CS571: Programming Languages

Exam 1

- Date: March 2nd (Friday)
- Time: 3:30pm – 5pm
- Place: LH008
- Materials: 01-05 (compiler, basic semantics, perl)
- Close book, close notes
- Triple-seating (two empty seats between two students)

What is Perl?

- Perl is a high-level, general-purpose, interpreted programming language
  - Runs on Windows, Unix, Linux and MacOS
- Originally developed by Larry Wall in 1987 as a general-purpose Unix scripting language.
- Perl borrows features from other programming languages including C, Shell scripting, AWK, and sed.

Applications of Perl

- Widely used for Common Gateway Interface (CGI) programming
  - CGI: used by web servers to run external programs (CGI scripts), most often to generate web content dynamically.
  - High-traffic websites that use Perl extensively include amazon, bbc.co.uk, priceline.com, imdb, craigslist etc.
- Text processing, e.g. reformatting text files, implementing simple search-and-replace operations etc.

The Basic Hello World Program (hello.pl)

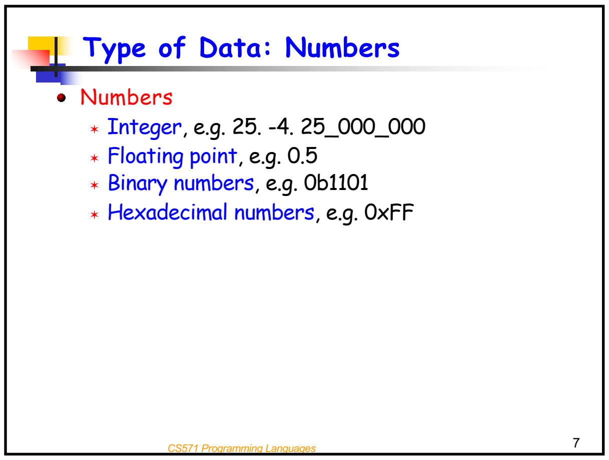
- Comments begin with a # character
- All statements end with ;
- A simple example:

```
#!/usr/bin/perl
print "Hello World!\n";
```

- Execution: `perl hello.pl`

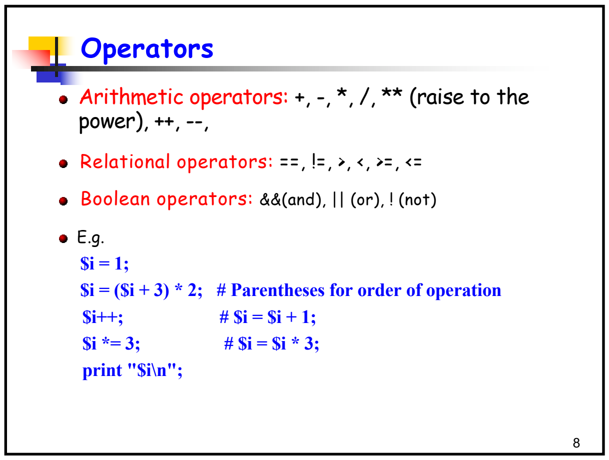
Types of Data: Scalar Variables

- Scalar variables: store a single value
  - $ followed by a letter or _, e.g. $a, $b, $c, $_
  - Up to 251 letters, digits, and _
- Scalar variables are case-sensitive
  e.g. $user is different from $User
- Variable type (int, char, …) is decided at run time
  - $a = 5; # now an integer
  - $a = "perl"; # now a string



Type of Data: Numbers

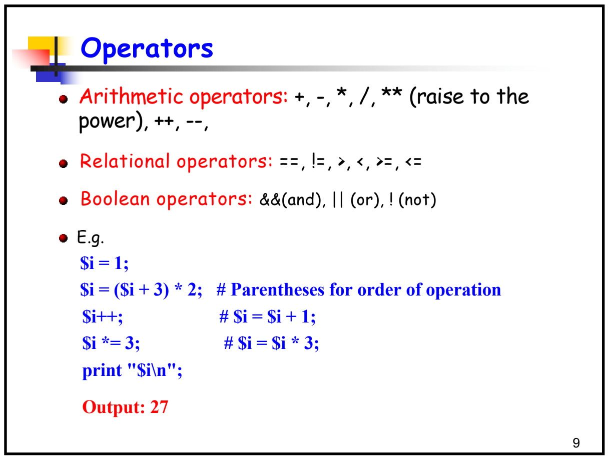
- Numbers
  - Integer, e.g. 25. -4. 25_000_000
  - Floating point, e.g. 0.5
  - Binary numbers, e.g. 0b1101
  - Hexadecimal numbers, e.g. 0xFF



Operators

- Arithmetic operators: +, -, *, /, ** (raise to the power), ++, --,
- Relational operators: ==, !=, >, <, >=, <=
- Boolean operators: &&(and), || (or), ! (not)
- E.g.

```
$i = 1;
$i = ($i + 3) * 2;   # Parentheses for order of operation
$i++;                # $i = $i + 1;
$i *= 3;             # $i = $i * 3;
print "$i\n";
```

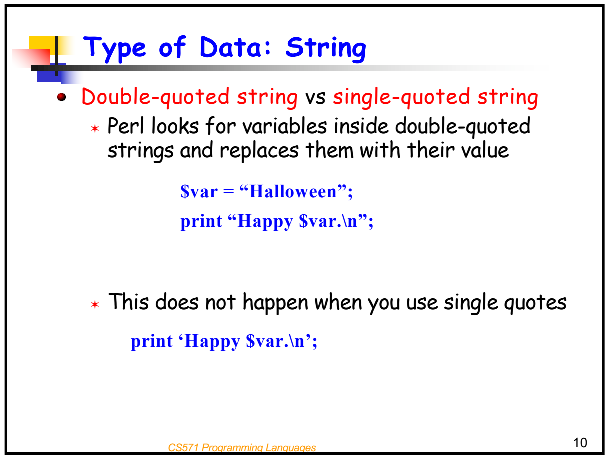


Operators

- Arithmetic operators: +, -, *, /, ** (raise to the power), ++, --,
- Relational operators: ==, !=, >, <, >=, <=
- Boolean operators: &&(and), || (or), ! (not)
- E.g.

```
$i = 1;
$i = ($i + 3) * 2;   # Parentheses for order of operation
$i++;                # $i = $i + 1;
$i *= 3;             # $i = $i * 3;
print "$i\n";
```

