new);
```

ex12-3e. One possible solution for the program is:

```
#!/usr/bin/perl -w
$new = 'myfileout_old';
chmod (0777, $new);
```

Exercise 13 answers

ex13-1. One possible solution for the program is:

```
# This is a shell script which executes a Perl command
perl -ane '/Airport/ and print $F[0], "\n";' < /home/perl/data/myfilein
```

ex13-2. One possible solution for the program is:

```
# This is a shell script which executes a Perl command
perl -d ex09-1
```

Exercise 14 answers

ex14-1. One possible solution for the program is:

```
#/usr/bin/perl -w
# This file has already been created for you in /home/perl/data/orders
print "The file has already been created in /home/perl/data/orders.\n";
ex14-2. One possible solution for the program is:
#!/usr/bin/perl -w
$ordrs = "/home/perl/data/orders";
$pcklst = "picklist";

open(ORDERS, $ordrs) || die "Cannot open $ordrs.\n";
open(PICKLIST, ">$pcklst") || die "Cannot create $pcklst.\n";

while (<ORDERS>)
{
chomp;
($stock, $quan, $desc, $row, $col, $bin, $price, $store) = split(/:/);
write PICKLIST;
}

format PICKLIST =

stock#: @<<<<<< quantity: @<<
$stock, $quan
@<<<<<<<<<<<<<<<<<<<<<<<<<<<
$desc
row: @<< col: @< bin: @<
$row, $col, $bin
price: @<<<< store#: @<<<
$price, $store

.
```

Exercise 15 answers

ex15-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
$\ = $, = "\n";
($name, $passwd, $uid, $gid, $quota, $comment, $gcos,
 $dir, $shell) = getpwnam("team01");
print $name;
print $passwd;
print $uid;
print $gid ;
print $quota;
print $comment;
print $gcos;
print $dir;
print $shell;

.
```

ex15-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
$\ = "\n"; # defines separator between print statements
($gname, $gpasswd, $gid, $gmembers) = getgrnam("staff");

print $gname;
print $gpasswd;
print $gid;
print $gmembers;
```

Exercise 16 answers

ex16-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
system("who");
```

ex16-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
foreach $k (keys %ENV)
{
print "$k = $ENV{$k}\n"; #This prints "VARNAME = VARVALUE"
}
```

ex16-3. One possible solution for the program is:

```
#!/usr/bin/perl -w
open(WHO, "who|") || die "Cannot open the who pipe to the system.\n";

while (<WHO>)
{
print;
}

close(WHO);
```

ex16-4. One possible solution for the program is:

```
#!/usr/bin/perl -w
`date` =~ /Wed/ && print "\n\t Mid-week! \n\n";
`date` =~ /Thu/ and print "\n\t Are you done yet? \n\n";
qx/date/ =~ /Fri/ && print "\n\t The weekend starts here! \n\n";
```

Exercise 17 answers

ex17-1.bat One possible solution for the program is:

```
# On Windows Systems use:
perl -e "print join qq(\n), @INC, ''"
dir `perl -e 'print "@INC"'`
```

ex17-1.ksh. One possible solution for the program is:

```
#!/bin/ksh
# for UNIX-like systems
print "\nList of Directories in \@INC:";
perl -e 'print join "\n", @INC, ""'

print "\nList of Modules in the Directories in \@INC:";
#ls `perl -e 'print "@INC"'`
perl -e '$,=$\="\n"; foreach (@INC) {print <$_/*.p[lm]>}'

print "\nContents of Cwd.pm:";
cat /usr/opt/perl5/lib/5.10.1/aix-thread-multi/Cwd.pm
ex17-2. One possible solution for the program is:
#!/usr/bin/perl -w
use Cwd;
$pwd = cwd;
print "Current Working Directory is:\t$pwd\n";
```

Exercise 18 answers

ex18-1. One possible solution for the program is:

