LECTURE 11: FILE OPERATIONS

QUIZ 1

- Results up
- Objections:
  - What are the outputs for the following pieces of code?: Me
  - Complete the following program segment: Ying
  - Identify and fix errors: David

LAST LECTURE

- Subroutines
  - How to define subroutines
  - Subroutines with multiple params
- References
  - Nested arrays

FILES

- Access to files is similar to shell redirection
- Standard files
- Reading from files
- Writing to files
- File checks

FILE ACCESS

- open allows access to the file
- Redirect characters (<,>, etc) define access type
- Can read, write, append etc
- File handle refers to opened file
- close stops access to the opened file
- $! contains IO error messages

1.
```
open INPUT,"<datafile” or die "Can't open input file: $!";
open OUTPUT,">outfile” or die "Can't open output file: $!";
open LOG,">>logfile” or die "Can't open log file: $!";
open RWFILE,"+<myfile” or die "Can't open file: $!";
```
2.
```
close INPUT;
```

FILE ACCESS MODES

- < Read
- > Write
- >> Append
- +< Read and Write
- +> Read and Write(Create if does not exist)
- +>> Read and Write(Create if it does not exist, append if exists)

STANDARD FILES

- Standard files are opened automatically
  - STDIN: standard input
  - STDOUT: standard output
  - STDERR: standard error
  - print uses STDOUT by default
  - die uses STDERR by default

```
print STDOUT "Hello, world.\n"; # STDOUT not needed
open STDERR, ">>logfile” or die "Can't redirect errors to log: $!";
print STDERR "Oh no, here's an error message.\n";
warn "Oh no, here's another error message.\n";
close STDERR;
```

READING FROM FILES

- Input operator <> reads one line from the file, including the newline character
  - chomp removes newline if you want

```
print "What type of pet do you have? ";
my $pet =<STDIN>; # Read a line from STDIN
chomp $pet;# Remove newline
print "Enter your pet's name: ";
my $name =<>; # STDIN is optional
chomp $name;
print "Your pet $pet is named $name.\n";
```

READING FROM FILES

- Loop over entire file, assign each line to $_ by default
- Be sure that the file is open for reading first
- See read-from-stdin-file-pl

```
open CUSTOMERS, "<mailing_list" or die "Can't open input file: $!";

while (my $line =<CUSTOMERS>){
   my @fields = split(":", $line); # Fields separated by colons
   print "$fields[1] $fields[0]\n"; # Display selected fields
   print "$fields[3], $fields[4]\n";
   print "$fields[5], $fields[6] $fields[7]\n";
}
print while<>;# cat
print STDOUT $_ while($_=<STDIN>); #same, but more verbose
```

WRITING TO FILES

- print writes to a file
- print writes to STDOUT
- Be sure that the file is open for writing first

```
open CUSTOMERS, "<mailing_list" or die "Can't open input file: $!";
open LABELS, ">labels" or die "Can't open output file: $!";
while (my $line =<CUSTOMERS>) {
    my @fields = split(":", $line);
    print LABELS "$fields[1] $fields[0]\n";
    print LABELS "$fields[3], $fields[4]\n";
    print LABELS "$fields[5], $fields[6] $fields[7]\n";
}
```

FILE CHECKS

- File tests operators check if a exists, readable, etc.
- -e tests if file exists -r tests if readable
- -w tests if writeable -x tests if executable
- -l tests if symlink -T tests if a text file

```
my $filename = "pie_recipe";
if (-r $filename){
   open INPUT,“<$filename" or die "Can't open $filename: $!";
} else {
   print "The file $filename is not readable.\n";
}
```

READING FROM USER INPUT

```
print "Type your name and press return: ";
$name =<STDIN>;
chomp $name;
$size = length $name;
print "\nHello, $name!\n";
print "Your name contains $size letters.\n";
```

READING FROM A FILE(SIMPLE WAY)

```
while (<>) {
   print "$.: $_";
}
print "\n$. lines in total\n\n";
```

- <>: diamond operator. Reads a single line from the specified file
- $: refers to the current line number in the file
- $_: the default input. It's the line currently being processed
- To use the program, type **perl filename inputfile**
- Perl inspects command line argument, if a valid file name, Perl opens it implicitly for reading. Can even read >1 files

COMMAND LINE ARGUMENTS AND FILE HANDLES

```
#command-line-pl N FILE, adds a number to each line of a given file
$num = $ARGV[0];
$file = $ARGV[1];
open FILE, “<“, $file; # Can also use open(FILE, $file);

while ($line =<FILE>) {
   print $num + $line."\n"
}
```

- **open FILE, “<“, $file;** opens the file identified by $file, gives it the file handle FILE for later use
- It's conventional to user upper cases for filehandles
- You can also use **open FILE, ‘<‘, $file; open(FILE, $file); open FILE, $file;** when doing simple reading

COMMAND LINE ARGUMENTS AND FILE HANDLES (THE SIMPLE WAY)

```
#command-line-pl-2 N FILE, adds a number to each line of a
given file

$num = shift;
$file = shift;
open(FILE, $file);
while ($line =<FILE>) {
   print $num, " ", $line;
}
```

- shift is a list operator. It operates on ARGV implicitly when you first invoke a program

OTHER PERL MATERIALS(WE DO NOT COVER)

- Object oriented programming
- CPAN and modules
  - Comprehensive Perl Archive Network, cpan.org
  - Thousands of third party code from community
- For more information
  - man perl
  - perldoc

FIN!