Appendix A

Perl Programming

Languages like C and C++ allow a programmer to write code at a very detailed level which has good execution speed. But in many applications, one would prefer to write a program with a higher level language that makes it easy to crank out code because the programmer is completely freed from mundane tasks like managing memory and checking bounds, and can use extremely simple constructs for string comparison, manipulation, iterators, etc. For example, for text manipulation applications, the basic unit in C/C++ is a character, while for languages like Perl and Python the basic units are lines of text and words within lines. One can work with lines and words in C/C++, but he or she must go to greater effort to accomplish the same thing. C/C++ may usually give better speed, but if speed is not an issue, the convenience of a high level scripting language is very attractive. A *scripting language* is a programming language that supports the writing of scripts – programs written for a special runtime environment that can interpret and automate the execution of tasks which could alternatively be executed one-by-one by a human operator.

Among several scripting languages such as Python, Perl, PHP, Javascript, etc., Perl is an ideal text manipulation language. It was originally developed for text manipulation, but is now also used for a wide range of tasks including system administration, web development, network programming, and GUI development. Perl is a family of high-level, general-purpose, interpreted, dynamic programming languages. The languages in this family include Perl 5 and Perl 6. Perl was originally developed by Larry Wall in 1987 as a general-purpose UNIX scripting language to make report processing easier. Since then, it has undergone many changes and revisions. The latest major stable revision of Perl 5 is 5.16, released in May 2012. Perl 6 is a complete redesign of the language, announced in 2000 and still under active development as of 2013. Perl borrows features from other programming languages including C, shell scripting, and so on.

ActivePerl, developed by a Canadian software company ActiveState®, is the leading commercial-grade distribution of the open source Perl scripting language. Both win32 and win64 are supported. It can be downloaded from ActiveState® web site free of charge under the ActiveState Community License. Installation of ActivePerl is easy, simply accepting all default settings. The installation wizard will usually add `Perl\bin`

path to your computer's PATH environment automatically. If it failed to do so for some reason, you will have to add the path yourself to ensure Windows operating system can find it.

Perl is a feature-rich language, which clearly cannot be discussed in full detail here. Instead, our goal in this appendix is to help readers quickly become proficient at writing simple Perl programs to manipulate text strings and automate the execution of tasks. Readers can also refer to many excellent online tutorials and published books for detail on Perl programming.

Our first example is assigning a text string to a variable and then performing several string manipulation tasks:

```
1    $myString = "He is a programmer.";
2    print "$myString\n";
3
4    # Replace "He" by "She"
5    $myString = reverse $myString;
6    chop $myString;
7    chop $myString;
8    $myString = reverse $myString;
9    $myString = "She$myString";
10   print "$myString\n";
11
12   # Make the string all lower case
13   $myString =~ tr/A-Z/a-z/;
14   print "$myString\n";
15
16   # Make the string all upper case
17   $myString =~ tr/a-z/A-Z/;
18   print "$myString\n";
```

It is noted that the name of variable begins with $ and a data type of the variable is not declared explicitly in Perl. This is because most scripting languages are intended to be very fast to pick up. This generally implies relatively simple syntax and semantics. At line 1, we simply assign the text string "He is a programmer." to the variable $myString. We then print the text at the command line, the syntax of which is very similar to C language. For printing the output, we could simply write

```
print $myString;
```

In this case, however, there is no end-of-line or newline marker to signify the end of a line of text. Accordingly, the next print statement will print the string at the same line without the word wrap.

From line 4 to 10, "He" is replaced by "She." This is achieved by first reversing the string, then chopping out two ending characters "H" and "e" that stands for "He," reversing the string again, and concatenating the string with "She" for printing out

"She is a programmer." From line 12 to 14, the string is converted to lower case before being printed at the command line. From line 16 to 18, the string is converted to the upper case before being printed.