Output: 27



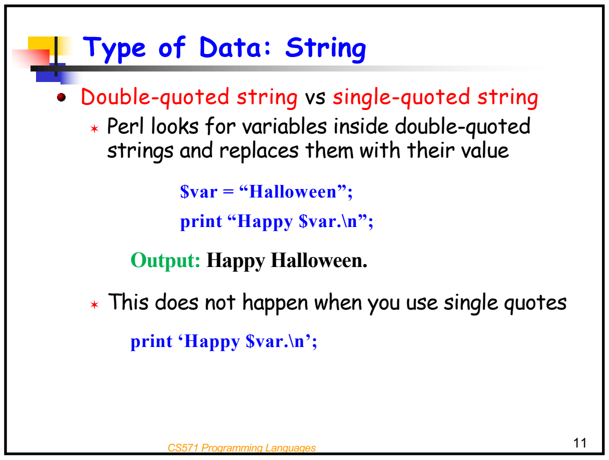
Type of Data: String

- Double-quoted string vs single-quoted string
  - Perl looks for variables inside double-quoted strings and replaces them with their value

```
$var = “Halloween”;
print “Happy $var.\n”;
```

  - This does not happen when you use single quotes

```
print ‘Happy $var.\n’;
```



Type of Data: String

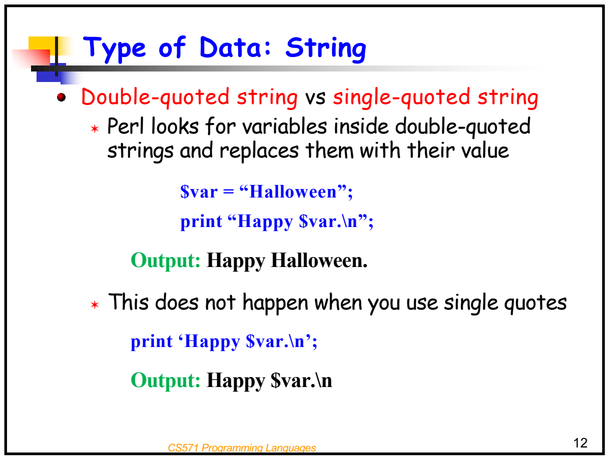
- Double-quoted string vs single-quoted string
  - Perl looks for variables inside double-quoted strings and replaces them with their value

```
$var = “Halloween”;
print “Happy $var.\n”;
```

Output: Happy Halloween.

  - This does not happen when you use single quotes

```
print ‘Happy $var.\n’;
```



Type of Data: String

- Double-quoted string vs single-quoted string
  - Perl looks for variables inside double-quoted strings and replaces them with their value

```
$var = “Halloween”;
print “Happy $var.\n”;
```

Output: Happy Halloween.

  - This does not happen when you use single quotes

```
print ‘Happy $var.\n’;
```

Output: Happy $var.\n

Operators (concat.pl)

- String operators
  - Concatenation: "."

```
$first_name = "Tom";
$last_name = "Smolka";
$full_name = $first_name . " " . $last_name;
 print $full_name
```

  - Repetition: "marked by x

```
print "Ba" . "na"x4, "\n"
```

13

Operators (concat.pl)

- String operators
  - Concatenation: "."

```
$first_name = "Tom";
$last_name = "Smolka";
$full_name = $first_name . " " . $last_name;
 print $full_name
```

output: Tom Smolka

  - Repetition: "marked by x

```
print "Ba" . "na"x4, "\n"
```

14

Operators (concat.pl)

- String operators
  - Concatenation: "."

```
$first_name = "Tom";
$last_name = "Smolka";
$full_name = $first_name . " " . $last_name;
 print $full_name
```

output: Tom Smolka

  - Repetition: "marked by x

```
print "Ba" . "na"x4, "\n"
```

Output: Bananananana

15

Relational Operators for Strings (gt.pl)

- Equal: eq
- Greater than: gt
- Greater than or equal to: ge
- Less than: lt
- Less than or equal to: le

```
$language = "Perl";
if ($language == "Perl") ...  Wrong!
if ($language eq "Perl") ...  Correct

$name1 = "abc";  $name2 = "bca";
if ($name1 gt $name2) {print "greater";}
if ($name1 lt $name2) {print "smaller";}
```

16

Relational Operators for Strings (gt.pl)

- Equal: eq
- Greater than: gt
- Greater than or equal to: ge
- Less than: lt
- Less than or equal to: le

```
$language = "Perl";
if ($language == "Perl") ...  Wrong!
if ($language eq "Perl") ...  Correct

$name1 = "abc";  $name2 = "bca";
if ($name1 gt $name2) {print "greater";}
if ($name1 lt $name2) {print "smaller";}
```

Output: smaller

17

String Functions

- Convert to upper case: uc
- Convert only the first char to upper case: ucfirst
- Convert to lower case: lc
- Convert only the first char to lower case: lcfirst

```
$name = "abc"; $name = ucfirst($name); print $name, "\n";
$name = uc($name); print $name, "\n";
$name = lcfirst($name); print $name, "\n";
$name = lc($name); print $name, "\n";
```

