Research Skills 1: Programming
Lesson 2

Erik Tjong Kim Sang
Herman Stehouwer

13 September 2007

Last week: Variables and number processing

- names of numeric variables start with a dollar sign ($yearOfBirth)
- several arithmetic operators are available: + – * / % **
- as well as several functions: abs() int() rand() sqrt()
- input can be read from the keyboard: <STDIN>

This week

- conditional structures: `if ... then ...`
- truth expressions: `... and ...`
- iterative structures: `while ... do ...`

Conditional structures: example

```
# determine whether number is odd or even
print "Enter number:  ";
$number = <STDIN>;
chomp($number);
if ($number%2 == 0) {
    print "$number is even\n";
} elsif ($number%2 == 1) {
    print "$number is odd\n";
} else {
    print "Something strange has happened!\n";
}
```

Conditional structures: what is new?

- new Perl instructions: `if elsif else`
- truth expressions: `$a == 0` is different from `$a = 0`
- command blocks: { `...  commands ...` }
- indentation: extra spaces before commands in block

Numeric test operators for truth expressions

- == is equal to
- != is not equal to
- < is less than
- <= is less than or equal to
- > is greater than
- >= is greater than or equal to

Combining truth expressions

There are three operators available for combining truth expressions:

- `and` (alternative `&&`)
- `or` (alternative ||)
- `not` (alternative `!`)

Note: Perl does not have standard variables `true` and `false`. You can use 1 for true and 0 for false.

Truth expressions: example

```
# logical operator demo
$true = 1;
$false = 0;
$JohnSings = $true;
$BillSings = $false;
if ($JohnSings and $BillSings) {
    print "Everybody is singing\n";
} elsif ($JohnSings or $BillSings) {
    print "Someone is singing\n";
} elsif ((not $JohnSings) and not $BillSings) {
    print "No one is singing\n";
}
```

Iterative structures: example

Repeat a block of command a number of times:

```
# print numbers 1-10 in three different ways
# method 1:  use "while"
$i = 1;
while ($i<=10) { print "$i\n"; $i++; }
# method 2:  use "for"
for ($i=1;$i<=10;$i++) { print "$i\n"; }
# method 3:  use "foreach"
foreach $i (1,2,3,4,5,6,7,8,9,10) { print "$i\n"; }
```

Abbreviations for frequent operations

- `$i++;` # add 1 to the value stored in $i
- `$i--;` # subtract 1 from the value stored in $i
- `$i+=2;` # add 2 to the value stored in $i
- `$i-=3;` # subtract 3 from the value stored in $i
- `$i*=4;` # multiply the value stored in $i with 4
- `$i/=5;` # divide the value stored in $i by 5

Escaping from a loop: example

```
# loop control demo
$j = 0;
LOOP: for ($i=1;$i<=10;$i++) {
    $j += $i;
    if ($j == 2) { redo LOOP; }
    if ($j == 6) { next LOOP; }
    if ($j == 9) { last LOOP; }
    print "$j\n";
    $j = 0;
}
```

Programming: step by step

Assignment:

> Read ten numbers and print the largest, the smallest and a count representing how many of them are dividable by three.

We start with making a design of the program, with the tasks that the program needs to perform.

Design 1

Read ten numbers and print the largest, the smallest and a count representing how many of them are dividable by three.

1. Read the ten numbers and store them
2. Find the largest and print it
3. Find the smallest and print it
4. Count how many of them are dividable by three and print the count

This design is fine for processing ten numbers but not for dealing with a billion numbers.

Design 2

1. Repeat the following ten times
   (a) Read a number
   (b) Find out if it is larger than the previous numbers
   (c) Find out if it is smaller than the previous numbers
   (d) Find out if it is dividable by three
2. Print the largest, smallest and the dividable by three count

This seems like it could work. Let's see what variables we need.

Design 2 with variables specified

1. Initialize variables
2. Repeat the following ten times
   (a) Read $number
   (b) If $number > $largest then $largest = $number
   (c) If $number < $smallest then $smallest = $number
   (d) If ($number % 3) == 0 then $count3++
3. Print $largest, $smallest and $count3

Design 2 as Perl program

```
undef($largest); undef($smallest); $count3 = 0;
for ($i=1;$i<=10;$i++) {
    print "Please enter number $i:  ";
    $number = <STDIN>;
    if (not(defined($largest)) or $number > $largest) {
       $largest = $number; }
    if (not(defined($smallest)) or $number < $smallest) {
       $smallest = $number; }
    if ($number%3 == 0) { $count3++; }
}
print "L: $largest; S: $smallest; D3:  $count3\n";
```

EXERCISES WEEK 1

Exercise results

mark	1	2	3	4	5	mark	1	2	3	4	5
10.0	♣	♣	♣	♣	♣	8.5	♣	♠	♠	♣	
10.0	♣	♣	♣	♣	♣	8.4	♣	♣	♠	♣	
9.5	♣	♣	♠	♣	♣	8.0	♣	♣	♣		
9.2	♣	♣	♣	♣	♠	7.7	♠	♣	♠	♠	♠
9.2	♣	♣	♣	♣	♠	7.7	♠	♣	♠	♠	♠
9.2	♣	♣	♠	♣	♠	6.4	♣	♣	♠		
9.0	♣	♣	♣	♣		6.1	♠	♠	♠	♣	
8.7	♣	♣	♠	♣		5.3	♠	♣	♠	♠	♠
8.7	♣	♣	♠	♣	♠	3.5	♠	♣	♠		

♣ = perfect; ♠ = one or more errors

Exercise discussion (1-4)

1. Few problems. Some people forgot to include the test results.

2. Everybody managed to do this exercise

3. Problems with testing and debugging: see programming tips

4. Most people that tried it found the three lines to change:

```
$dateNbr = 0; # December 31, 2007 is a Monday
print "Is the day after February 29?  ";
$dateNbr = $dateNbr+$answer*29;
```

Exercise discussion (5)

5. Three perfect solutions of which one looked like this:

```
print "Largest number:",($a+$b+abs($a-$b))/2,"\n";
```

It works because if you add two numbers to each other and the difference as well, you always get twice the largest number:

- 33+12+abs(33-12) = 33+12+21 = 66
- 12+33+abs(12-33) = 12+33+21 = 66

Programming tips

- Test your programs with option -w: `perl -w program.pl`
- Use extra print statements to check values of variables
- Use indentations for optional command blocks
- Always include test results of your programs in your report

START WITH EXERCISES AT
http://ifarm.nl/erikt/perl2007/