Conditional Execution

- many problems require executing statements only in some circumstances
  e.g read two integers and print largest one
- sometimes called **control flow**, **branching** or **conditiional execution**
- The C **if** Statement can do this.

The if Statement

```
if (expression) {
    statement1;
    statement2;
    ....
}
```

- **statement1, statement2, ...** are executed
  if **expression** is non-zero.
- **statement1, statement2, ...** are **NOT** executed
  if **expression** is zero.
- There is no "boolean" type in C.
  0 is regarded as "FALSE"
  anything non-zero is regarded as "TRUE"

The else keyword

```
if (expression) {
    statement1;
    statement2;
    ....
} else {
    statement3;
    statement4;
    ....
}
```

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

The if Statement

Multiple **if** statements can be chained together:

```
    int a, b;

    printf("Please enter two numbers, a and b: ");
    scanf("%d %d", &a, &b);

    if (a > b) {
       printf("a is greater than b\n");
    } else if (a < b) {
       printf("a is less than b\n");
    } else {
       printf("a is equal to b\n");
    }
```

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

- Be careful comparing doubles for equality using `==` or `!=`
- Remember doubles are approximations.

Relational Operators

- Many languages have a separate type for true & false.
- C just uses numbers.
- C convention is zero is false, other numbers true.
- relational operators return:
  the int **0** for false
  the int **1** for true
- For example:

```
5 > 4    ↦  1
5 >= 4   ↦  1
5 < 4    ↦  0
5 <= 4   ↦  0
5 != 4   ↦  1
5 == 4   ↦  0
```

Logical Operators

- C has logical operators: `&&` `||` `!`
- Logical operators allow us to combine comparisons, eg:
  $mark > 0$ `&&` $mark < 100$
- logical operators return:
  the int **0** for false
  the int **1** for true
- `&&` is the **and** operator - true if both operands are true
  $2 > 0$ `&&` $2 < 10 \mapsto 1$ `&&` $1 \mapsto 1$
- `||` is the **or** operator - true if either operand is true
  $24 > 42$ `||` $2 < 10 \mapsto 0$ `||` $1 \mapsto 1$
- `!` is the **not** operator - true iff its operands is false
  `!`$(24 > 42) \mapsto$ `!`$0 \mapsto 1$

Logical Operators - Conditional evaluation

- The C operator `&&` `||` have a useful property.
- They always evaluate their left-hand side first.
- They only evaluate their right-hand side if needed.
- `&&` will not evaluate right-hand side if left-hand side is false (zero).
- `||` will not evaluate right-hand side if left-hand side is true (non-zero).
- For example we can write

```
x != 0 && y/x > 2
```

without risking division by zero.

Unary Negation operator

The unary negation operator converts a non-zero operand into 0 and 0 into 1. For example,

```
    if (!(height <= 130 && width <= 240)) {
        printf("Envelope too large!\n");
    }
```

.. is the same as ..

```
    if (height > 130 || width > 240) {
        printf("Envelope too large!\n");
    }
```