Explicit Cast

In this lesson, we'll learn how to perform explicit casting in C++.

WE'LL COVER THE FOLLOWING

So far, we've seen how the compiler performs implicit casts during operations. However, C++ allows us to apply explicit casts as well.

There are 4 different named cast operators:

- `dynamic_cast`
- `static_cast`
- `const_cast`
- `reinterpret_cast`

The functionality of each operator will be discussed shortly.

Syntax

Explicit cast operators require the `type` that we want to cast our data in. They can be used in the following way:

```
double myDouble{5.5};
int i = static_cast<int>(myDouble);
```

C-casts

C-Casts are primitive cast operators that convert data from one type to another. They use the following format:

```
(type) value or expression
```

Here's an example:

```
double myDouble{5.5};
int i = (int)myDouble;
```

We should avoid using C-casts since they present a number of problems.

What is bad about the C-cast? We don't see which cast is actually used. If we perform a C-cast, a combination of casts will be applied, if necessary. Roughly speaking, a C-cast starts with a `static_cast`, continues with a `const_cast`, and finally performs a `reinterpret_cast`.

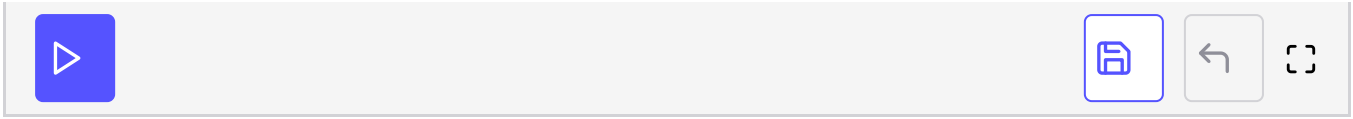
Let's see what happens if we screw up the type system:

```
#include <iostream>

int main(){

  double d = 2;
  auto p = (long*)&d;
  auto q = (long long*)&d;
  std::cout << d << ' ' << *p << ' ' << *q << '\n';

}
```



The output doesn't look too promising. A conversion like this could cause serious problems in a program. Hence, it is always better and safer to use named casts instead of C-casts.

Of course, you know how we will continue: *explicit is better than implicit*.

Let's explain each named cast operator in detail, starting with `dynamic_cast`.