References

Now, we'll learn what references are and how they differ from pointers.

We'll cover the following

- References vs. pointers
- References as parameters

> A reference is an alias for an existing variable. It can be created using the `&` operator.

Once created, the reference can be used instead of the actual variable. Altering the value of the reference is equivalent to altering the referenced variable.

Note that we don't have to worry about the `*` operator when de-referencing a variable reference. For example, lines 7, 11 and 14 in the following program.

> Note that we are using `std::cout` every time we want to display something to the screen, instead of `cout` since we haven't used the `using namespace std` statement. Similarly, we are using `std::endl` instead of `endl`. Both options work the same, but avoiding the `using namespace` statement has its advantages.

```
1  #include <iostream>
```

```cpp
#include <iostream>

int main() {
  int i = 20;
  int& iRef = i;
  std::cout << "i: " << i << std::endl;
  std::cout << "iRef: " << iRef << std::endl;

  std::cout << std::endl;

  iRef = 30; // Altering the reference

  std::cout << "i: " << i << std::endl;
  std::cout << "iRef: " << iRef << std::endl;
}
```

Output 1.72s

```
i: 20
iRef: 20

i: 30
iRef: 30
```

References vs. pointers#

There is a lot of overlap between pointers and references but the two have some stark differences as well.

A reference is never `nullptr`. Therefore, it must always be initialized by having an existing variable assigned to it. The following lines would not work:

```
int& intRef;
int& intRef = nullptr;
```

References behave like constant pointers. A reference always refers to its initial variable. The value of the variable can change but the reference cannot be assigned to another variable.

Like pointers, a reference can only be initialized by a variable of the same type.

References as parameters#

References allow functions to modify the value of a variable. When a normal variable is passed to a function, a copy of its value is made and the variable itself remains untouched. However, if a reference is passed, the actual value of the variable is used and can therefore be modified.

```
#include <iostream>

void xchg(int& x, int& y){ // Reference parameters
  int t = x;
  x = y;
  y = t;
}

int main() {
  int a = 10;
  int b = 20;
  std::cout << "a: " << a << std::endl;
  std::cout << "b: " << b << std::endl;

  xchg(a, b);
  std::cout << std::endl;

  std::cout << "a: " << a << std::endl;
  std::cout << "b: " << b << std::endl;
}
```

```
Output                                                          1.01s

 a: 10
 b: 20

 a: 20
 b: 10
```

Compared to when using pointers, notice the cleaner syntax. On line 15, if using pointers, we would have passed `&a` and `&b` as arguments. Similarly, lines 4 through 6 would have to be written as:

```
int t = *x;
*x = *y;
*y = t;
```

Also, line 3 would have been `void xchg(int* x, int* y)`.

In short, references provide a more convenient way to access variables from inside other functions. This functionality is also very useful when dealing with a large argument. Passing it by value would mean that a copy has to be made in the function. This is memory-intensive.

A reference to the argument would prevent unnecessary copying.