Research Skills 1: Programming
Lesson 3

Erik Tjong Kim Sang
Herman Stehouwer

20 September 2007

Week 1: 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>

Week 2: Control structures

- Conditional structures: `if (condition)` { `command` }
- Truth expressions: `and`, `or` and `not`
- Iterative structures: `while ()`, `for(;;)` and `foreach ()`

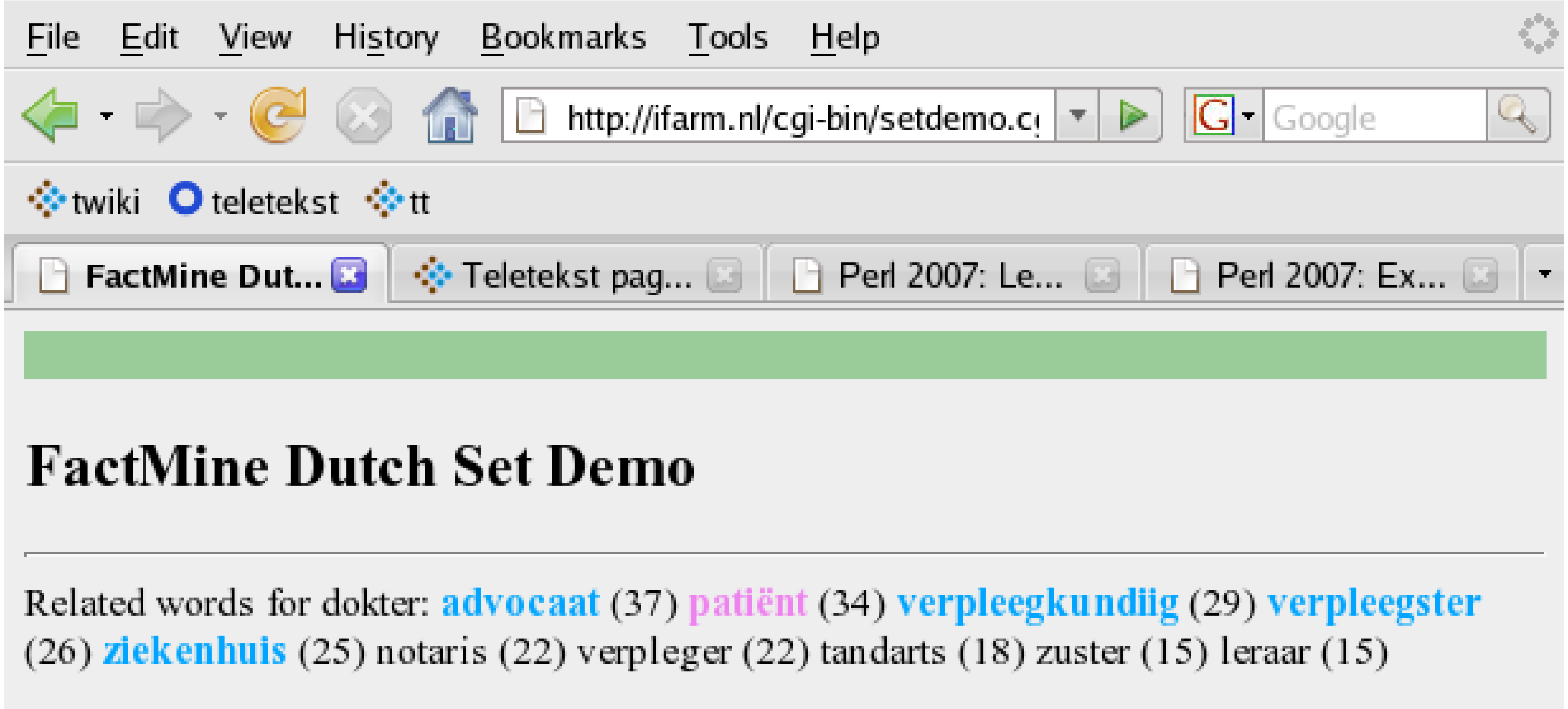
STRING PROCESSING

Exercise 3.5*

Write a program that reads the plural of a word (you are free to use either English or Dutch) and prints the singular form. The program should process plurals until the user presses Enter without entering a word. Example run:

```
Please enter a plural:  boys
The singular form is:   boy
...
```

