ASCII File Manipulation with Perl

- The previous lecture introduced you to the basics of Perl
- This lecture will concentrate on ASCII file manipulation with Perl
- Many engineering applications produce large ASCII files from which data must be extracted
  - Not always possible or desireable to place data in a spreadsheet for extraction

Parsing a 'passwd' file

Shown below is an */etc/passwd* file

```
root:x:0:13:admin account:/tmp:/usr/bin/ksh
daemon:x:1:1:daemons:/:/dev/null
Guest:x:499:511:guest account:/home/Guest:/usr/bin/ksh
profile:x:503:13::/home/profile:/usr/bin/ksh
reese:x:508:13::/home/reese:/usr/bin/ksh
```

Each field is separated by colons (':')

This type of file is very easy to parse via the 'split' function

Perl script for parsing /etc/passwd

```
#!/usr/bin/perl -w
$fname = "passwd.txt";
open(INPUT,$fname);
while (<INPUT>) {
    chop;
    @words = split(':',$_);
    print "name:$words[0],  shell: $words[6] \n";
}
close(INPUT);
```

Chop off end of line

Split on ':'

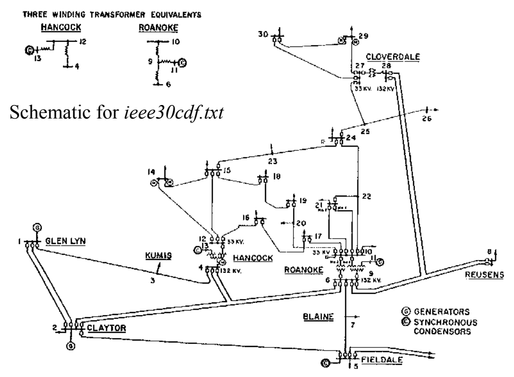
The '$_' is a special variable – typically the results of an operation is placed automatically into this variable. The results of the 'chop' operation is placed into '$_', which is then used by the 'split' function.

Parsing an IEEE Load Flow Data File

A tougher file to parse is an IEEE Load Flow data file. This is an ASCII data file that contains solved load flow data for a power distribution network. These files can be thousands of lines.

The following files can be found in the directory 'power_app' :

'cdf_format.txt'	Text file that defines the format
'ieee30cdf.txt'	A CDF file with 30 busses
'ieee300cdf.txt'	A CDF file with 300 busses
'30bus600.bmp'	A schematic of the 30 bus system



Schematic for *ieee30cdf.txt*

Load Flow file format

The low flow file format has fixed-width field and is intended to be parsed by FORTRAN programs.

The first few lines of ieee30cdf.txt appear below (line is clipped because of length):

```
08/20/93 UW ARCHIVE           100.0  1961 W IEEE 30 Bus Test Case
BUS DATA FOLLOWS                            30 ITEMS
   1 Glen Lyn 132  1  1  3 1.060    0.0      0.0      0.0   …
   2 Claytor  132  1  1  2 1.043  -5.48     21.7     12.7   …
   3 Kumis    132  1  1  0 1.021  -7.96      2.4      1.2   …
   4 Hancock  132  1  1  0 1.012  -9.62      7.6      1.6   …
```

Bus number (col 1-4), integer

Bus name (col 6-17), ASCII

Various numeric fields, integer and float

Load Flow file format (cont)

The file is split into various sections, the first two are called ‘BUS DATA’ and ‘BRANCH DATA’.

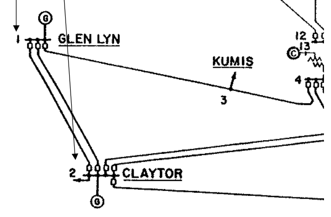
The ‘BUS DATA’ contains a line for each bus in the system and gives various details about that bus such as Load MW, Base KV, etc.

The ‘BRANCH DATA’ section details the connectivity of the system:

```
 30 Bus 30    33  1  1  0 0.992 -17.94     10.6      1.9 …
-999
BRANCH DATA FOLLOWS                              41 ITEMS
   1    2  1  1 1 0  0.0192    0.0575      0.0528     0   …
   1    3  1  1 1 0  0.0452    0.1652      0.0408     0    …
   2    4  1  1 1 0  0.0570    0.1737      0.0368     0     …
   3    4  1  1 1 0  0.0132    0.0379      0.0084     0     …
```

End of BUS DATA

```
 30 Bus 30    33  1  1  0 0.992 -17.94     10.6      1.9 …
-999
BRANCH DATA FOLLOWS                              41 ITEMS
   1    2  1  1 1 0  0.0192    0.0575      0.0528     0   …
   1    3  1  1 1 0  0.0452    0.1652      0.0408     0    …
   2    4  1  1 1 0  0.0570    0.1737      0.0368     0     …
   3    4  1  1 1 0  0.0132    0.0379      0.0084     0     …
```



Bus 1 & 2 connect.

Parsing a Load Flow File

To parse a load flow file, will need to find start of BUS DATA and BRANCH DATA sections.

```
#!/usr/bin/perl –w

$fname = "./power_app/ieee30cdf.txt";
open(INPUT,$fname);

# first, find the bus data
while (<INPUT>) {
    if ($_ =~/^BUS(.*)/) {
       last;
    }
}
```

Pattern matching – look for a line that starts with “BUS”.

Pattern Matching

```
while (<INPUT>) {
    if ($_ =~/^BUS(.*)/) {
      last;
    }
}
```

Last line read is in “$_ “ variable

‘=~’ is pattern match operator.

Search string is bracketed by ‘/’ .

The ‘^’ special character indicates match should be at start of string.

The ‘(.*)’ is wildcard that says match any number of characters after this.

‘last’ causes a loop exit

Another Way

```
while (<INPUT>) {
    chop;
    @words = split;
    if ($word[0] eq "BUS") {
       last;
    }
}
```

Split each line into words, look for a line whose first word is equal to ‘BUS’. ‘split’ with no arguments splits the string stored in $_ on whitespace.

Note the use of the ‘eq’ operator for string comparison.

Parsing Bus Data

Would like to extract fields from each bus line. We cannot just use the ‘split’ function because the name field may contain spaces in it, which would cause our word counts to be different for each line.

However, we know the starting, ending columns for each field.

The ‘substr’ function can be used to extract a substring from a string given a starting offset in the string, and a length:

$new_string = substr($target_string, $offset, $length)

If $length is not specified, then extract all characters from offset until the end of the string.

Parsing Bus Data (cont)

```
# bus data
while (<INPUT>) {
    chop;
    $this_line = $_;
    if ($this_line =~/^-999(.*)/) {
       last;
    }
    $bus_number = substr($this_line,0,4);
    $bus_name = substr($this_line,4,14);
    # the rest of the line is numbers, so can split it
    $line_rest = substr($this_line,17);
    @words = split(' ',$line_rest);
    $kv = $words[9];
    print "$bus_number $bus_name Base_KV = $kv \n";
}
```

Exit if at end of bus data

Get bus#

Get bus name

Get rest of line

Split rest of line and get BaseKV value for bus

Find BRANCH DATA

```
# find branch data

while (<INPUT>) {
    if ($_ =~/^BRANCH(.*)/) {
     last;
    }
}

# branch data

while (<INPUT>) {
### do something with branch data
###
}
```

Summary

- One of the most common Perl applications is to parse ASCII data files
- Perl has powerful features for parsing ASCII files
  - *split* function
  - *substr* function
  - pattern matching