```
#!/usr/bin/perl -w
# This script will insert entries into your table mysql:ipp:teamXX_cd
# from a file containing 542 CD Titles
#
# 1. $ /home/teamXX/ex18-1 /home/perl/data/cddata.txt
# You should get confirmation of 542 rows being successfully added.
#
# Should you need to remove the 542 entries and start over for any
# reason you can run the script ex18-cldb:
# ie. $ ex18-cldb

use DBI;

if (scalar(@ARGV) != 1) {
print "ERROR: Invalid number of command line arguments\n";
print "\tUSAGE: $0 /home/perl/data/cddata.txt\n";
exit;
}

my $db = "DBI:mysql:ipp;host=localhost";

my $dbh = DBI->connect($db) or
die "$0: can't connect to $db: $DBI::errstr\n";

my $sth = $dbh->prepare( <<END_SQL ) or die "$0: can't prepare:
$DBI::errstr\n";
insert into $ENV{LOGNAME}_cd values (?,?,?,?,?,?,?,?,?,?,?)
END_SQL

$good = $bad = 0;

while (<>) {
chomp;
@cd = split /\|/;

# fix the date
$cd[6] = join '/', reverse split '/', $cd[6];

if ( $sth->execute( @cd[0..10] ) ) { # ensure exactly 11 fields
$good ++;
}
else {
warn "$0: can't insert $cd[0], $cd[1]: $DBI::errstr\n";
$bad ++;
}
}
```

```
$dbh->disconnect;

print "Added $good rows in ", time - $^T, "seconds. $bad failures.\n";
```

ex18-2. One possible solution for the program is:

```
#!/usr/bin/perl -w
# This script will use a report format to generate a report on your
# database mysql:ipp:teamXX_cd
#
# 1. $ /home/teamXX/ex18-2 | more
# You should get 542 entries in report format output.
#
# Should you need to remove the 542 entries and start over for any
# reason you can run the script ex18-cldb:
# ie. $ ex18-cldb

use DBI;

my $db = "DBI:mysql:ipp;host=localhost";

my $dbh = DBI->connect($db) or
die "$0: can't connect to $db: $DBI::errstr\n";

my $sth = $dbh->prepare( <<END_SQL ) or die "$0: can't prepare:
$DBI::errstr\n";
select * from $ENV{LOGNAME}_cd
order by artist
END_SQL

$sth->execute or die "$0: can't execute select: $DBI::errstr\n";

while ( @cd = $sth->fetchrow_array ) {
write;
}
$sth->finish;
$dbh->disconnect;

format STDOUT_TOP =
CD LIST Page @<<
$%
------------------------------------------------------------------------
.
format STDOUT =
@<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$cd[3]
@<<< @<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$cd[0],$cd[1]
.

# end
```

ex18-3. One possible solution for the program is:

```
#!/usr/bin/perl -w
# This script will allow you to add user-supplied queries to the report
# format and generate reports on your database (mysql:ipp:teamXX_cd)
#
# 1. $ /home/teamXX/ex18-3 | more
# You should get 542 entries in report format output same as ex18-2.
#
# Additional Queries:
#
# 2. $ /home/teamXX/ex18-3 'artist like "%bowie%"' | more
# 3. $ /home/teamXX/ex18-3 'year(purchased) = 1997' | more
# 4. $ /home/teamXX/ex18-3 'title like "%blue%"' | more
# 5. $ /home/teamXX/ex18-3 'id < 10' | more
#
# Should you need to remove the 542 entries and start over for any
# reason you can run the script ex18-cldb:
# ie. $ ex18-cldb

use DBI;

my $db = "DBI:mysql:ipp;host=localhost";

my $dbh = DBI->connect($db) or
die "$0: can't connect to $db: $DBI::errstr\n";

my $query = "select * from $ENV{LOGNAME}_cd ";
$query .= 'where ' . shift if @ARGV;
$query .= ' order by artist';
my $sth = $dbh->prepare( $query ) or die "$0: can't prepare: $DBI::errstr\n";

$sth->execute or die "$0: can't execute select: $DBI::errstr\n";

while ( @cd = $sth->fetchrow_array ) {
write;
}
$sth->finish;
$dbh->disconnect;

format STDOUT_TOP =
CD LIST Page @<<
$%
------------------------------------------------------------------------------
.
format STDOUT =
@<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$cd[3]
```

```
@<<< @<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$cd[0],$cd[1]
.
# end
```

ex18-cldb. One possible solution for the program is:

```
#!/usr/bin/perl -w
# remove all entries from the database table

use DBI;

print "\nYou are about to clear all entries from the $ENV{LOGNAME}_cd DB
table\n";

do {
print "Do you wish to continue? (y/n)\n"; $_ = <STDIN>; chomp;
}
while ( $_ !~ /^[yn]/i) ;

if (/^n/i) {
print "Command Aborted!\n";
exit;
}
elsif (/^y/i) {
print "Proceeding to remove DB table entries\n";
}