Plural-to-singular conversion is not easy



FactMine Dutch Set Demo

Related words for dokter: advocaat (37) patiënt (34) verpleegkundiig (29) verpleegster (26) ziekenhuis (25) notaris (22) verpleger (22) tandarts (18) zuster (15) leraar (15)

About strings

- string variables look like numeric variables: `$string`
- string content is placed between quotes:
  ```
  $string1 = "test";
  $string2 = "$string1"; # $string2 contains "test"
  $string2 = ’$string1’; # $string2 contains "$string1"
  ```
- examples of string concatenation:
  ```
  $string1 = "$string1$string2";
  $string1 = $string1 . $string2;
  $string1 .= $string2;
  ```

String comparison operators

strings	**numbers**	**explanation**
`eq`	`==`	equal
`ne`	`!=`	not equal
`lt`	$<$	less than
`le`	$<=$	less than or equal to
`gt`	$>$	greater than
`ge`	$>=$	greater than or equal to

Note: numbers are ranked before capital characters which in turn are ranked before lower case characters. For other characters, see Wikipedia.

Computing the length of a string

- `length($string)` returns the number of characters in `$string`
- Examples:
  `length("abcde")` → 5
  `length(100)` → 3
- Note: spaces, tabs and newlines are counted as well!
  `length("abcde ")` → 6

Reading text from the keyboard

```
print "Please enter abcde and press Enter: ";
$line = <STDIN>;

# now $line contains "abcde\n"; let’s test this
print length($line),"\n"; # this will print 6

# let’s remove the newline
chomp($line);

# testing again
print length($line),"\n"; # this will print 5
```

String manipulation

We use two operations for changing the content of strings:

- **character translation**: `tr/abc/123/`
  replace every character from the left part (`abc`) by the corresponding character in the right part (`123`)
  `a` → `1` `b` → `2` `c` → `3`

- **string substitution**: `s/before/after/`
  replace the left part (`before`) by the right part (`after`)
  `before` → `after`

Character manipulation examples: tr///

- convert all capital characters to lower case: TEST12 → test12
  `$text =∼ tr/A-Z/a-z/;`

- delete all vowels: TEST12 → TST12
  `$text =∼ tr/AEIOUaeiou//d;`

- replace nonnumber sequences with x: TEST → x12
  `$text =∼ tr/0-9/x/cs;`

Common operators: d: delete; c: complement; s: squeeze

String manipulation examples: s///

- replace first occurrence of *bug*: bugs4bugS → features4bugS
```
$text =∼ s/bug/feature/;
```

- replace all occurrences of *bug*: bugs4bugS → features4featureS
```
$text =∼ s/bug/feature/g;
```

- replace all capital characters by CAPS: bugs4bugS → bugs4bugCAPS
```
$text =∼ s/[A-Z]/CAPS/g;
```

Common operators: g: global; i: ignore case

String test examples: //

- test if a string variable contains some string:

  ```
  if ($text =~ /danger/i) { command }
  ```

Common operators: g: global; i: ignore case

This works fine for checking if a text contains a word.

But how can we perform more complex checks?

For example, how do we check if a string contains exactly three a’s?

Regular expressions

A regular expression is a formula which represents a set of strings.

In order to make this possible, some character sequences have a special meaning in regular expressions.

For example: the Kleene star (`*`) in combination with a preceding character represents the set of strings consisting of repeated occurrences of the character:

`a*` represents { `"" , "a" , "aa" , "aaa" , ...` }

Special characters in regular expressions

\b	word boundaries	∧	beginning of string
\d	digits	$	end of string
\n	newline	.	any character
\r	carriage return	[bdkp]	characters b, d, k and p
\s	white space characters	[a-f]	characters a to f
\t	tab	[∧a-f]	all characters except a to f
\w	alphanumeric characters	(abc|def)	string abc or string def

Note: \B represents every character except word boundaries; \D every character excepts digits, and so on.

Specifying repetitions

*	zero or more times
+	one or more times
?	zero or one time
{p,q}	at least p times and at most q times
{p,}	at least p times
{p}	exactly p times

Use brackets for forcing these characters to operate on strings:

`ha*` → { `"h" , "ha" , "haa" , "haaa" , ...` }

`(ha)*` → { `"" , "ha" , "haha" , "hahaha" , ...` }

Referring back to previous matches

If a part of the last regular expression is enclosed between parentheses, it can be referred to with `$N` where `N` is a number indicating the position of the part in the expression.