18

String Functions

- Convert to upper case: uc
- Convert only the first char to upper case: ucfirst
- Convert to lower case: lc
- Convert only the first char to lower case: lcfirst

```
$name = "abc"; $name = ucfirst($name); print $name, "\n";
$name = uc($name); print $name, "\n";
$name = lcfirst($name); print $name, "\n";
$name = lc($name); print $name, "\n";
```

```
Output:
Abc
ABC
aBC
abc
```

19

Type of Data: Array

- Array variable is denoted by the @ symbol
  - @array = ("good", "afternoon");
  - @array = ();
  - @array = (10..20);

20

Type of Data: Array (array.pl)

- Indexed by number
  - Index starts at 0
  - To access one element of the array : $array[$index]
  - Why? Because every element in the array is scalar

```
@array = (1..5);
print "$array[0]\n";


print "$array[8]\n"


print "$array[-1]\n"
```

21

Type of Data: Array (array.pl)

- Indexed by number
  - Index starts at 0
  - To access one element of the array : $array[$index]
  - Why? Because every element in the array is scalar

```
@array = (1..5);
print "$array[0]\n";
Output:  1

print "$array[8]\n"


print "$array[-1]\n"
```

22

Type of Data: Array (array.pl)

- Indexed by number
  - Index starts at 0
  - To access one element of the array : $array[$index]
  - Why? Because every element in the array is scalar

```
@array = (1..5);
print "$array[0]\n";
Output:  1

print "$array[8]\n"
Output:

print "$array[-1]\n"
```

23

Type of Data: Array (array.pl)

- Indexed by number
  - Index starts at 0
  - To access one element of the array : $array[$index]
  - Why? Because every element in the array is scalar

```
@array = (1..5);
print "$array[0]\n";
Output:  1

print "$array[8]\n"
Output:

print "$array[-1]\n" # print 5
Output: 5
```

24

Type of Data: Array (array1.pl)

- Access all elements in the array

```
@array = (1 ..20);
print @array;
```

- Accessing multiple elements in the array

```
print @array[3, 4, 5..7];
```

- To find the index of the last element in an array

```
print $#array;

@ numbers = ( );
print $#numbers;
```

25

Type of Data: Array (array1.pl)

- Access all elements in the array

```
@array = (1 ..20);
print @array;
Output: 1234567891011121314151617181920
```

- Accessing multiple elements in the array

```
print @array[3, 4, 5..7];
```

- To find the index of the last element in an array

```
print $#array;

@ numbers = ( );
print $#numbers;
```

26

Type of Data: Array (array1.pl)

- Access all elements in the array

```
@array = (1 ..20);
print @array;
Output: 1234567891011121314151617181920
```

- Accessing multiple elements in the array

```
print @array[3, 4, 5..7];
Output :45678
```

- To find the index of the last element in an array

```
print $#array;

@ numbers = ( );
print $#numbers;
```

27

Type of Data: Array (array1.pl)

- Access all elements in the array

```
@array = (1 ..20);
print @array;
Output: 1234567891011121314151617181920
```

- Accessing multiple elements in the array

```
print @array[3, 4, 5..7];
Output :45678
```

- To find the index of the last element in an array

```
print $#array;
Output: 19

@ numbers = ( );
print $#numbers;
```

28

Type of Data: Array (array1.pl)

- Access all elements in the array

```
@array = (1 ..20);
print @array;
Output: 1234567891011121314151617181920
```

- Accessing multiple elements in the array

```
print @array[3, 4, 5..7];
Output :45678
```

- To find the index of the last element in an array

```
print $#array;
Output: 19

@ numbers = ( );
print $#numbers;
Output: -1
```

29

Arrays: Quoted Word

- Quoted word lists using qw operator

```
@fruits = ("apples", "bananas", "cherries");

@fruits = qw(apples bananas cherries); # Same as above
```

30

Array Operations (test1.pl)

- Can dynamically lengthen or shorten arrays
- To append to the end of an array : push

```
@array = qw(red blue green);
push (@array, "black" );
print $array[3];
```

- To remove the last element of the array: pop

```
$element = pop @array;
print $element;
```

31

Array Operations (test1.pl)

- Can dynamically lengthen or shorten arrays
- To append to the end of an array : push

```
@array = qw(red blue green);
push (@array, "black" );
print $array[3];
```

Output: black

- To remove the last element of the array: pop

```
$element = pop @array;
print $element;
```

32

Array Operations (test1.pl)

- Can dynamically lengthen or shorten arrays
- To append to the end of an array : push

```
@array = qw(red blue green);
push (@array, "black" );
print $array[3];
```

Output: black

- To remove the last element of the array: pop

```
$element = pop @array;
print $element;
```

Output: black

@array now contains ("red", "blue", "green")

33

Arrays Operations (test1.pl)

- unshift: to prepend to the beginning of an array

```
@array = qw( red blue green);
unshift (@array, "black");
```

- To remove the first element of the array

```
$element = shift @array;
print $element;  # prints "black"
```

34

Arrays Operations (test1.pl)

- unshift: to prepend to the beginning of an array

```
@array = qw( red blue green);
unshift (@array, "black");
```

The array now contains "black", "red", "blue", "green"

- To remove the first element of the array

```
$element = shift @array;
print $element;  # prints "black"
```

35

Arrays Operations (test1.pl)

- unshift: to prepend to the beginning of an array

```
@array = qw( red blue green);
unshift (@array, "black");
```

The array now contains "black", "red", "blue", "green"

- To remove the first element of the array

```
$element = shift @array;
print $element;  # prints "black"
```

The array now contains "red", "blue", "green"

36

Arrays Operations (splice.pl)

