

Topic 4


Control structures

Learning Perl 2nd edition
chapter 4, pages 58-65
chapter 6, pages 72-75
chapter 9, pages 105-106

Programming Perl 3rd edition
pages 111-127, 658-659, 80-83

Programming Perl 2nd edition
pages 95-105, 53-54, 131

perlsyn, perlvar manpages



CSE2395/CSE3395

1

Last time

Covered in Topic 3

- Lists
 - sequences of scalars
- Arrays
 - variables that contain lists
 - always start with a character
- List and array functions
 - sort, reverse
 - push, pop, shift, unshift
- Context
 - scalar context when scalar expected
 - list context when list expected

CSE2395/CSE3395

2

To be covered today

- Control structures
 - if, while/until, for, foreach, do-while
- True and false values
- Loop control
 - labels, next, last, redo
- Expression modifiers
 - another way of doing control structures
- Program arguments
 - @ARGV array
- Perl defaults
 - \$_ variable (standard argument)
 - <> filehandle (diamond operator)

CSE2395/CSE3395

3

Control structures

Definition

- Any programming construct that alters the flow of a program
 - selection
 - if/else, unless
 - iteration
 - while, until, for, foreach
 - last, next, redo

CSE2395/CSE3395

4

The if/else statement

Performing selection


```

unless (condition) is same as if (!condition)
if (condition) ← evaluated in scalar context
{
  # These lines executed if condition is true.
}
elsif (condition2) ← zero or more elsif clauses allowed
{
  # These lines executed if condition2 is true.
}
else ← else clause is optional
{
  # These lines executed if all
  # conditions are false.
}

```

braces are always required, unlike in C

Llama2 pages 59-61; *Camel3* page 114-115;
Camel2 page 98; perlsyn manpage



CSE2395/CSE3395

5

The while statement

Performing iteration


```

until (condition) is same as while (!condition)
while (condition)
{
  # This code executed until condition is false.
  # Block is never entered if condition was
  # always false.
}

```

braces are always required

Llama2 pages 61-62; *Camel3* pages 115-116
Camel2 page 98; perlsyn manpage



CSE2395/CSE3395

6

The for statement

Another way of performing iteration

evaluated in scalar context

```
for (initializer; condition; increment)
{
    # initializer code executed once before loop.

    # Block is executed while condition is true.

    # increment code always executed before end
    # of each iteration.
}
```



braces are always required



Llama2 page 63; *Camel3* pages 116-118
Camel2 pages 98-99; *perl5yn* manpage

CSE2395/CSE3395

7

Boolean evaluation

True or false?

- Boolean conditions evaluated in scalar context as strings
- False values are
 - ▶ "" (the empty string)
 - ▶ undef (the undefined value)
 - converted to empty string when evaluated as string
 - ▶ "0" (number 0 should be false)
 - ▶ empty arrays
 - in scalar context, returns number of elements (0)
- True values are
 - ▶ everything else



Llama2 page 59; *Camel3* pages 29-30
Camel2 page 46; *perl5yn* manpage

CSE2395/CSE3395

8

Example

Printing a random line from input

```
# Read in lines until EOF.
while (defined($input = <STDIN>))
{
    chomp $input;
    push @lines, $input;
}

# Get a random number of the right size.
srand;
$pick = int(rand(scalar @lines));

# Print line indexed by $pick.
print $lines[$pick], "\n";
```

CSE2395/CSE3395

9

The foreach statement

Iterating over a list

```
foreach $var (List)
{
    # $var equals each element of list in turn.
    # List may be a literal or named array.
}
```

if omitted, default iterator is \$_
parentheses are always needed
braces are always needed



Llama2 pages 63-65; *Camel3* pages 118-120
Camel2 pages 100-101; *perl5yn* manpage

CSE2395/CSE3395

10

Example

Adding a list of numbers

```
# Read input until EOF (note list context).
@data = <STDIN>;

# Iterate over every element in @data.
# Use of @data could be eliminated with:
# foreach $number (<STDIN>)
foreach $number (@data)
{
    # This actually modifies elements of @data.
    chomp $number;

    $total += $number;
}

# Print result.
print "Total is $total\n";
```

CSE2395/CSE3395

11

The default argument

The \$_ variable

- With many functions taking an argument, naming this argument is optional
- Default value used in this case is special variable \$_
- print;
 - ▶ same as print \$_;
- while (<STDIN>)
 - ▶ same as while (defined(\$_ = <STDIN>))
 - ▶ special case, only applies to filehandle in while condition



Llama2 pages 64, 72-73; *Camel3* pages 658, 682
Camel2 page 131; *perl5yn* manpage

CSE2395/CSE3395

12

Example

cat: duplicating standard input

```
# Explicit version using $line
while (defined($line = <STDIN>))
{
    print $line;
}
# Implicit version using $_
while (<STDIN>)
{
    print;
}
# Tiny version using expression modifier
print while <STDIN>;
```

CSE2395/CSE3395

13

Expression modifiers

A shorter way of doing control structures

- Works for if, unless, while, until, foreach


```
if (condition)
{
    statement;
}
```

