4.5 Regular expressions

Regular expressions are used to search for patterns.

4.5.1 Modifiers

The following example demonstrates how to use a modifier on a regular expression.

```
#!/usr/local/bin/perl  -w
# Create a basic string.
my $string = "XXX";
# Perform a basic case sensitive pattern match.
if  (  $string =~  /XXX/ )
{
     print "Case Match!\n";
 }
# Perform a case insensitive pattern match.
if ( $string =~  /XXX/i )
 {
     print "Case insensitive Match!\n";
 }
```

4.5.2 Pattern quantifiers

The following example scans an array of numbers looking for a number that starts with 870 and is followed by 3 2s in a row.

```
#!/usr/local/bin/perl  -w
# Create a list.
my @numbers = qw  (870226980  870222000  870222555  871333111  870222333);
# Cycle through the list.
for (@numbers)
{
     # Look for 870222
     if (/^870(2{3})/)
     {
          print  $_. "\n";
      }
 }
```

4.5.3 Character patterns

The following example gets the current date in the Unix date format and looks for the hours, minutes and seconds.

```
#!/usr/local/bin/perl  -w
# Get the date in the standard date format. (ex: Tue Oct 24 19:03:03 1995 )
my $date = localtime( );
   #Search through the date looking for the hour minute and second.
if ($date =~ / (\d\d) : (\d\d) : (\d\d) /)
{
      # Save the information.
      my $hours = $1;
      my $minutes = $2;
      my $seconds = $3;
      # Print out the split date.
      print "Hours: $hours \n";
      print "Minutes: $minutes \n";
      print "Seconds: $seconds \n";
}
```

4.6 Operators: Numeric and String

4.6.1 String operators

The following example demonstrates the correct use of operators eq, lt, and gt .

```
#!/usr/local/bin/perl  -w
# Do this forever
for ( ; ; )
 {
      # Get the information from the user.
      print "Enter a word: " ;
      my $word1 =  <STDIN>; chomp $word1;
      print "Enter another word: " ;
      my $word2 =  <STDIN>; chomp $word2;
      # Perform some basic operations.
      if ($word1 eq $word2)
       {
          print  “The two phrases are equivalent.\n”;
       }
      if ($word1  lt  $word2)
       {
          print “<$word1>  is  alphabetically less than <$word2>\n" ;
```

```
    }
  if ($world  gt  $word2)
    {
      print "<$wordl> is alphabetically greater than <$word2>\n" ;
    }
  }
```

4.6.2 Numeric operators

The following example demonstrates the use of basic numeric operators:

```
#!/usr/local/bin/perl -w
# Do this forever
for ( ; ; )
 {
      # Get the information from the user.
      print "Enter a number: " ;
      my $numl = <STDIN>;  chomp $numl;
      print "Enter another number: " ;
      my $num2 = <STDIN>; chomp $num2;
      # Perform some basic operations.
      my $sum = $numl  +  $num2;
      my $diff = $numl  -  $num2;
      print "The sum of  $numl and  $num2 is  $sum\n" ;
      print "The difference of $numl and $num2 is $diff\n";
      if ($numl  ==  $num2)
      {
           print  "Both numbers are equal. \n " ;
      }
      if ($numl  <  $num2)
      {
           print  "<$numl> is numerically less than <$num2>\n";
      }
      if ($numl  >  $num2)
      {
        print  "<$num1>  is numerically greater than <$num2>\n";
       }
}
```

4.7 Control statements

4.7.1 Conditional control statements

The following example demonstrates the use of if statement, using both the *elsif* and *else* statements.

```
#!/usr/local/bin/perl  -w
while (<STDIN>)
{
    chomp;
     if ($_  <  10)
     {
          print "$ is less than 10.\n";
     }
     elsif ($_  <  20)
     {
           print "$_ is between the value of 10 and 19. \n" ;
     }
     else
     {
           print "$ is greater than or equal to 20.\n";
      }
 }
```

4.7.2 Loop control statements

An example for the for loop is shown in the following script :

```
 #!/usr/local/bin/perl -w
for ($x=l;  $x  <=  10;  $x++)
{
     print "$x\n";
}
```

The **foreach** statement is very much like the *for* loop except it iterates through list values. The following is an example of a **foreach loop .**

```
#! /usr/local/bin/perl  -w
# Create a list of the days of the week.
my @days = qw (Monday Tuesday Wednesday Thursday Friday Saturday Sunday);
my $day;
# Cycle through the loop, and print out the contents.
Foreach  $day  (@days)
{
  print  "$day\n" ;
}
```

4.7.3 Labels

Labels are used when you want to jump to a specific location within the code. The following example uses the *last* keyword to break out of nested for loops.

```
#! /usr/local/bin/perl  -w
# Count from 1 to 5
EXIT:
{
     for ( ; ; )
     {
          for ($x=l; $x<=20; $x++)
          {
               # Leave the for loop early.
               last EXIT if {$x >  5);
               print “$x\n”;
          }
     }
}
print “All done . . .\n” ;
```