# $host=`hostname`;
# my $db = "DBI:mysql:ipp;host=$host";
my $db = "DBI:mysql:ipp;host=localhost";

my $dbh = DBI->connect($db) or
die "$0: can't connect to $db: $DBI::errstr\n";

my $sth = $dbh->prepare( <<END_SQL ) or die "$0: can't prepare:
$DBI::errstr\n";
delete from $ENV{LOGNAME}_cd
END_SQL

$sth->execute or die "$0: can't execute delete: $DBI::errstr\n";

$sth->finish;
$dbh->disconnect;
```

Appendix B. Using CGI.pm

What this exercise is about

This exercise is for practicing use of the CGI.pm module.

What you should be able to do

After completing this exercise, you should be able to:

- Write simple CGI programs using Perl to generate the HTML
- Write programs that generate forms and process the returned values
- Install the programs
- Debug CGI programs

Exercise instructions

Write, test, and install a simple CGI program

__ 1. Copy the example from the page "CGI.pm Example: O-O Interface" into the file `simple`.

__ 2. Test the program from the command-line to ensure that it compiles without errors or warnings.

__ 3. Test the program with and without parameters and check that the HTML produced looks reasonable.

__ 4. Install the program in the **`cgi-bin`** directory specified by your instructor, with permissions that allow the program to be run by the web server.

__ 5. Test the program from a web browser using the URL `http://<your-machine>/cgi-bin/team<XX>/simple` .

Write a program that uses cookies

__ 6. Write a program called **buttons** that produces a form that asks for:

- A personal name.
- A preferred background color from the list: white, red, orange, yellow, green, blue, violet, black.
- The range of their age (<=20, 21-30, 31-40, 41-50, 51-60, >60).

__ 7. Store the values in cookies and return them to the client in a "Thank You" page.

__ 8. Test the program from the command-line to ensure that it compiles without errors or warnings.

__ 9. Test the program with and without parameters and check that the HTML produced looks reasonable.

__ 10. Install the program in the **`cgi-bin`** directory specified by your instructor, with permissions that allow the program to be run by the web server.

__ 11. Test the program from a web browser using the web address: `http://<your-machine>/cgi-bin/team<XX>/buttons` .

__ 12. Check the cookies file of your browser to determine if the cookies exist.

End of exercise

Optional exercises

Extend the previous program to retrieve cookie values

__ 1. Extend the **buttons** program to retrieve the cookies. Greet the user by name if the personal name value exists. Use the preferred color instead of the default value if the background color was specified.

End of optional exercise

Appendix B answers

Write, test, and install a simple CGI program

1. This is the simple oo-CGI-Example from figure 19-10.

```
#!/usr/bin/perl -w

use CGI;
my $q = new CGI;
$| = 1;
print $q->header, $q->start_html(-title => 'CGI Simple Form (OO)',
-meta => {keywords => 'simple form'}, -BGCOLOR => 'white');
my $name;
if ($name = $q->param('Name') ) { # input from previous run
print $q->h1("Welcome back $name!"), $q->hr,
"The time here is: ", scalar localtime, $q->p,
"Parameters are: ", $q->Dump;
}
else { # no input, send a form
print $q->h1("Hello stranger!"), $q->hr,
$q->start_form(-method => 'post'), 'Please enter your name: ',
$q->textfield(-name => 'Name', -size => 20, -maxlength => 50),
$q->p, $q->submit, $q->end_form;
}
print $q->p, $q->hr, $q->end_html;
```

Write a program that uses cookies

1. One possible solution to the exercise is:

```
#!/usr/bin/perl -w
# a simple form
use CGI;
use strict;
use diagnostics;
my $q = new CGI;
my $name;
if ( $name = $q->param('Name') ) { # some input, assume fields filled
my @std_params = qw(-expires +1y -path /cgi-bin/cookie
-domain .ibm.com -secure 0);
my $namec = $q->cookie( -name => 'Name', -value => $name, @std_params);
my $agec = $q->cookie( -name => 'Age',
-value => $q->param('Age'), @std_params);
my $bgcolorc = $q->cookie( -name => 'bgcolor',
-value => $q->param('bgcolor'), @std_params);
print $q->header( -cookie => [$namec, $agec, $bgcolorc] );
print $q->start_html(-title => 'Thank You!',
-meta => {keywords => 'button test form'},
-BGCOLOR => $q->param('bgcolor') ? $q->param('bgcolor') : 'white');
print $q->h1("Welcome back $name!"), $q->hr,
"Thank you for completing the preferences form."
}
else { # no input, send a form
print $q->header; # no cookies yet!
print $q->start_html(-title => 'Cookie Test',
-meta => {keywords => 'Cookie test form'},
-BGCOLOR => $q->param('bgcolor') ? $q->param('bgcolor') : 'white');
print $q->h1('Hello stranger!'), $q->hr,
$q->start_form(-method => 'post'),
'Please enter your name: ',
$q->textfield(-name => 'Name', -size => 20, -maxlength => 50),

$q->p, 'Select a background colour ',
$q->popup_menu(-name => 'bgcolor', -values => [
qw/white red orange yellow green blue violet black/ ],
-default => 'white',),

$q->p, 'Please select your age ',
$q->radio_group( -name => 'Age',
-values => [1,2,3,4,5,6], -rows => 2,
-labels => {
1 => '<=20', 2 => '21-30', 3 => '31-40',
4 => '41-50', 5 => '51-60', 6 => '>60' }),
$q->p, $q->submit, $q->end_form;
}
```

```
print $q->hr, $q->end_html;
exit 0;
```

Extend the previous program to retrieve cookie values

1. One possible solution to the exercise is:

```
#!/usr/bin/perl -w
# a simple form
use CGI;
use strict “vars”;
use diagnostics;
my $q = new CGI;
my ($name, $bgcolor, $age);
if ( $name = $q->param(‘Name’) ) { # some input, assume fields filled
my @std_params = qw(-expires +1y -path /cgi-bin
-secure 0);
my $namec = $q->cookie( -name => ‘Name’, -value => $name, @std_params);
my $agec = $q->cookie( -name => ‘Age’,
-value => $q->param(‘Age’), @std_params);
my $bgcolorc = $q->cookie( -name => ‘bgcolor’,
-value => $q->param(‘bgcolor’), @std_params);
print $q->header( -cookie => [$namec, $agec, $bgcolorc] );
print $q->start_html(-title => ‘Thank You!’,
-meta => {keywords => ‘button test form’},
-BGCOLOR => $q->param(‘bgcolor’) ? $q->param(‘bgcolor’) : ‘white’);
print $q->h1(“Welcome back $name!”), $q->hr,
“Thank you for completing the preferences form.”
}
else { # no input, send a form
# get values from cookies or set defaults
$name = $q->cookie(‘Name’) ? $q->cookie(‘Name’) : ‘’;
$bgcolor = $q->cookie(‘bgcolor’) ? $q->cookie(‘bgcolor’) : ‘white’;
$age = $q->cookie(‘Age’) ? $q->cookie(‘Age’) : ‘1’;

print $q->header;
print $q->start_html(-title => ‘Cookie Test’,
-meta => {keywords => ‘Cookie test form’},
-BGCOLOR => $bgcolor );

if ($name) {
print $q->h1(“Hello $name!”), $q->hr,
‘Please check your data: ‘;
}
else {
print $q->h1(‘Hello stranger!’), $q->hr,
‘Please enter your name: ‘;
}
print $q->start_form(-method => ‘post’),
```

```
$q->textfield(-name => 'Name', -size => 20, -maxlength => 50,
-default => $name),

$q->p, 'Select a background colour ',
$q->popup_menu(-name => 'bgcolor', -values => [
qw/white red orange yellow green blue violet black/ ],
-default => $bgcolor),

$q->p, 'Please select your age ',
$q->radio_group( -name => 'Age',
-values => [1,2,3,4,5,6], -rows => 2,
-default => $age,
-labels => {
1 => '<=20', 2 => '21-30', 3 => '31-40',
4 => '41-50', 5 => '51-60', 6 => '>60' }),
$q->p, $q->submit, $q->end_form;
}
print $q->hr, $q->end_html;
exit 0;
```

Appendix C. Creating graphics with Perl

What this exercise is about

This exercise is for practicing the use of the GD::Graph module.

What you should be able to do

After completing this exercise, you should be able to:

- Created business graphics using data retrieved from a database
- Display the data in a web page

Exercise instructions

Create a graphic

__ 1. Write a program called `pie.pl` that creates a pie chart by using the values:

