4.8 Subroutine, packages and Modules

4.8.1 Subroutines

The following example demonstrates a subroutine declaration and calling the subroutine.

```
#! /usr/local/bin/perl

# Declare the subroutine named usage
sub usage
{
    my ($program, $exitCode) = @_;
    print "Usage: $program [-v] [-h]\n";
    exit ($exitCode) ;
}
# Call usage.
usage ($0,   1) ;
```

4.8.2 Packages

The following example demonstrates the creation of the package named Nothing and usage of the package in a script.

```
package nothing;
sub doNothing
{
     print "This package does nothing!\n";
 }
1 ;
```

Usage of the package:

```
#!/usr/local/bin/perl
# Use the package nothing.
require "nothing.pl" ;
# Call the function 'doNothing' in the 'nothing' package. nothing: :doNothing( );
```

4.8.3 Modules

The following example is a module that reads the password file and stores the account information in an object.

```
package AcctInfo;
# Set up internal variables.
sub new
{
      my $self = { };
      my ($loginId, $passwd, $uid, $gid, $quota, $comment, $gcos, $home, $shell);
      my $login = getlogin( ) ;
       # Get information from the passwd file.
       ($loginId, $passwd, $uid, $gid, $quota, $comment, $gcos, $home, $shell), =
       getpwnam($login);
       # Store information in the object.
      $self-> {'login'}  =  $login;
      $self-> {'uid'}  =  $uid;
      $self-> {‘gid’}  =  $gid;
      $self-> {'home'}  =  $home;
      $self-> {'shell'}  =  $shell;

      # Bless this object . . .
      return bless $self;
 }

# Return the user's login id.
sub getloginid
{
      my  $self = shift;
      return  $self->{'login'};
}

# Return the user's uid
sub getuid
{
      my $self = shift;
      return $self->{'uid'};
}

# Return the user's gid
sub getgid
{
     my $self = shift;
      return $self->{'gid'};
```

```
# Return the user's home
sub gethome
{
    my $self = shift;
    return $self-> { 'home' } ;
}

# Return the user's shell
sub getshell
{
    my $self = shift;
    return $self->{'shell'};
}
1;
```

The following example uses the module defined above and calls one of the methods defined.

```
#!/usr/local/bin/perl

# Use the account information module .
use  AcctInfo ;
# Call the new method.
my  $object = ne w AcctInfo( );
# Get the uid .
my  $uid = $object ->getuid( ) ;
# Print out the results.
print  "UID: $uid \n";
```

4.9 Variable Localization

To demonstrate how to use MY, the following function creates global variable named $xxx and prints out the value.

```
#! /usr/local/bin/perl
# Define a basic subroutine.
sub myFunction
{
      # Define $xxx locally within this function.
      my $xxx = 5;
      # Print out. the local value of $xxx
      print  "Inside the function  \$xxx  =  $xxx  \n";
}
# Set the variable $xxx
$xxx = 1;
```

```
# Print out the global value of the variable
print "Before function \$xxx =  $xxx \n" ;
# Call the function .
myFunction( ) ;
# Print out the global value of the variable
print "After function \$xxx = $xxx \n" ;
```

4.10 File manipulation

4.10.1 Check if a file exists

The following example checks whether the named file exists.

```
#! /usr/loca1/bin/perl  -w
# Purpose
# Determines if a file exists.
use Getopt : : Long ;
# Set up the command line to accept a filename.
my $ret  = GetOptions ("f | filename : s");
my $filename = $opt_f  | | die "Usage:  $0  -f   filename\n" ;
# Check if the file exists
if (-e $filename)
{
      print  "The file $filename exists. \n " ;
 }
else
{
      print "The file $filename does not exist. \n" ;
 }
```

4.10.2 Read from a file

The following example demonstrates how to read from a file

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

# Reads from a file.
use Getopt: :Long ;

# Set up the command line to accept a filename.
my $ret = GetOptions ( " f  | filename: s “ ) ;
```

```
my $filename = $opt_f  | | die "Usage: $0  -f  filename\n";
# Open the file .
open (INPUT, "$filename")   | |  die "Could not open file
$filename : $!\n";
# Start reading from the file.
while (<INPUT>)
{
      chop;
      print "Line $ .= <$->\n" ;
 }
# Close the file
close (INPUT) ;
```

4.10.3 Write to a file

The following example demonstrates a file is opened for read and another is opened for write. The input file is read and the contents are written on to the output file.

```
#! /usr/local/bin/perl  -w
# Purpose
#   Writes to a file .
use Getopt: :Long;

# Set up 'the command line to accept a filename .
my $ret = GetOptions ( "i | input : s", "o | output : s");
my $input = $opt-i  | | die "Usage: $0  -i  Input filename -o Output filename \n";
my $output = $opt_o | | die "Usage: $0 -i  Input filename -o Output filename\n";
# Open the input file.
open (INPUT, "$input")  | | die "Could not open file $input  : $ ! \n" ;

# open the output file.

open  (OUTPUT,  ">$output")  | |  die "Could not open file $output  :  $ !\n " ;
# Start reading from the input file.
while (<INPUT>)
{
      chop;
      # Write to the output filename.
      print OUTPUT "Line $ . =  <$_ >\n" ;
 }
# Close the files.
close (INPUT) ;
close (OUTPUT) ;
```