- splice: cut out and return a chunk or portion of an array

```
splice(@ARRAY, INDEX, LENGTH, @REPLACE_WITH);

@fruits = qw(Banana Orange Apple Mango);
@removed = splice(@fruits, 1, 2);
print @fruits, "\n";
print @removed, "\n";
```

Arrays Operations (splice.pl)

- splice: cut out and return a chunk or portion of an array

```
splice(@ARRAY, INDEX, LENGTH, @REPLACE_WITH);

@fruits = qw(Banana Orange Apple Mango);
@removed = splice(@fruits, 1, 2);
print @fruits, "\n";
print @removed, "\n";
```

```
BananaMango
OrangeApple
```

Arrays: foreach

- Foreach allows you to iterate over an array
  Example:

```
@array = (1..5);
foreach $element (@array)
{   print "$element\n"; }
```

Arrays: foreach

- Foreach allows you to iterate over an array
  Example:

```
@array = (1..5);
foreach $element (@array)
{   print "$element\n"; }
```

Output:

```
1
2
3
4
5
```

Adding to An Arrays (Array2.pl)

```
@array1 = (1, 2, 3);
@array2 = (@array1, 4, 5, 6);
print @array2;
```

Adding to An Arrays (Array2.pl)

```
@array1 = (1, 2, 3);
@array2 = (@array1, 4, 5, 6);
print @array2;
```

Output: 123456

Types of Data: Hash (hash.pl)

- Each entry of a hash contains two components: Key and Value .
- The Hash is denoted with % E.g.

```
%data = ('John' => 45, 'Lisa'=> 30, ‘Tom’ => 40);
```

- Elements are accessed using {} (like [] in arrays)

```
print "$data{'John'}\n";
print "$data{'Lisa'}\n";
print "$data{‘Tom'}\n";
```

43

Types of Data: Hash (hash.pl)

- Each entry of a hash contains two components: Key and Value .
- The Hash is denoted with % E.g.

```
%data = ('John' => 45, 'Lisa'=> 30, ‘Tom’ => 40);
```

- Elements are accessed using {} (like [] in arrays)

```
print "$data{'John'}\n";
print "$data{'Lisa'}\n";
print "$data{‘Tom'}\n";
```

```
output:
45
30
40
```

44

Types of Data: Hash (hash.pl)

```
%data = ('John' => 45, 'Lisa'=> 30, ‘Tom’ => 40);
```

- Adding a new key-value pair
  `$data{‘Mary’} =  20`
- Each key can have only one value
  `$data{‘Mary’} =  25`

45

Types of Data: Hash (hash.pl)

```
%data = ('John' => 45, 'Lisa'=> 30, ‘Tom’ => 40);
```

- Adding a new key-value pair
  `$data{‘Mary’} =  20`
- Each key can have only one value
  `$data{‘Mary’} =  25`
  # overwrites previous assignment
- Multiple keys can have the same value
- Deleting a key-value pair
  `delete $data{‘John’}`

46

Types of Data: Hash (hash.pl)

- keys returns a list of the keys
- values returns a list of the values

```
%data = ('John' => 45, 'Lisa'=> 30, ‘Tom’ => 40);
```

Accessing all keys
```
print keys %data;
```

Accessing all values
```
print values %data;
```

Accessing all key-value pairs
```
for (keys %data) {print $_; print “$data{$_} \n”};
```

47

Types of Data: Hash (hash.pl)

- keys returns a list of the keys
- values returns a list of the values

```
%data = ('John' => 45, 'Lisa'=> 30, ‘Tom’ => 40);
```

Accessing all keys
```
print keys %data;
Output: LisaJohnTom
```

Accessing all values
```
print values %data;
```

Accessing all key-value pairs
```
for (keys %data) {print $_; print “$data{$_} \n”};
```

48

Types of Data: Hash (hash.pl)

- keys returns a list of the keys
- values returns a list of the values

```
%data = ('John' => 45, 'Lisa'=> 30, ‘Tom’ => 40);
```

Accessing all keys

```
print keys %data;
Output: LisaJohnTom
```

Accessing all values

```
print values %data;
Output: 304540
```

Accessing all key-value pairs

```
for (keys %data) {print $_; print “$data{$_} \n”};
```

Types of Data: Hash (hash.pl)

- keys returns a list of the keys
- values returns a list of the values

```
%data = ('John' => 45, 'Lisa'=> 30, ‘Tom’ => 40);
```

Accessing all keys

```
print keys %data;
Output: LisaJohnTom
```

Accessing all values

```
print values %data;
Output: 304540
```

Accessing all key-value pairs

```
for (keys %data) {print $_; print “$data{$_} \n”};
Output: Lisa30
John45
Tom40
```

Check If a Key is in the Hash (hash.pl)

```
%data = ('John' => 45, 'Lisa'=> 30, 'Tom' => 40);

$s = "John";
for (keys %data) {if ($s eq $_) {print "match";};};
```

Check If a Key is in the Hash

```
%data = ('John' => 45, 'Lisa'=> 30, 'Tom' => 40);

$s = "John";
for (keys %data) {if ($s eq $_) {print "match";};};

Output: match
```

Scope (test11.pl)

- Lexical variable: my $variable

```
$record = 4;
print “record is “, $record, “\n”;
{  my $record;
   $record = 7;
   print “inside the block, record is “, $record, “\n”;
}
print “exit the block, record is “, $record, “\n”;
```

Scope (test11.pl)

- Lexical variable: my $variable

```
$record = 4;
print “record is “, $record, “\n”;
{  my $record;
   $record = 7;
   print “inside the block, record is “, $record, “\n”;
}
print “exit the block, record is “, $record, “\n”;

Output:
     record is 4
     inside the block, record is 7
     exit the block, record is 4
```

Control Structures

55

Conditional Statements (if.pl)