```
@data = (
[1990 .. 1997],
[qw/37 74 59 74 86 92 59 22/],
);
```

__ 2. Use the documentation for GD::Graph (`perldoc GD::Graph`) to look up the options to add titles.

__ 3. Output the image to a file and check the results by opening the file with a web browser. You can also try `"pie.pl | display -"`, if the display tool is available.

Create a graphic using data from a database

__ 4. Change the previous program to get its data from the `team`*`XX`*`_cd` using the SQL query:

```
select year(purchased) as y, count(*) from teamXX_cd group by y
```

The data is returned as pairs of year/count values so the data need manipulation to get it into the required format.

__ 5. Output the image to a file and check the results by opening the file with a web browser.

Create a web graphic by using data from a database

__ 6. Change the previous program into a CGI program.

__ 7. Test the program from the command-line to ensure that it compiles and runs without errors or warnings.

__ 8. Install the program in the **`cgi-bin`** directory specified by the instructor, with permissions that allow the program to be run by the web server.

__ 9. Test your program from a web browser using:
`http://<server-machine>/cgi-bin/team<XX>/pie.pl`

Change the web graphic and add a legend

__ 10. Change the previous program to get bar charts (use GD::Graph::bars).

__ 11. Use the documentation to find out how to add a legend to the chart.

__ 12. Install and test your program.

End of exercise

Optional exercises

Create other graphics from the database

__ 1. Create any other graphics that you can create by using data in the database. Possible queries include:

```
SELECT publication_year AS y, count(*) FROM teamXX_cd GROUP BY y

SELECT YEAR(purchased) AS x, MONTH(purchased) AS y,
COUNT(*) FROM teamXX_cd GROUP BY x, y

SELECT YEAR(purchased)*100+MONTH(purchased) AS x,
COUNT(*) FROM teamXX_cd GROUP BY x ORDER BY purchased
```

End of optional exercise

Appendix C answers

__ 1. One possible solution to the exercise is:

```
#!/usr/bin/perl -w

use GD::Graph::pie;

@data = (
[ 1990 .. 1997 ],
[ qw/37 74 59 74 86 92 59 22/ ]
);

# draw the graph
$my_graph = new GD::Graph::pie();
$my_graph->set(
title => 'CD Purchases by Year',
label => 'Copyright (C) 2000, Chris Benson',
axislabelclr => 'black',
pie_height => 36,
);
my $gdo = $my_graph->plot( \@data );

# and output the image to a file
open IMG, ">$0.png"or die "can't open $0.out: $!\n";
print IMG $gdo->png;
close IMG;

# or print the image to stdout
#print IMG $gdo->png;
One possible solution to the exercise is:
#!/usr/bin/perl

use DBI;
use GD::Graph::pie;

# get some data
my $sql = "select year(purchased) as y, count(*)
from team00_cd group by y";
my $dbh = DBI->connect("DBI:mysql:ipp", "team00") or
die "$0: can't connect to MySQL: $DBI::errstr\n";
my $sth = $dbh->prepare($sql) or
die "$0: can't prepare statement: $sql: $DBI::errstr\n";
$sth->execute or
die "$0: can't execute statement: $DBI::errstr\n";
my $cd_ref = $sth->fetchall_arrayref() or
die "$0: can't fetch data: $DBI::errstr\n";
```



```
$sth->finish;
$dbh->disconnect;

# separate labels and values
foreach ( @$cd_ref ) {
push @{$data[0]}, $_->[0];
push @{$data[1]}, $_->[1];
}

# draw the graph
$my_graph = new GD::Graph::pie();
$my_graph->set(
title => 'CD Purchases by Year',
label => 'Copyright (C) 2000, Chris Benson',
axislabelclr => 'black',
pie_height => 36,
);
my $gdo = $my_graph->plot( \@data );
# and output the image to a file
open IMG, ">$0.png"or die "can't open $0.out: $!\n";
print IMG $gdo->png;
close IMG;

# or print the image to stdout
#print IMG $gdo->png;
One possible solution to the exercise is:
#!/usr/bin/perl

use CGI qw/:standard *table/;
use CGI::Carp 'fatalsToBrowser';
$CGI::POST_MAX = 1024;
$CGI::DISABLE_UPLOADS = 1;
use DBI;
use GD::Graph::bars;

