Introduction to *Perl*

CIS*2750

Advanced Programming Techniques

What is *Perl*?

- *Perl* was developed by Larry Wall.
  - started out as a scripting language to supplement *rn*, the USENET reader.
  - available on virtually every computer platform
- *Perl* is an interpreted language that is optimized for string manipulation, I/O, and system tasks
  - has builtins for most of the functions in **section 2** of the UNIX manuals -- very popular with sys administrators
  - incorporates syntax elements from the Bourne shell, csh, awk, sed, grep, and C ☹
  - provides a quick and effective way to write interactive web applications

BSD/Linux Manual Sections

1. **Commands available to users** (ls, more, perl)
2. **Unix and C system calls** (mkdir, time, exec)
3. **C library functions** (fopen, malloc)
4. Special files (especially /dev)
5. File formats and conventions
6. Games, demos
7. Word processing packages, misc.
8. System administration commands and daemons

"man printf" vs. "man 3 printf"

Basic Syntax

- *Perl* is free form.
- All *Perl* statements end in a semicolon, like C.
- **Comments**
  - begin with #
  - everything after the #, and up to the end of the line is ignored
  - the # needn't be at the beginning of the line

```
#!/usr/bin/perl
#
# ereader - a simple Perl program to re-format email
```

Variables

- *Perl* has several kinds of variables and data structures.
  - **Strongly typed** languages (C, C++, Java)
    - Explicitly **declare** variables before you use them.
  - **Dynamically typed** languages (Lisp, Python)
    - If a variable holds a number, the **programmer** is responsible not to pull substrings out of it.
  - *Perl* falls in the middle:
    - Which data type you use is implicit in *how you access* it, but you don't need to declare it before you use it.

Perl Functions

- *Perl* has many builtin functions identified by their unique names (print, chop, close, etc).
  - **man perlfunc** and **man perldoc**
- Arguments supplied as comma separated list in parentheses.
  - The commas are necessary, the parentheses *often* not! ☹

```
print("length: ", length("hello world"));
print "length: ", length "hello world";
```

Perl Functions: Example

```
$date = `date`;
chop($date);
```

- The first line executes the UNIX command **date** and captures its **stdout** output in the variable *$date*.
- Since the date has a trailing **newline**, we want to **chop** that off.

Scalars

- **Scalar Definition**

  A *scalar* is a single value, either numeric or a character string.

  Compare *composite*, having multiple values.

- Scalars are accessed by prefixing an identifier with **$**.
- **Identifier Definition**

  An *identifier* is a variable name.

  – Composed of upper or lower case letters, numbers, and the underscore **_**.

  – Identifiers are case sensitive (like all of *Perl*).

- Scalars are assigned by using **=**

```
$scalar = expression;
```

Scalar Example

```
$progname = "mailform";
```

- This is read as *the scalar progname is assigned the string mailform*.
- The **$** determines that *progname* is a **scalar**.
- The **=** makes this an **assignment**.
- The double quotes (**"**) define the string.
- All statements end with a semi-colon **;**.

Strings

- There are several ways of quoting strings in *Perl*, corresponding to the three quote characters on the keyboard.
- **" (double quote)** *interpolates* (substitutes, expands) variables between the pair of quotes.

```
$instr = "saxophone";
$little = "soprano $instr";
# the value of $little is “soprano saxophone”
```

' (apostrophe)

- The simplest quote, text placed between a pair of apostrophes is interpreted literally → **no variable interpolation** takes place.

```
$instr = 'saxophone';
$little = 'soprano $instr';
# the value of $little includes the text “$instr”
```

  - To include an apostrophe in the string, you must escape it with a backslash: `'sax\'s'`

` (backtick)

- This quote performs as it does in the UNIX shells
  - the text inside the *backticks* is executed as a separate process, and the **standard output** of the command is returned as the value of the string.
  - *Backticks* perform variable interpolation, and to include a *backtick* in the string, you must escape it with a backslash.

```
$memberList = "/usr/people/conductor/roster";
$memberCount = `wc -l $memberList`;
# $memberCount is the no. of members in the roster file,
#   assuming that each member is listed on a separate line.
```

Example

```
# mail program
$sendmail = "/usr/bin/mail";
# base of your httpd installation
$basedir = '/www';
# log file
$progname = "apache";
$logfile = "$basedir/etc/logs/$progname.log";
# mail the logfile
`$sendmail –s “Apache log” sysadmin < $logfile`;
```

/www/etc/logs/apache.log