- Conditional statements
  - if, elsif, else
  - Unless, elsif, else

```
$weather = “Sun”;
if ( $weather eq “Rain” )  { print “Umbrella!\n”; }
elsif ( $weather eq “Sun” ) {print “Sunglasses!\n”;}
else {print “Anti Radiation Armor!\n”;}
```

56

Conditional Statements (if.pl)

- Conditional statements
  - if, elsif, else
  - Unless, elsif, else

```
$weather = “Sun”;
if ( $weather eq “Rain” )  { print “Umbrella!\n”; }
elsif ( $weather eq “Sun” ) {print “Sunglasses!\n”;}
else {print “Anti Radiation Armor!\n”;}
```

Output: Sunglasses

57

Conditional Statements (unless.pl)

- **unless** statements are the opposite of if ... else statements.
  - Equivalent to **if (not $boolean)**

```
$weather = “Rain”;
unless ($weather eq “Rain”) {
      print “Dress as you wish!\n”; }
else {print “Umbrella!\n”;}
```

- And remember the braces are required!

58

Conditional Statements (unless.pl)

- **unless** statements are the opposite of if ... else statements.
  - Equivalent to **if (not $boolean)**

```
$weather = “Rain”;
unless ($weather eq “Rain”) {
      print “Dress as you wish!\n”; }
else {print “Umbrella!\n”;}
```

Output: Umbrella

- And remember the braces are required!

59

While Loop (while.pl)

- While: Loops when the boolean expression is true
- Example :

```
$i = 0;
while ( $i <= 1000 ) {
   print “$i\n”;
   $i++;
}
```

60

While Loop (while.pl)

- While: Loops when the boolean expression is true
- Example :

```
$i = 0;
while ( $i <= 1000 ) {
   print "$i\n";
   $i++;
}
```

Output: 0--1000

Until Loop (until.pl)

- until: evaluates an expression repeatedly until a specific condition is met.
  - Loops until boolean is true
  - Opposite of while
- Example:

```
$i = 0;
until ($i == 1000) {
   print "$i\n";   $i++;
 }
```

Until Loop (until.pl)

- until: evaluates an expression repeatedly until a specific condition is met.
  - Loops until boolean is true
  - Opposite of while
- Example:

```
$i = 0;
until ($i == 1000) {
   print "$i\n";   $i++;
 }
```

Output: 0 -- 999

For Loops

- for loop
  - Like C: for (initialization; condition; increment)

Example:

```
for ( $i = 0; $i <= 1000; $i=$i+2 ) {
      print "$i\n";
  }
```

For Loops

- for loop
  - Like C: for (initialization; condition; increment)

Example:

```
for ( $i = 0; $i <= 1000; $i=$i+2 ) {
      print "$i\n";
  }
```

Output: 0, 2, 4, ..., 1000

Moving around in a Loop (next.pl)

- next: ignore the current iteration
- last: terminates the loop.
- Example

```
for ( $i = 0; $i < 10; $i++) {
      if ($i == 1 || $i == 3) { next; }
      elsif($i == 5) { last; }
          else {print "$i\n";}
  }
```

Moving around in a Loop (next.pl)

- next: ignore the current iteration
- last: terminates the loop.
- Example

```
for ( $i = 0; $i < 10; $i++) {
      if ($i == 1 || $i == 3) { next; }
      elsif($i == 5) { last; }
          else {print "$i\n";}
  }
```

Output:

```
0
2
4
```

Regular Expression

Regular Expressions

- Regular expressions perform textual pattern matching
- Does a string
  - contain the letters "dog" in order?
  - not contain the letter "z"?
  - begin with the letters "Y" or "y"?
  - end with a question mark?
  - contain only letters?
  - contain only digits?

Match Operator (match.pl)

- m/PATTERN/ or /PATTERN/ - the match operator

  `if($word=~m/ing/) { print "$word\n";}`

- =~ : return true if the string matches the regular expression
- !~: return true if string doesn't match.
- Match line position
  - ^ start of a line
  - $ end of a line

E.g. `$word=~m/ing$/`

Match Operator (match.pl)

```
$word = "going home";
print $word, "\n";
if ($word=~m/ing/) { print "match\n";}
if ($word=~m/^ing/) { print "start with ing\n";}
if ($word=~m/ing$/) { print "end with ing\n";}
```

Match Operator (match.pl)

```
$word = "going home";
print $word, "\n";
if ($word=~m/ing/) { print "match\n";}
if ($word=~m/^ing/) { print "start with ing\n";}
if ($word=~m/ing$/) { print "end with ing\n";}
```

Output:

```
going home
match
```

Match Operator (match.pl)

```
$word = "home going";
print $word, "\n";
if ($word=~m/ing/) { print "match\n";}
if ($word=~m/^ing/) { print "start with ing\n";}
if ($word=~m/ing$/) { print "end with ing\n";}
```

Match Operator (match.pl)

```
$word = "home going";
print $word, "\n";
if ($word=~m/ing/) { print "match\n";}
if ($word=~m/^ing/) { print "start with ing\n";}
if ($word=~m/ing$/) { print "end with ing\n";}
```

Output:

```
home going
match
end with ing
```

Match Operator (match.pl)

```
$word = "ing";
print $word, "\n";
if ($word=~m/ing/) { print "match\n";}
if ($word=~m/^ing/) { print "start with ing\n";}
if ($word=~m/ing$/) { print "end with ing\n";}
```

Match Operator (match.pl)

```
$word = "ing";
print $word, "\n";
if ($word=~m/ing/) { print "match\n";}
if ($word=~m/^ing/) { print "start with ing\n";}
if ($word=~m/ing$/) { print "end with ing\n";}
```

Output:

```
ing
match
start with ing
end with ing
```

Ranges of Regular Expressions

- Ranges can be specified in Regular Expressions
- Match any characters in a list : [...]
  - [A-Z] Upper case letters
  - [a-z] Lower case letter
  - [A-Za-z] Upper or lower case letter
- Ranges of Digits can also be specified, e.g. [0-9]

