Constants and the const keyword

The best way to define a constant is to use the `const` keyword

`const` can be added to any variable declaration with an initialization

It means that the value of the variable cannot be modified at run-time:

```
const double pi = 3.1415926;
pi = 3; // This is a compile-time error
```

Even though a `const` can't be modified, it's still a variable, located somewhere in memory - so you can take it's address

Constants in C should be **lowercase**

`const` can also be applied to pointers:

```
const int x = 5; // const int variable
const int *p = &x; // make p point to x
*p = 10; // *p cannot be modified: compile-time error
int y = 6; // normal (non-const) int variable
p = &y; // p itself can be modified...
*p = 20; // but *p cannot: compile-time error
```

`const int *p` means that p is a pointer to a constant int

`const` can also be applied to arrays

Recall that arrays are passed as basal pointers to functions, so by default the array elements can be modified. `const` marks the array as read-only - the compiler stops you modifying elements:

```
const int arr[] = {10,20,30,40,50,60}; // no array elements can be
modified
arr[3] = 10; // compile-time error
```

If you have a `const int arr[]`, its basal pointer will have type `const int`:
`const int *p = arr; // correct basal decay of const int arr[]`

Having marked `p` as a pointer to a constant int, our attempts to modify `*p` lead to compile-time errors