Feedback

"I was pretty scared of starting computing in a uni setting but I've been enjoying how it's taught and now it's my "fun" subject this term! :D"

- Putting lecture notes single screen
- Releasing labs earlier
- Repeating questions that are asked
- Challenge lab exercises

4. C Conditions

In this lecture we will cover:

- More on linux commands
- Making Choices
- Relational Operators
- Logical Operators
- If/Else Statements

Linux Command: cp

- Linux Command **cp**: copies files and directories.
- cp *sourceFile destination*
- If the destination is an existing file, the file is overwritten
- if the destination is an existing directory
  the file is copied into the directory
- To copy a directory use `cp -r` *sourceDir destination*

Linux Command: mv

- Linux Command **mv** moves or renames a file.
- `mv` *source destination*
- If the destination is an existing file, the file is overwritten
- if the destination is an existing directory
  the file is moved into the directory.

Linux Command: rm

- Linux Command **rm** removes a file.
- Usually no undo or recycle bin - be careful & have backups
- `rm` *filename*
- `rm -r` *directoryName*
  - This will delete a whole directory.
  - Be extra careful with this command

Control Flow

Problem: "read an integer and tell me if it's between 5 and 10."

- We know how to read in an integer
- But how can we say whether it's less than 5?

What we need is a way of making choices in our programs. This functionality is known as control flow or branching and is provided by the `if` statement.

```
int x;
scanf("%d", &x);
if (x > 5 && x < 10) {
  printf("Between 5 and 10!");
}
```

Before we can use if statements properly we need to understand relational operators and logical expressions.

Relational Operators

C has the usual operators to compare numbers:

`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`!=`	not equal to
`==`	equal to

- Don't confuse equality (==) with assignment (=)
- Be careful comparing doubles for equality using == or !=. Remember doubles are approximations.

Relational Operators

- Many languages have specific "boolean" types for TRUE and FALSE
- C does not have this type, so we just use int
- C convention is zero is false, other numbers true.
- All relational and logical operators return a "boolean":
  the int **0** for false
  the int **1** for true
- For example:
  `5 > 4` $\mapsto$ 1
  `5 >= 4` $\mapsto$ 1
  `5 < 4` $\mapsto$ 0
  `5 <= 4` $\mapsto$ 0
  `5 != 4` $\mapsto$ 1
  `5 == 4` $\mapsto$ 0

Logical Operators

Logical operators allow us to combine Boolean expressions (e.g., comparisons, etc.). We use them to answer questions like "Is x greater than y and less than z?"

The logical operators are:

- and (`&&`) true if both operands are true
- or (`||`) true if either operand is true
- not (`!`) true if its operand is false

Here are some examples:

```
(2 > 0) && (2 < 2)    ↦  1 && 0  ↦  0  and
(0 > 1) || (2 < 10)   ↦  0 || 1  ↦  1  or
!(0 > 1)              ↦  !0      ↦  1  not
```

Logical Operators

Truth tables show the results of logical operators with all different combinations of inputs

X	Y	X && Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X || Y
0	0	0
0	1	1
1	0	1
1	1	1

Logical Operators / De Morgan's Law

These two conditions are logically equivalent

```
!(height <= 130 && width <= 240)
```

.. is the same as ..

```
height > 130 || width > 240
```

Logical Operators / Short Circuit Evaluation

This is an important concept, the operators `&&` and `||` evaluate their left-hand-side operand first and only evaluate their right-hand-side operand if necessary.
Operator `&&` only evaluates its RHS if the LHS is *true*.

Operator `||` only evaluates its RHS if the LHS is *false*.

This is very useful because we can safely write:

```
(x != 0) && (y / x > 10)
```

Precedence

A list of all operators in order of precedence, from high to low:

- `!x, -x`
- `x * y, x / y, x % y`
- `x + y, x - y`
- `x < y, x <= y, x > y, x >= y`
- `x == y, x != y`
- `x && y` (short-circuit left to right)
- `x || y` (short-circuit left to right)
- `x = y`

Explicit Order

The evaluation order can be changed and/or made explicit via parentheses, e.g., `7 * (4 + 3)`.

Don't Do This

Something like: `10 > x > 0` will compile (albeit with a compiler warning), but what does it mean? Suppose x = -1.

- `((10 > -1) > 0)`
- `( 1 > 0)`
- `1`

What you probably mean to write is `(10 > x) && ( x > 0)`

- `(10 > -1) && ( -1 > 0)`
- `1 && ( -1 > 0)`
- `1 && 0`
- `0`

The `if` Statement

This is the structure of the `if` statement:

```
if (expression evaluates non-zero) {
    statement1;
    statement2;
    ....
}
```

- **statement1, statement2, ...** are executed if **expression** is non-zero.
- **statement1, statement2, ...** are **NOT** executed if **expression** is zero.

The `else` keyword

```
if (expression evaluates non-zero) {
    statement1;
    statement2;
    ....
} else if (expression evaluates non-zero) {
    statement3;
    statement4;
    ....
} else {
    statement5;
    statement6;
    ....
}
```

- **statement1, statement2** executed if **expression** is non-zero.
- **statement3, statement4** executed if **expression** is zero.

The if Statement

We can also have nested if statements. ie if statements inside if statements

```
printf("%d is a ", a);
if (a < 0) {
    if (a < -100) {
        printf("big");
    } else if (a > -10){
        printf("small");
    } else {
        printf("medium");
    }
    printf(" negative");
} else {
    printf(" positive");
}
printf(" number.\n");
```

The if Statement

This syntax is also valid:

```
if (a == 0)
    printf("a is zero\n");
    a = 1; // this does not belong to if-block
```

If the braces ({}) are not supplied then the `if` statement controls only the statement that immediately follows.

Always use braces!

Doing this will ensure that you avoid bugs and ambiguity. The style guide requires it.