Ranges of Regular Expressions

- Negating Ranges
  - / [^0-9] /:

Ranges of Regular Expressions

- Negating Ranges
  - / [^0-9] /:
    Match anything except a digit
  - / [^a] /:

Ranges of Regular Expressions

- Negating Ranges
  - / [^0-9] /:
    Match anything except a digit
  - / [^a] /:
    Match anything except an a
  - / ^[^A-Z] /:

Ranges of Regular Expressions

- Negating Ranges
  - / [^0-9] /:
    Match anything except a digit
  - / [^a] /:
    Match anything except an a
  - / ^[^A-Z] /:
    Match anything that starts with something other than a single upper case letter
    - First ^ :start of line
    - Second ^: negation

\$

- What if we want to look for all strings that equal to '$'
  - Use the \ symbol
  - / \$ / Regular expression to search for $
- What does the following Regular Expressions Match?
  / [ABCDEFGHIJKLMNOP\$] \$/
  / [A-P\$] \$ /

\$

- What if we want to look for all strings that equal to '$'
  - Use the \ symbol
  - / \$ / Regular expression to search for $
- What does the following Regular Expressions Match?
  / [ABCDEFGHIJKLMNOP\$] \$/
  / [A-P\$] \$ /

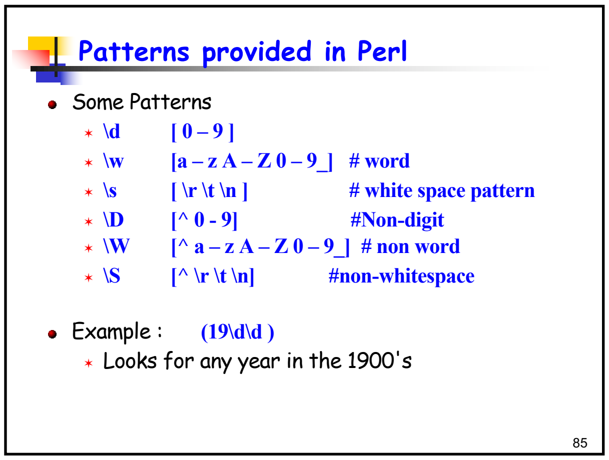
  Matches any line containing (A-P or $) followed by $

Patterns provided in Perl

- Some Patterns

\d	[0 – 9]	
\w	[a – z A – Z 0 – 9_]	# word
\s	[\r \t \n]	# white space pattern
\D	[^ 0 - 9]	#Non-digit
\W	[^ a – z A – Z 0 – 9_]	# non word
\S	[^ \r \t \n]	#non-whitespace

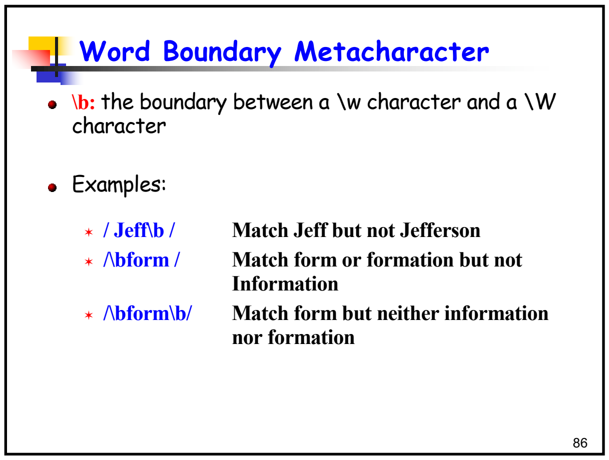
- Example : (19\d\d)



Patterns provided in Perl

- Some Patterns
  - \d [0 – 9]
  - \w [a – z A – Z 0 – 9_] # word
  - \s [\r \t \n] # white space pattern
  - \D [^ 0 - 9] #Non-digit
  - \W [^ a – z A – Z 0 – 9_] # non word
  - \S [^ \r \t \n] #non-whitespace
- Example : (19\d\d)
  - Looks for any year in the 1900's

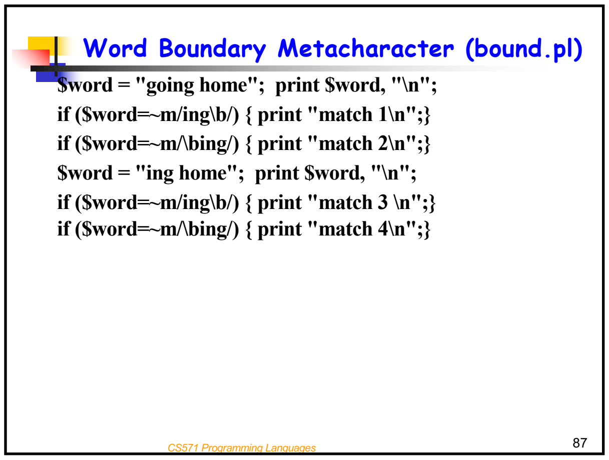
85



Word Boundary Metacharacter

- \b: the boundary between a \w character and a \W character
- Examples:
  - / Jeff\b / Match Jeff but not Jefferson
  - /\bform / Match form or formation but not Information
  - /\bform\b/ Match form but neither information nor formation

