## Type Information

This lesson describes techniques for obtaining the type of an entity.

#### WE'LL COVER THE FOLLOWING ^

- typeid
- type\_info
- Example
- Further information

## typeid #

We have seen numerous instances of the typeid operator in this course. It can be used to retrieve the type of a variable or object at *runtime*. Because of this, it works well with pointers.

To use typeid, we must include the <typeinfo> header. The operator returns a type\_info object that has various methods of its own.

Here's how typeid can be used:

```
Circle c(5.0);
const std::type_info& t = typeid(Circle);
const std::type_info& v = typeid(c);
```

Notice the & in the assignment operations. This specifies that this variable will be a reference to a type object. We must also make it const because each type has a single type\_info instance associated with it.

```
type_info #
```

A type\_info object stores information about a type. One useful feature is that it allows two types to be compared using comparison operators.

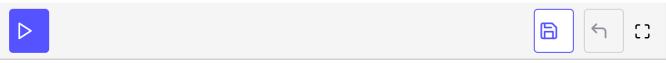
It can also tell us the name of the type through the name() method.

```
if (typeid(a) == typeid(b)){
   // a and b are of the same type
}
std::cout << typeid(a).name() << std::endl;</pre>
```

The name is implementation-defined and must be the same for each variable of the same type.

# Example #

```
#include <iostream>
                                                                                               G
#include <typeinfo>
int main(){
  std::cout << std::endl;</pre>
 // types
 if (typeid(int) == typeid(long long)){
    std::cout << "The types int and long long are the same" << std::endl;</pre>
  }
  else{
    std::cout << "The types int and long long are different" << std::endl;</pre>
  std::cout << "typeid(int).name(): " << typeid(int).name() << std::endl;</pre>
  std::cout << "typeid(long long).name(): " << typeid(long long).name() << std::endl;</pre>
  std::cout << std::endl;</pre>
  // variables
  int i{2011};
  int long long il{2011};
  std::cout << "typeid(i).name(): " << typeid(i).name() << std::endl;</pre>
  std::cout << "typeid(il).name(): " << typeid(il).name() << std::endl;</pre>
  if (typeid(i) == typeid(il)){
      std::cout << "The variables i and il are of the same type" << std::endl;</pre>
    else{
      std::cout << "The variables i and il are of different types" << std::endl;</pre>
  std::cout << std::endl;</pre>
```



The first part of the code runs typeid on the int and long long classes

- When compared in line 9, the compiler tells us that the two types are not the same.
- The names of the types can be obtained through the name() method, as done in lines 15 and 16.
- The same procedure is applied to the variables, i and il. Once again, their types are different.

### Further information #

• Type information

This brings us to the end of our discussion on casts in C++. We will now move on to **Unified Initialization** in the next chapter.