We now save the above Perl script in `D:\ArtProgram\bin` as `example_1.pl` and invoke it in the same directory to display the result at the command line:

```
D:\ArtProgram\bin>Perl example_1.pl
He is a programmer.
She is a programmer.
she is a programmer.
SHE IS A PROGRAMMER.
```

Saving a Perl script in the default suffix has two drawbacks. Firstly, we have to type `Perl` before the name of a script whenever we invoke it. Secondly, we need to type the full path of the script if it is invoked in a different directory. For example, we have to type

```
D:\ArtProgram\test>Perl D:\ArtProgram\bin\example_1.pl
```

if we are currently not in the `D:\ArtProgram\bin` directory. As is known, Windows command prompt will search through the Environment Variable `PATH` for executables, DLLs, and Windows scripts such as `.bat` and `.cmd` files. If a Perl script program is saved with either the `.cmd` or `.bat` suffix in the folder whose path is listed in the `PATH`, we can invoke the script anywhere. To save a Perl script as the `.cmd` file, we have to rewrite the script as

```
1    @rem = '
2    @goto endofperl
3    ';
4
5    $myString = "He is a programmer.";
6    print "$myString\n";
7
8    # Replace "He" by "She"
9    $myString = reverse $myString;
10   chop $myString;
11   chop $myString;
12   $myString = reverse $myString;
13   $myString = "She$myString";
14   print "$myString\n";
15
16   # Make the string all lower case
17   $myString =~ tr/A-Z/a-z/;
18   print "$myString\n";
19
20   # Make the string all upper case
21   $myString =~ tr/a-z/A-Z/;
22   print "$myString\n";
```

```
23
24    __END__
25    :endofperl
26    @perl -S %0.cmd %1 %2 %3 %4 %5
```

Referring to the last line of code, %0, %1, etc. are the potential command-line arguments with %0 being the script name. Therefore, the last statement invokes the same script, denoted by `%0.cmd`, at the same directory to avoid typing the full path. The @ character placed in front of a statement is to suppress the echoing or printing out the execution of this statement at the command line. We can now invoke this script from the `test` folder conveniently as illustrated below:

```
D:\ArtProgram\test>example_1
He is a programmer.
She is a programmer.
she is a programmer.
SHE IS A PROGRAMMER.
```

As seen above, it is very easy to write a Perl script to manipulate a text string. On the other hand, it would require considerable effort to achieve the same task via C/C++ programming.

Out second example is writing the Perl script `AddCopyright.cmd` that opens a named `.cpp` file to add copyright information to the description. Opening, for example, `apiHornerEval.cpp`, we shall see the following description

```
/*-------------------------------------------------------------
API Name
   apiHornerEval

Description
   It evaluates polynomial and optionally its first derivative.

Signature
   void apiHornerEval(int ndeg, double *pA,
            double x, double &f, double *q)

    INPUT:
      ndeg     degree of polynomial
      pA       coefficients of polynomial
      x        variable at which polynomial is evaluated
    OUTPUT:
      f        polynomial value
      q        (optional) first derivative of polynomial

History

   Y.M. Li     10/15/2012 : Creation date
-------------------------------------------------------------*/
```

Suppose that we want to add the copyright information before `API Name` and keep the rest of the code to be the same. Then, the Perl script would be

```
1   @rem = '
2   @goto endofperl
3   ';
4
5   if ($ARGV[0] eq "")
6   {
7      die print "\nSyntax: AddCopyright <filename.cpp>\n\n";
8   }
9
10  open(INFILE, "$ARGV[0]") || die "Cannot open $ARGV[0].\n";
11
12  $filename = $ARGV[0];
13  chop $filename;
14  chop $filename;
15  chop $filename;
16  chop $filename;
17  $filename = "$filename.txt";
18
19  open(OUTFILE, ">$filename") || die "Cannot open $filename.\n";
20
21  foreach $Line (<INFILE>)
22  {
23    if ($Line =~ /API Name/)
24    {
25       print OUTFILE "Copyright (C) 2003. All rights reserved.\n\n"
26    }
27    print OUTFILE "$Line";
28  }
29
30  close(OUTFILE);
31  close(INFILE);
32  unlink("$ARGV[0]");
33  rename("$filename", "$ARGV[0]");
34
35  __END__
36  :endofperl
37  @perl -S %0.cmd %1 %2 %3 %4 %5
```