86

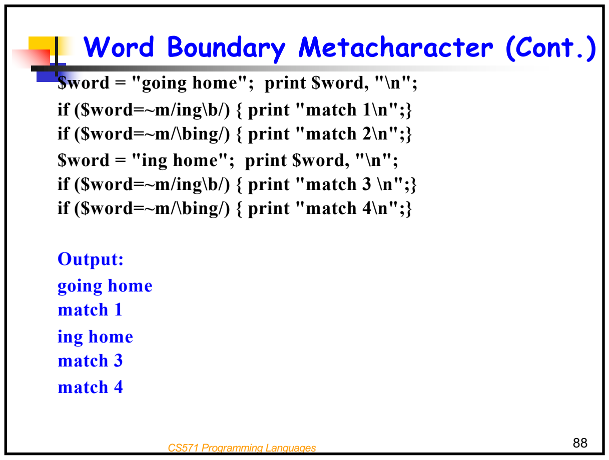


Word Boundary Metacharacter (bound.pl)

```
$word = "going home";  print $word, "\n";
if ($word=~m/ing\b/) { print "match 1\n";}
if ($word=~m/\bing/) { print "match 2\n";}
$word = "ing home";  print $word, "\n";
if ($word=~m/ing\b/) { print "match 3 \n";}
if ($word=~m/\bing/) { print "match 4\n";}
```

CS571 Programming Languages

87



Word Boundary Metacharacter (Cont.)

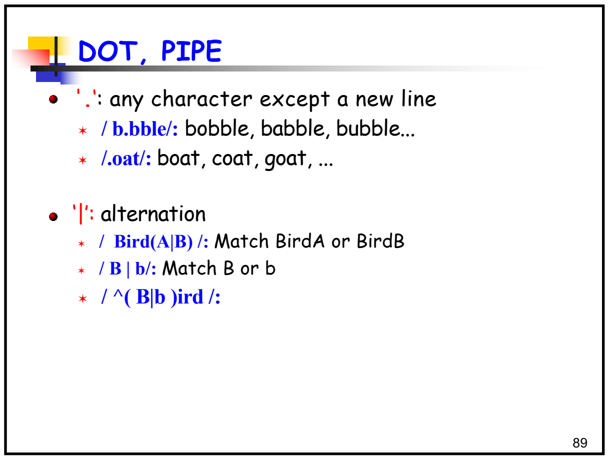
```
$word = "going home";  print $word, "\n";
if ($word=~m/ing\b/) { print "match 1\n";}
if ($word=~m/\bing/) { print "match 2\n";}
$word = "ing home";  print $word, "\n";
if ($word=~m/ing\b/) { print "match 3 \n";}
if ($word=~m/\bing/) { print "match 4\n";}
```

Output:

```
going home
match 1
ing home
match 3
match 4
```

CS571 Programming Languages

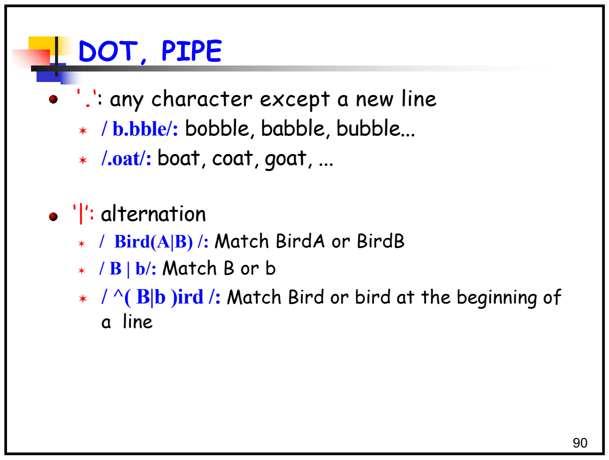
88



DOT, PIPE

- '.': any character except a new line
  - / b.bble/: bobble, babble, bubble…
  - /.oat/: boat, coat, goat, …
- '|': alternation
  - / Bird(A|B) /: Match BirdA or BirdB
  - / B | b/: Match B or b
  - / ^(B|b)ird /:

89



DOT, PIPE

- '.': any character except a new line
  - / b.bble/: bobble, babble, bubble…
  - /.oat/: boat, coat, goat, …
- '|': alternation
  - / Bird(A|B) /: Match BirdA or BirdB
  - / B | b/: Match B or b
  - / ^(B|b)ird /: Match Bird or bird at the beginning of a line

90

?, *, +

- ?: the character occurs zero or one time
  /worl?ds/: match either worlds or words
- *: the character occurs zero or more times
  /ab*c/: match 'ac', 'abc', 'abbc', 'abbbc' ect...
- +: the character occurs one or more times
  /ab+c/: match 'abc', 'abbc', 'abbbc' ect...

Modifying Text With Regular Expressions

Modifying Text (sub.pl)

- Substitution: ~s
  - If there is a match, then replace it with the given string
- Example :

```
# replace the first occurrence of abc with cba
$var1 = "abced abcde";
$var1 =~s/abc/cba/;  print $var1;

# replace all occurrence of abc with cba
$var2 = "abced abcde";
$var2 =~ s/abc/cba/g;  print $var2;
```

Modifying Text (sub.pl)

- Substitution: ~s
  - If there is a match, then replace it with the given string
- Example :

```
# replace the first occurrence of abc with cba
$var1 = "abced abcde";
$var1 =~s/abc/cba/;  print $var1;
Output: cbaed abcde

# replace all occurrence of abc with cba
$var2 = "abced abcde";
$var2 =~ s/abc/cba/g;  print $var2;
```

Modifying Text (sub.pl)

- Substitution: ~s
  - If there is a match, then replace it with the given string
- Example :

```
# replace the first occurrence of abc with cba
$var1 = "abced abcde";
$var1 =~s/abc/cba/;  print $var1;
Output: cbaed abcde

# replace all occurrence of abc with cba
$var2 = "abced abcde";
$var2 =~ s/abc/cba/g;  print $var2;
Output: cbaed cbade
```

$&, $' etc (match3.pl)

