1. Conditionals

Any programming language provides the programmers the ability to perform choices.

We want to do X in some cases, and Y in other cases.

We want to check data, and do choices based on the state of that

data.C provides us 2 ways to do so.

The first is the `if` statement, with its `else` helper, and the second is the `switch` statement.

1.1. `if`

In an `if` statement, you can check for a condition to be true, and thenexecute the block provided in the curly brackets:

```
int a = 1;

if (a == 1) {
```

You can append an `else` block to execute a different block if the original condition turns out to be false;

```
int a = 1;

if (a == 2) {
  /* do something */
} else {
  /* do something else */
}
```

Beware one common source of bugs - always use the comparison operator

`==` in comparisons, and not the assignment operator `=`, otherwise the `if` conditional check will always be true, unless the argument is `0`, for exampleif you do:

```
int a = 0;

if (a = 0) {
  /* never invoked */
}
```

Why does this happen? Because the conditional check will look for a boolean result (the result of a comparison), and the `0` number always equates to a false value. Everything else is true, including negative numbers.

You can have multiple `else` blocks by stacking together multiple `if` statements:

```
int a = 1;

if (a == 2) {
  /* do something */
} else if (a == 1) {
  /* do something else */
} else {
  /* do something else again */
}
```

1.2. `switch`

If you need to do too many if / else / if blocks to perform a check, perhaps because you need to check the exact value of a variable, then `switch` can be very useful to you.You can provide a variable as condition, and a series of `case` entry

```
int a = 1;

switch (a) {
  case 0:
    /* do something */
    break;
  case 1:
    /* do something else */
    break;
  case 2:
    /* do something else */
    break;
}
```

points for each value you expect:

We need a `break` keyword at the end of each case, to avoid the next case to be executed when the one before ends. This "cascade" effect can be useful in some creative ways.

You can add a "catch-all" case at the end, labeled `default` :

```
int a = 1;

switch (a) {
  case 0:
    /* do something */
    break;
  case 1:
    /* do something else */
    break;
  case 2:
    /* do something else */
    break;
  default:
    /* handle all the other cases */
    break;
}
```