Research Skills 1: Programming
Lesson 6

Erik Tjong Kim Sang
Herman Stehouwer

11 October 2007

Week 1: Variables and number processing

- 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 ()`

Week 3: String processing

- Basic string operations: concatenation (`.`); testing (`eq` and others)
- String substitution: `tr///`, `s///` and `//`
- Regular expressions: special tokens like `\s`, `$` and `*`

Week 4: Lists and hashes

- List basics: `@list = (1,2,3)` and `$list[0] = "a"`
- List functions: e.g. `push(),pop()` and `sort()`
- Hash basics: `%hash = qw(key1 value1 key2 value2)`
  `$hash{"key3"} = "value3"`
- Hash functions: e.g. `keys(),values()` and `exists()`

Week 5: Subroutines

- subroutines are blocks of code with names
- subroutine calls look like: `&subroutine(arg1,arg2,...)`
- inside a subroutine, the argument list is called `@_`
- subroutines return lists: `return(value1,value2,...)`
- subroutines can contain local variables: `my $local`

FILE HANDLING

File handling: today's overview

- file formats: character encodings
- file operations: open, read, write, close
- other topics: file handles, system commands
- programming tips: error handling and testing

File encodings

We work with text files which can be encoded in several ways:

- ASCII: seven-bit encoding covering the usual keyboard characters
- ISO 8859-1: eight-bit encoding which includes diacritics
- UTF-8: 32-bit encoding covering characters from many languages

Document structure

Apart from characters, documents also contain different structures: headings, paragraphs, lists, tables, ... which can also defined in different ways:

- with human-readable text codes, like in HTML
- with binary codes, like in Word's DOC format

File operations: opening

- open(INFILE,"myfile"): reading
- open(OUTFILE,">myfile"): writing
- open(OUTFILE,">>myfile"): appending
- open(INFILE,"someprogram |"): reading from program
- open(OUTFILE,"| someprogram"): writing to program
- opendir(DIR,"mydir"): open directory

File handles

- INFILE and OUTFILE mentioned above are file handles
- these are file variables: conventionally with CAPITAL names
- there are three special file handles: STDIN, STDOUT and STDERR
- STDIN is the familiar input handle; other two are for output
- use STDERR for sending error messages to

File operations: reading

- $a = <INFILE>: read a line from INFILE into $a
- @a = <INFILE>: read all lines from INFILE into @a
- $a = readdir(DIR): read a filename from DIR into $a
- @a = readdir(DIR): read all filenames from DIR into @a
- read(INFILE,$a,$length): $length characters from INFILE into $a

File operations: writing and closing

- print OUTFILE ”text”: write some text in OUTFILE
- close(FILE): close a file
- closedir(DIR): close a directory

More file instructions (1)

- binmode(HANDLE): change file mode from text to binary
- unlink(”myfile”): delete file myfile
- rename(”file1”,”file2”): change name of file file1 to file2
- mkdir(”mydir”): create directory mydir
- rmdir(”mydir”): delete directory mydir

More file instructions (2)

- chdir(”mydir”): change the current directory to mydir
- system(”command1”): execute command command1
- die(”message”): exit program with message message
- warn(”message”): warn user about problem message
- Example: open(INFILE,”myfile”) or die(”cannot open myfile!”)

More file instructions (3)

```
print <<"EOF";
this text will be printed
the fact that it spans four lines
is no problem
variable $x will be evaluated as well
EOF

# this will print "1 1.234   1.23\n"
printf "%d %-7s %4.2f\n",1.234,1.234,1.234;
```

ERROR MESSAGES AND TESTING

Error messages and warnings

After you have made a plan for your program and written the first version of the code, you need to check if your code is working properly.

Perl can assist you in this task by providing error messages for instructions that are invalid and warnings for code that it finds suspicious.

In order to benefit most from these diagnostic messages, always start your programs with `use strict` and always run Perl with the warning option: `perl -w program.pl`

Example program (version 1)

```
# program.pl
use strict;
$a = 1
$b = 2
$c = $a/$b
printf "%d",$c

# example run
erikt@stuwww:~$ perl -w program.pl
Scalar found where operator expected at line 4, near "$b"
(Missing semicolon on previous line?)
syntax error at program.pl line 4, near "$b "
```

Example program (version 2)

```
# program.pl
use strict;
$a = 1;
$b = 2;
$c = $a/$b;
printf "%d",$c;

# example run
erikt@stuwww:~$ perl -w program.pl
Global symbol "$c" requires explicit package name at line 5.
Global symbol "$c" requires explicit package name at line 6.
Execution of program.pl aborted due to compilation errors.
```

Example program (version 3)

```
# program.pl
use strict;
my $a = 1;
my $b = 2;
my $c = $a/$b;
printf "%d",$c;

# example run
erikt@stuwww:~$ perl -w program.pl
0erikt@stuwww:~$
```

Example program (version 4)

```
# program.pl
use strict;
my $a = 1;
my $b = 2;
my $c = $a/$b;
printf "%d\n",$c;

# example run with debugger
erikt@stuwww:~$ perl -d program.pl
Loading DB routines from perl5db.pl version 1.27
Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.
```

Example program (debugging version 4)

```
main::(program.pl:3):  my $a = 1;
DB<1> n
main::(program.pl:4):  my $b = 2;
DB<1> n
main::(program.pl:5):  my $c = $a/$b;
DB<1> n
main::(program.pl:6):  printf "%d\n",$c;
DB<1> p$a
1
DB<3> p$c
0.5
```

Example program (version 5)

```
# program.pl
use strict;
my $a = 1;
my $b = 2;
my $c = $a/$b;
printf "%3.1f\n",$c;

# example run
erikt@stuwww:~$ perl -w program.pl
0.5
erikt@stuwww:~$
```

About testing your code

- use arguments for passing data to subroutines
- use `return()` for passing data out of subroutines
- this way you can test each subroutine separately
- always test boundary cases: no input, single input (lowest/highest)

EXERCISES WEEK 5

Exercise results week 5

mark	1	2	3	4	5
9.2	♣	♠	♠	♣	♠
9.1	♠	♠	♠	♠	♠
8.5	♠	♠	♠	♠	
8.0	♣	♠	♠	♠	♠
7.2	♠	♠	♠	♠	
6.7	♠	♠	♠		
4.8	♣	♠	♠		
2.9	♠	♠			
2.1	♠	♠			

mark	1	2	3	4	5
2.1	♠	♠			
2.1	♠	♠	♠		
1.9	♠	♠	♠		
1.6	♠	♠			
0.0	♠	♠	♠		
0.0	♠	♠			
0.0	♠	♠			
0.0	♠				
0.0					

♣ = perfect; ♠ = one or more errors

Average marks for the first five exercises

9.3	9.3	8.3	7.5	7.0	6.8
6.0	5.5	5.1	4.8	4.7	4.6
4.4	4.2	4.1	4.1	3.7	2.8

Tips for the final exercises

- examine the stuff in the lecture notes
- start with the exercises as soon as possible
- ask for help when you do not understand something
- attend the extra lab class at Monday 14:45

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