Here is an example for date detection:

```
if ($text =~ /(\d?\d)-(\d?\d)-(\d\d\d\d)/) {
   $day = $1; # first part between brackets
   $month = $2; # second part between brackets
   $year = $3; # third part between brackets
}
```

PROGRAMMING EXAMPLE

Programming task

URIs (Uniform Resource Identifiers) are addresses of (online) resources like `http://www.uvt.nl/`

Write a program that can extract all URIs from a web page

How can we find URIs in a webpage?

Example web page code:

```
<div class="block">
<a href="http://www.uvt.nl/faculties/fsw/emotions2007/">
```

Answer: URIs appear in web pages after the phrase `href=` between a pair of double quotes (").

Code attempt one

```
# read lines of html
while (<STDIN>) {
   $line = $_; # store the current line in $line
   chomp($line);
   # is there a URI in this line?
   if ($line =~ /href="(.*)"/) {
     # yes: put URI in $uri and print the result
     $uri = $1;
     print "$uri\n";
   }
}
```

Testing code attempt one

```
perl -w test1.pl < uvt.html
```

- the program generates 42 URIs; 27 are correct
- problem 1: it generates 13 URIs with extra information
  `http://www.uvt.nl/uvtsite/">Site info...uvt.nl`
- problem 2: it generates 5 incomplete URIs:
  `#nieuws`
- problem 3: it can only identify one URI per line

A note about URIs

Here is an example of a complete URI:
http://www.host.com/directory/file.html#pointer

It consists of five parts: protocol (http), host name (www.host.com), directory, file name (file.html) and location pointer.

Web pages contain complete URIs as wel as partial URIs:
- only the pointer: #pointer
- the file name (with a pointer): file.html
- the directory name (with a file name): /directory

We would like the program to only output complete URIs.

Code attempt two (1)

```
# read the address of this web page from STDIN
$address = <STDIN>;
chomp($address);
# split the address in host, directory and file
if ($address =  /(.*:\/\/[∧\/]*)(.*)([∧\/]*)/) {
   $host = $1;
   $dir = $2;
   $file = $3;
} else {
   die "error: cannot parse URI $address\n";
}
```

Code attempt two (2)

```
# read lines of html
while (<STDIN>) {
   # store the current line in $line
   $line = $_;
   chomp($line);
   # are there URIs in this line?
   while ($line =∼ /href="([∧"]*)"/g) {
     # yes:  put the uri in $uri
     $uri = $1;
```

Code attempt two (3)

```
      # add the current file name to location URIs
      if ($uri =∼ /∧#/) { $uri =∼ s/∧/$file/; }
      # add the current directory name to file URIs
      if ($uri !∼ /∧[a-zA-Z]*:/ and $uri !∼ /∧\//) {
        $uri =∼ s/∧/$dir/; }
      # add the current host name to directory URIs
      if ($uri !∼ /∧[a-zA-Z]*:/) { $uri =∼ s/∧/$host/;}
      # print the result
      print "$uri\n";
   }
}
```

Testing code attempt two

This code performs well on the UvT home page: outputs 80 absolute URIs

But it cannot handle every (possibly incorrect) web page.

For example, a common problem in web page code is missing quotes.

It takes too much time and effort that can handle every input so we will leave the program as it is.

EXERCISES WEEK 2

Exercise results

mark	1	2	3	4	5	mark	1	2	3	4	5
9.4	♣	♣	♠	♠	♣	7.5	♠	♣	♣		
9.2	♠	♠	♠	♠	♣	7.5	♠	♣	♣		
8.8	♣	♠	♣	♠	♠	7.2	♠	♣	♣		
8.8	♠	♠	♠	♠	♠	6.9	♠	♣	♣		
8.6	♣	♠	♣	♠		5.9	♠	♠	♣		
8.4	♠	♣	♣	♠	♠	4.3	♣	♠	♠		
8.3	♣	♠	♣	♠		4.0	♠	♠	♠		
8.0	♣	♣	♣			4.0	♠	♠	♠		
8.0	♣	♣	♣			0.0					
8.0	♣	♣	♣								

♣ = perfect; ♠ = one or more errors

START WITH EXERCISES AT

http://ifarm.nl/erikt/perl2007/