- $&: contains the string matched
- $`: the text until the first match
- $': the text after the last match
- $1, $2: the text matched in the first, second parenthesis

```
$target="I have 25 apples";
if($target=~/(\d+)/) {print "match\n";}
print("$&\n"); print("$'\n"); print("$`\n");
print("$1\n");
```

$&, $' etc (match3.pl)

- $&: contains the string matched
- $`: the text until the first match
- $': the text after the last match
- $1, $2: the text matched in the first, second parenthesis

```
$target="I have 25 apples";
if($target=~/(\d+)/) {print "match\n";}
print("$&\n"); print("$'\n"); print("$`\n");
print("$1\n");
```

Output:

```
match
25
 apples
I have
25
```

$&, $' etc (match4.pl)

```
$exp =“I crave to rule the world!”;
if($exp=~/^([A-Za-z+\s]*)\bcrave\b([\sA-Za-z]+)/)
    {
    print “$1\n”;
    print “$2\n”;
    }
```

$&, $' etc (match4.pl)

```
$exp =“I crave to rule the world!”;
if($exp=~/^([A-Za-z+\s]*)\bcrave\b([\sA-Za-z]+)/)
    {
    print “$1\n”;
    print “$2\n”;
    }
```

Output:

```
I
 to rule the world
```

Subroutines

Subroutines (test9.pl)

- Subroutines are declared with the sub keyword
- Arguments are passed into the @_ array

```
sub add_one {
my ($n) = @_[0];        # Copy first argument
return ($n + 1);   }
my ($a, $b) = (10, 0);
add_one($a);            # Return value is lost
$b = add_one($a);   # $a is 10, $b is 11
```

Subroutines (test9.pl)

```
sub add_one {my ($n) = @_[0];     return ($n + 1); }
sub add_two { my ($n) = @_[0]; my ($m) = @_[1];
              return ($m + 2); }
my ($a, $b) = (10, 0);
add_one($a);
$c = add_one($a);
$d = add_two($a, $b);
print $a, "\n";
print $b, "\n";
print $c, "\n";
print $d, "\n";
```

Subroutines (test9.pl)

```
sub add_one {my ($n) = @_[0];     return ($n + 1); }
sub add_two { my ($n) = @_[0]; my ($m) = @_[1];
                return ($m + 2); }
my ($a, $b) = (10, 0);
add_one($a);
$c = add_one($a);
$d = add_two($a, $b);
print $a, "\n";
print $b, "\n";
print $c, "\n";
print $d, "\n";
```

Output: 10 0 11 2 (each # is in a different line)

@_

```
sub add {
    @_[0] = @_[0]+1;
    }
    my $a = 10;
    add($a);
    print $a, "\n";
```

@_ (ref.pl)

```
sub add {
    @_[0] = @_[0]+1;
    }
    my $a = 10;
    add($a);
    print $a, "\n";
```

Output: 11

call-by-reference

File Operation

Open a File

- open FH, `output.log' or die $!;
  - open file output.log. If the file does not exist, die and print message held in $!
  - **FH:** file descriptor
  - **$!:** I/O error message

Open a File for Writing (test7.pl)

- Create a new file or overwrite an existing file
  open FH, "> $filename" or die $!;

- To add content to the end of existing file
  Open FH, ">> $filename" or die $!;

```
open FH,  "> writetest.txt" or die $!;
print FH "abc";
open FH,  ">> writetest.txt" or die $!;
print FH "def";
```

- To close the file: **close FH**

Read a Line (test5.pl)

- Input Operator < >: reads one line from a file, including new line
- chomp: removes newline

Example:

```
print "What type of pet do you have?";
my $pet = <STDIN>;
chomp $pet;
print "You have pet $pet";
```

Read a Line (test6.pl)

- Reading from files
  - Loops will assign to $_ by default
  - Be sure that the file is opened before read

```
open FILE,  "readtest.txt" or die $!;
my $lineno = 1;
while (<FILE>) {
     print $lineno++;
     print ": $_"; }
```

Read a Line (test6.pl)

- Reading from files
  - Loops will assign to $_ by default
  - Be sure that the file is opened before read

```
open FILE,  "readtest.txt" or die $!;
my $lineno = 1;
while (<FILE>) {
     print $lineno++;
     print ": $_"; }
```

Output: content of readtest.txt with line numbers

Read A Number of Bytes (file1.pl)

- Read a number of bytes

  read FILEHANDLE, SCALAR, LENGTH
  - SCALAR: stores the characters read
  - LENGTH: the number of characters read

```
open FILE, "readtest.txt" or die $!;
my ($data, $n);
while (($n = read FILE, $data, 4) != 0)
{ print "$n bytes read\n";
  print $data, "\n";}
 close(FILE);
```

Read One Character (file2.pl)

- Read a character

  getc FILEHANDLE

```
open FILE, "readtest.txt" or die $!;
while(my $char = getc FILE)
{ print $char; }
close(FILE);
```

File Checks (test8.pl)

- File test operators check if a file exists, is readable or writable, etc.
  - -e: if the file exists
  - -r: if the file is readable
  - -w: if the file is writable
  - -x: if the file is executable
  - ……
- E.g.

```
my $filename = "test.txt";
if (-r $filename) { print "the file is readable\n")
```

Renaming/Deleting a File (file3.pl)

- Renaming a file
  rename(file1, file2);

- Deleting a file
  unlink(file);
  success: returns the number of files deleted
  failure: returns false and sets $! Errno

```
my $file = "hello.txt";
unlink $file;
if (-e $file) { print "File still exists!"; }
else { print "File gone."; }
```

115

Web Sources for Perl

- Link
  - http://www.perl.org/books/beginning-perl/
  - www.perl.com
  - www.perldoc.com
  - www.perl.org
  - www.perlmonks.org

- Perl Debugger
  - http://www.thegeekstuff.com/2010/05/perl-debugger/

116