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.

```
#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 << "ii: " << i << std::endl;
   std::cout << "ii: " << i << std::endl;
}</pre>
```

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 **NULL**. Therefore, it must always be initialized by having an existing variable assigned to it. The following lines would not work:

```
int& intRef;
int& intRef = NULL;
```

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>
                                                                                               G
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;</pre>
  std::cout << "b: " << b << std::endl;</pre>
 xchg(a, b);
  std::cout << std::endl;</pre>
  std::cout << "a: " << a << std::endl;
  std::cout << "b: " << b << std::endl;</pre>
```

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.

That is all about pointers. In the next chapter, we'll deal with **automatic type deduction** in C++.