Line 5 to line 8 validate the input to make sure the command-line argument is not empty. The statement at line 10 opens the named `.cpp` file as an input file. From line 12 to 17, we replace the suffix `.cpp` by `.txt`. The statement at line 19 opens the named output file whose suffix is `.txt`. From line 21 to 28, we loop through each line of the input file and print the same line to the output file. But if this line contains the string `API Name`, we need to print out the copyright information before printing this line. Statements at line 30 and 31 close the output and input file. The statement at

line 32 deletes the original `.cpp` file, and the statement at line 33 rename the `.txt` file to `.cpp` file.

Copy `apiHornerEval.cpp` to `test` folder and type the following at the command line:

```
D:\ArtProgram\test>AddCopyright apiHornerEval.cpp
```

Opening the newly-updated `apiHornerEval.cpp`, we should see the copyright information has been added to the description. Everything else in this file is still the same as shown below.

```
/*-----------------------------------------------------------------
Copyright (C) 2003. All rights reserved.

API Name
   apiHornerEval

Description
   It evaluates polynomial and optionally its first derivative.

Signature
   void apiHornerEval(int ndeg, double *pA,
            double x, double &f, double *q)

    INPUT:
      ndeg     degree of polynomial
      pA       coefficients of polynomial
      x        variable at which polynomial is evaluated
    OUTPUT:
      f        polynomial value
      q        (optional) first derivative of polynomial

History

   Y.M. Li     10/15/2012 : Creation date
-----------------------------------------------------------------*/
```

Our last example is writing the Perl script `AddCopyrightLib.cmd` that adds copyright information to all the `.cpp` files residing in the current directory. Through this example, we will learn how to loop through every file in a directory and how to call a subroutine. Since this script is similar to `AddCopyright.cmd`, we make a copy of this script and name it `AddCopyrightLib.cmd`. Open this script file by any text editor and modify it as follows:

```
1    @rem = '
2    @goto endofperl
3    ';
4
5    opendir(DIR, ".");
```

```
6     @files = readdir(DIR);
7     foreach $filename (@files)
8     {
9        if ($filename =~ /\.cpp/)
10       {
11          addCopyright(@_=($filename));
12       }
13    }
14    close(DIR);
15
16
17    # Subroutine adds copyright information to each named .cpp file.
18
19    sub addCopyright
20    {
21      local($filename) = @_;
22
23      open(INFILE, "$filename") || die "Cannot open $filename.\n";
24
25      $filename_new = $filename;
26      chop $filename_new;
27      chop $filename_new;
28      chop $filename_new;
29      chop $filename_new;
30      $filename_new = "$filename_new.txt";
31
32      open(OUTFILE, ">$filename_new") || die "Cannot open $filename.\n";
33
34      foreach $Line (<INFILE>)
35      {
36        if ($Line =~ /API Name/)
37        {
38           print OUTFILE "Copyright (C) 2003. All rights reserved.\n\n"
39        }
40        print OUTFILE "$Line";
41      }
42
43      close(OUTFILE);
44      close(INFILE);
45      unlink("filename");
46      rename("$filename_new", "$filename");
47    }
48
49    __END__
50    :endofperl
51    @perl -S %0.cmd %1 %2 %3 %4 %5
```

Line 5 opens the current directory, and line 6 reads file names in the opened directory

and stores them in the `@files` array. From line 7 to 13, we loop through each filename and, when the filename has the suffix `.cpp`, call the subroutine `addCopyright` to add the copyright information. Note how the filename is passed to the subroutine. The implementation of subroutine is basically the same as `AddCopyright.cmd`. To test this script, copy some source files to the `test` folder and type

```
D:\ArtProgram\test>AddCopyrightLib
```

Upon the completion of execution, we should see that copyright information has been added to every source file in this folder.

These examples were designed to familiarize readers with basic Perl scripting. By walking through these examples, it is adequate for readers to understand the automation scripts used in this book. There are many excellent books and online tutorials on Perl programming, readers can refer to these resources for more information.