Const Cast

This lesson highlights the key features of the const_cast operator.

Features

- `const_cast` allows us to remove or add the `const` or `volatile` property from a variable.
- `const_cast` is perhaps the most rarely used cast operator because it is undefined behavior to remove `const` or `volatile` from a variable if the variable was declared `const` or `volatile` in the first place.

Example

```
#include <iostream>
#include <typeinfo>

int main(){

  std::cout << std::endl;

  int i{2011};
  const int* constI = const_cast<const int*>(&i); // A const int pointer for an int
  int* nonConstI = const_cast<int*>(constI); // Casting to an int pointer
  *nonConstI = 9000;

  std::cout << "i: " << i << std::endl;

  std::cout << std::endl;

  std::cout << "typeid(constI).name(): " << typeid(constI).name() << std::endl;
  std::cout << "typeid(nonConstI).name(): " << typeid(nonConstI).name() << std::endl;

  std::cout << std::endl;

}
```

- In line 9, `const_cast` is being used to cast an integer reference to `const int` so that it can be assigned to the `constI` pointer.
- We can see the opposite happening in line 10 where `const_cast` is being used to remove the `const` property of the `constI` pointer.
- This sort of behavior creates a discrepancy in the code since two pointers of different types are pointing to the same variable. Hence, it's better to avoid such a practice.

In the next lesson, we'll discuss the `reinterpret_cast` operator.