# get some data
my $sql = "select year(purchased)*100+month(purchased) as x, count(*)
from team00_cd where music_type != 'CLASSICAL'
group by x order by purchased";
my $dbh = DBI->connect("DBI:mysql:ipp", "team00") or
die "$0: can't connect to MySQL: $DBI::errstr\n";
my $sth = $dbh->prepare($sql) or
die "$0: can't prepare statement: $sql: $DBI::errstr\n";
$sth->execute or
die "$0: can't execute statement: $DBI::errstr\n";
my $cd_ref = $sth->fetchall_arrayref() or
die "$0: can't fetch data: $DBI::errstr\n";
$sth->finish;
```

```
# and store it as yymm[1] => count
foreach (@$cd_ref) {
$tmp{$_->[0]}[1] = $_->[1];
}
# get some more data
$sql = "select year(purchased)*100+month(purchased) as x, count(*)
from team00_cd where music_type = 'CLASSICAL'
group by x order by purchased";
$sth = $dbh->prepare($sql) or
die "$0: can't prepare statement: $sql: $DBI::errstr\n";
$sth->execute or
die "$0: can't execute statement: $DBI::errstr\n";
$cd_ref = $sth->fetchall_arrayref() or
die "$0: can't fetch data: $DBI::errstr\n";
$sth->finish;
$dbh->disconnect;
# and store it as yymm[2] => count
foreach (@$cd_ref) {
$tmp{$_->[0]}[2] = $_->[1];
}

# invert the data
foreach (sort keys %tmp) {
push @{$data[0]}, $_;
push @{$data[1]}, $tmp{$_}[1];
push @{$data[2]}, $tmp{$_}[2];
}

# draw the graph
$my_graph = new GD::Graph::bars(800,600);
$my_graph->set(
title => "CD Purchases by Month",
overwrite => 2,
long_ticks => 1,
b_margin => 20,
box_axis => 1,
line_types => [1,2],
x_labels_vertical => 1,
x_label_position => 1/2,
y_label => '# CDs',
y_label_skip => 1,
);
$my_graph->set_legend( 'other', 'classical');
# send the HTTP header
print header(-type=>'image/png',-expires=>'+10');
my $gdo = $my_graph->plot( \@data );
print $gdo->png;
exit 0;
```

Appendix D. Embedding Perl in Apache

What this exercise is about

This exercise is for experimenting with programs running under mod_perl: to see whether your programs run and what changes might be necessary or appropriate.

What you should be able to do

After completing this exercise, you should be able to:

- Run programs under Apache/mod_perl
- Locate warning and error messages
- Change programs to run better in the Apache::Registry environment

Introduction

This exercise usually involves a lot of troubleshooting. It is important to check that all steps have been completed before continuing. If a Perl program does not run from the command line, it will not work as a CGI program. If a program does not work in the CGI environment, it will not work in the Apache::Registry environment.

This exercise may work differently if you are sharing a web server with other attendees. Coordinate with other people using the server if it is shared.

Exercise instructions

Check Apache configuration

__ 1. Find the Apache configuration files.

__ a. Check that the Apache::Registry module is loaded. If it is not loaded, add the line and restart the server. Check with the instructor for details.

__ b. Check which locations have `PerlHandler Apache::Registry` enabled. (If there is none, add:

```
Alias /perl/ "/home/httpd/mod_perl/"
<Location /perl>
SetHandler perl-script
PerlHandler Apache::Registry
Options +ExecCGI
</Location> to the configuration and restart the server).
```

__ c. Check that the server is running with `ps -ef`.

Install and test existing programs

__ 2. Choose one of your existing CGI programs. Add `use strict;` and `-w`. Ensure that it runs from the command line *cleanly* without warning messages.

__ 3. Install the program in the location identified above for programs to be run in the Apache::Registry environment, giving the program the relevant permissions.

__ 4. Test the program using the URL for the mod_perl version of the program, for example: `http://<web-server>/perl/<your-program>`.

__ 5. Check the Apache log files for warnings, errors or both.

End of exercise

Optional exercises

Install and test more programs

__ 1. Convert, install, and test your other CGI programs.

__ 2. Try comparing speed of execution as a CGI program and under Apache::Registry.
Note: This is difficult to do accurately without a scriptable browser. `$^T` holds the current time which can be checked at the start and end of a program, but most of the performance benefit comes outside this: forking a copy of Perl and compiling the program.

End of optional exercise