Control Flow

We still don’t know how to solve a simple problem like: “read two integers and print the larger one.”

- we know how to read in two integers
- But how can we compare them and print the larger one?
Control Flow

We still don’t know how to solve a simple problem like: “read two integers and print the larger one.”

- we know how to read in two integers
- But how can we compare them and print the larger one?

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.
Control Flow

We still don’t know how to solve a simple problem like: “read two integers and print the larger one.”

- we know how to read in two integers
- But how can we compare them and print the larger one?

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.

The switch statement is another way, but for now we will just stick with if statements.
Control Flow

We still don’t know how to solve a simple problem like: “read two integers and print the larger one.”

- we know how to read in two integers
- But how can we compare them and print the larger one?

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.

The switch statement is another way, but for now we will just stick with if statements.

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 a separate type for true & false.
- C just uses numbers. Zero is false, other numbers true.
- Relational operators return:
  - the int 0 for false
  - the int 1 for true

For example:

- $5 > 4 \mapsto 1$
- $5 \geq 4 \mapsto 1$
- $5 < 4 \mapsto 0$
- $5 \leq 4 \mapsto 0$
- $5 \neq 4 \mapsto 1$
- $5 == 4 \mapsto 0$
The **if** Statement

This is the structure of the **if** statement:

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

- used to decide if `statement` should be executed
- there is no "boolean" type in C. Instead, zero is regarded as "FALSE" and anything non-zero is regarded as "TRUE"
- `statement1, statement2, ...` are executed if evaluation of `expression` is non-zero.
- `statement1, statement2, ...` are **NOT** executed if evaluation `expression` is zero.
The `else` keyword

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

- `statement1, statement2, ...` are executed if `expression` is non-zero.
- `statement3, statement4, ...` are executed if `expression` is zero.
Chained if Statements

```c
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");
}
```
Nested if Statement

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

```c
printf("%d is a ", a);
if (a < 0) {
    if (a < -100) {
        printf("big");
    } else {
        printf("small");
    }
    printf(" negative");
} 
printf(" number.\n");
```
The if Statement

This syntax is also valid:

```c
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.
The if Statement

This syntax is also valid:

```c
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.