Normal if statement

```
statement if condition;
```

same statement as modified expression

braces, parentheses not needed
only single simple expression allowed

 *Llama2* pages 105-106; *Camel3* pages 112-113
Camel2 page 96; *perlsyn* manpage

CSE2395/CSE3395

14

The do-while statement

Performing a loop at least once

unlike with a normal `while` expression modifier, which tests the condition first, the `do-while` statement always performs the loop body once before testing the condition

```
do {
    # This code is always executed at least once.
} while condition;
```

until also allowed



Llama2 page 62; *Camel3* page 112, 701; *Camel2* page 158
perlsyn, *perlfunc* manpages

CSE2395/CSE3395

15

Command-line arguments

The @ARGV array

- `@ARGV` contains command-line parameters given to script
 - empty array if no parameters
 - unlike C, no need for `argc` parameter because `@ARGV`'s size is known
- Script name is in special variable `$0`
 - different from C where `argv[0]` is program name and first parameter is in `argv[1]`



Llama2 page 73; *Camel3* page 659
Camel2 pages 138, 136; *perlvar* manpage

CSE2395/CSE3395

16

Example

Using @ARGV to print all command-line parameters

params.pl

```
#!/usr/bin/perl -w
print "Program name is: $0\n";
print "Parameters are:\n";
for ($i = 0; $i <= $#ARGV; $i++)
{
    print " $i: $ARGV[$i]\n";
}
```

```
% params.pl blorb rezrov cleesh
Program name is: params.pl
Parameters are:
0: blorb
1: rezrov
2: cleesh
```

CSE2395/CSE3395

17

<> (diamond) operator

Input from the default filehandle

- Often useful to have a program read from files if named, standard input otherwise
- If no command-line arguments given (`@ARGV` is empty):
 - `<>` is same as `<STDIN>`
- If command-line arguments given (`@ARGV` contains elements):
 - `<>` reads each named file in turn (each element of `@ARGV`)



Llama2 page 73-74; *Camel3* pages 80-83
Camel2 pages 53-55; *perlsyn* manpage

CSE2395/CSE3395

18

Example

A complete `cat` program

- Reading from `<>` duplicates behaviour of many Unix commands

```
# cat program that duplicates
# from STDIN if no parameters, otherwise
# concatenates all named files to STDOUT

while (<>)
{
    print;
}

# Even tinier version using expression modifier

print while <>;
```

CSE2395/CSE3395

19

Loop control commands

Changing the way a loop is executed

- `last`
 - exits the innermost containing loop
 - like C's `break` statement
- `next`
 - jumps to end of innermost containing loop
 - executes increment code of `for` loop, then tests loop condition
 - continues from beginning of loop
 - like C's `continue` statement



Llama2 pages 101-104; *Camel3* pages 120-123
Camel2 pages 101-103; `perlsyn` manpage

CSE2395/CSE3395

20

Loop control commands

Changing the way a loop is executed

- `redo`
 - jumps to start of innermost containing loop
 - does not test loop condition
 - no equivalent in C without use of `goto`
- `last`, `next` and `redo` can take a label
 - label attached to an enclosing block
 - command now repeats/exits that loop instead of innermost containing loop
 - not possible in C without use of `goto`
 - label should be typed in all capitals

CSE2395/CSE3395

21

Example

Printing the header of an email message

```
# Mail messages come in two parts: the header
# and the body. The first blank line in the
# message separates the header from the body.

# Read each line of the email message.
LINE: while (<>)
{
    # If line is blank (contains only a newline)
    # then exit the while loop.
    # Note use of expression modifier (if)
    # and (in this case redundant) label LINE.
    last LINE if $_ eq "\n";

    # Print header line.
    print;
}
```

CSE2395/CSE3395

22

Covered today

- Control structures
 - `if`, `while`, `for`, `foreach`, `do-while`
- True and false values
- Loop control
 - labels, `next`, `last`, `redo`
- Expression modifiers
 - another way of doing control structures
- Program arguments
 - `@ARGV` array
- Perl defaults
 - `$` variable (standard argument)
 - `<>` filehandle (diamond operator)

CSE2395/CSE3395

23

Going further

More things related to today's topic

- Loops, blocks and `goto`
 - more scary things to do with control structures, including implementing C's `switch`
 - *Camel3* pages 123-127, 729-730; *Camel2* pages 103-106, 178
- Perl special variables
 - `$_`, `@ARGV`, `$>`, `%ENV`, `$&`, `@INC`: line noise with meaning
 - *Camel3* pages 653-675; *Camel2* pages 127-140

CSE2395/CSE3395

24

Next time

To be covered in Topic 5

- Hashes
 - ▶ associative arrays
 - ▶ arrays indexed by strings
- Hash variables and functions

Reading:

Learning Perl 2nd edition chapter 5, pages 66-71

Programming Perl 3rd edition pages 76-78

Programming Perl 2nd edition pages 50-51

`perldata` manpage

