Control Statements

- Some Useful Things
  - Comments
  - Leaving programs
  - Increment/decrement
  - Boolean operators
- Domains of Programming Languages
- Control Flow
- Branching
- Repetition
  - Indefinite
  - Definite
- Breaking Loops

Program Comments

- Start a comment with a #
- Write many comments in your program
- Write a header block in EVERY program

```
#
#  Author:   Chris Kelly
#  Name: first_name.pl
#  Date: 11 Sept 2017
#  Purpose: To solve a problem
#
```

Ways to get out of a program

- **`exit (0);`**
  - When a script finishes, it can return a numeric exit code to the caller
    - An exit code of **zero** indicates all is well/successful
    - A **non-zero** exit code can identify error conditions
  - In Unix, upon the scripts completion, the exit code will be contained in the shell variable **`$?`**
- **`die ($string);`**
  - If the problem is more severe, we can use the die function
  - Ends the program with a non-zero exit status

Ways to get out of a program

- **`die ($string);`**
  - String parameter is printed to standard error, along with other info
    - Script file name
    - Line number
    - Error location

Increment/Decrement

- A very helpful construct is the increment/decrement statement
- **`$i++`** is equivalent to **`$i = $i+1`**

```
$j = $i++
$j = ++$i
$j = $i--
$j = --$i
```

- *What differentiates* ***pre***-*increment and* ***pre***-*decrement from* ***post***?

Arithmetic Comparisons

- `$x > $y`
- `$x < $y`
- `$x >= $y`
- `$x <= $y`
- `$x == $y`
- `$x != $y`

String Comparisions

- `$x gt $y`
- `$x lt $y`
- `$x ge $y`
- `$x le $y`
- `$x eq $y`
- `$x ne $y`

Logical Operators

- `$x and &y`
- `$x && $y`

- `$x or $y`
- `$x || $y`

- `not $x`
- `!$x`

Four Domains of Programming Languages

- Input / Output
  - Keyboard
  - Terminal screen
  - Files
  - Web sockets
- Variables / Information Storage
  - Declare variable
  - Assign variable
  - Read variable
- Decision / Control Structures
  - Sequence
  - Branching
  - Repetition
- Data Manipulations / Calculations
  - Arithmetic
  - String concatenation
  - Boolean logic

Control Flow Constructs

- What is a control statement?
- Types of control statements:
  - Branching: ***`if`***
  - Repetition: ***`while`*** and ***`for`***

If Statements

- We use ***branching*** to tell the program to do one thing or another, depending on some condition.
  - The condition is boolean, something that can be interpreted as true or false
  - Usually, the condition will be based on the *present state* of the program and its data
- if
  - `if ( condition ) { action }`
  - Either ***`action`*** is taken or it isn't

If Statements

- if else

```
if (condition ) { action1 }
else { action2 }
```

  - Take either ***action1*** or ***action2***

- if elsif else

```
if ( condition ) { action1 }
elsif (another condition ) { action2 }
... # more elsifs
else { default }
```

  - Take ***action1***, ***action2***, ***action3***, ... , or ***default***

If Statements

- unless statement

```
unless (condition)  {action};
```

  - Functionally equivalent to:

```
if ( ! condition ) { action }
```

- Reversed order syntax

```
print "Hello Al\n" if($inputName = "Al");
die "Can’t divide by zero\n:"  if ($num4 == 0);
```

- Both of these may be useful for making code sound more similar to human language.

While Statement

- One type of repetition we can use in a program is an <u>indefinite</u> loop.
  - Here, the idea is that the repeated execution of loop code eventually causes the loop to end
  - For this, we use <u>***while***</u> loops
- **`while ( condition ) { action }`**
- Example:

```
$i=1;
while ($i<=5) {
      print $i, "\n";
      $i++;
}
```

While Statement

- ***<u>until</u>*** loops
    - Same form but opposite action of the while loop
    - Functionally equivalent to:

```
while ( ! condition ) { action }
```

An interesting variable

- **$_** is the default variable for many functions

```
while ( $line  =  <STDIN>) {
   print $line;
   …
   }
while (<STDIN>) {
   print $_;
   …
   }
```

Another Form of Loops

```
do { action } while ( condition );
do { action } until ( condition );
```

- In other words:
  - Carry out ***action***
  - Examine ***condition***
  - Decide whether or not to repeat ***action***
- Here, ***action*** is always executed at least once!

For statement

- In contrast, for some kinds of repetition, the program can determine the number of repetitions *before the loop begins*. This could be based on:
  - A predetermined number
  - All the items within a sequence
- For such definite repetition, we use ***for*** loops

```
for ( init_exp ; test_exp; step_exp ) { action }
```

- Example:

```
for ( $i=1;  $i<5;  $i++ )  { print $i, "\n" ; }
```

Foreach loop

```
foreach  $number (1..10) {
   print "The number is: $number \n";
   }
```

- Here, we are basically saying, "Execute this block of code *for each item* in the sequence."

Modifying Loop Behavior

- There are three ways to modify the execution of the loop:
  - **`last;`** - end the loop
  - **`next;`** - skip to the next iteration
  - **`redo;`** - restart the loop
- You can use these with statement labels
  - Labels a location in the program
  - Best to use all uppercase letters; label should also be ***meaningful***

```
OUTER: while(…) {
    …
    }
```

Modifying Loop Behavior - Example

```
OUTER: while( some condition ) {
     # some code...
     INNER: foreach $num (1..20) {
          if ( some other condition ){
               last INNER;
          }
          else { next OUTER; }
     }
     #some more code
}
```

Realistically, you probably will not use these kinds of tricks often, but it is good to know that they are out there. This way, you are aware of your *options* for directing the flow of control in your scripts/programs.