- Examples

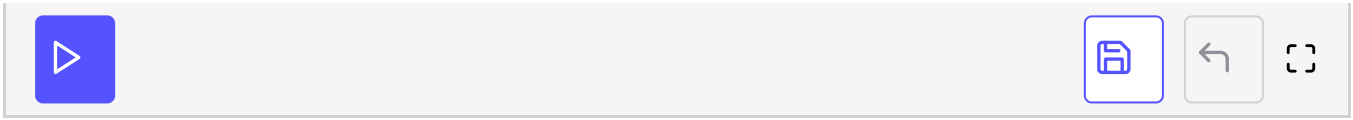
Let's have a look at the examples of policy and traits in this lesson.

WE'LL COVER THE FOLLOWING

Example 1: Templates Policy

```
// PolicytemplatesPolicy.cpp

#include <iostream>
#include <unordered_map>

struct MyInt{
    explicit MyInt(int v):val(v){}
    int val;
};

struct MyHash{
    std::size_t operator()(MyInt m) const {
        std::hash<int> hashVal;
        return hashVal(m.val);
    }
};

struct MyEqual{
    bool operator () (const MyInt& fir, const MyInt& sec) const {
        return fir.val == sec.val;
    }
};

std::ostream& operator << (std::ostream& strm, const MyInt& myIn){
    strm << "MyInt(" << myIn.val << ")";
    return strm;
}

int main(){

    std::cout << std::endl;
```

```
    typedef std::unordered_map<MyInt, int, MyHash, MyEqual> MyIntMap;


    std::cout << "MyIntMap: ";
    MyIntMap myMap{{MyInt(-2), -2}, {MyInt(-1), -1}, {MyInt(0), 0}, {MyInt(1), 1}};

    for(auto m : myMap) std::cout << '{' << m.first << ", " << m.second << "}";

    std::cout << "\n\n";

}
```



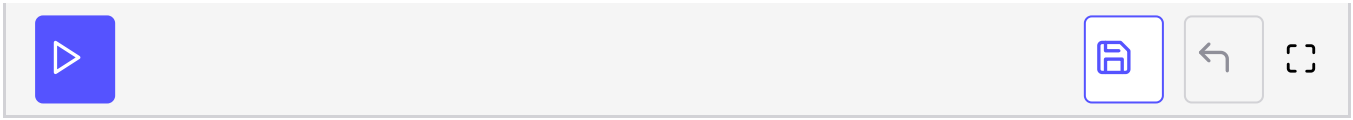
Explanation

The example uses the user-defined type `MyInt` as key for an `std:unordered_map`. To use `MyInt` as a key, `MyInt` must support the hash value and must be equal comparable. The classes `MyHash` (lines 11 – 16) and `MyEqual` (lines 18 – 22) provides the functionality for `MyInt` by delegating the job to the underlying `int`. The `typedef` in line 33 brings all together. In line 36, `MyInt` is used as a key in a `std::unordered_map`. The class `std::unoredered_map` in line 33 is an example of the policy class. The adaptable behavior is, in this case, the hash function and the equal function. Both policy parameters have default but can also be specified.

Example 2: Templates Traits

```
// TemplatesTraits.cpp

#include <iostream>
#include <type_traits>

using namespace std;

template <typename T>
void getPrimaryTypeCategory(){

  cout << boolalpha << endl;

  cout << "is_void<T>::value: " << is_void<T>::value << endl;
  cout << "is_integral<T>::value: " << is_integral<T>::value << endl;
  cout << "is_floating_point<T>::value: " << is_floating_point<T>::value << endl;
  cout << "is_array<T>::value: " << is_array<T>::value << endl;
  cout << "is_pointer<T>::value: " << is_pointer<T>::value << endl;
  cout << "is_reference<T>::value: " << is_reference<T>::value << endl;
  cout << "is_member_object_pointer<T>::value: " << is_member_object_pointer<T>::value << end
  cout << "is_member_function_pointer<T>::value: " << is_member_function_pointer<T>::value <<
  cout << "is_enum<T>::value: " << is_enum<T>::value << endl;
  cout << "is_union<T>::value: " << is_union<T>::value << endl;
```

```
  cout << "is_class<T>::value: " << is_class<T>::value << endl;
  cout << "is_function<T>::value: " << is_function<T>::value << endl;
  cout << "is_lvalue_reference<T>::value: " << is_lvalue_reference<T>::value << endl;
  cout << "is_rvalue_reference<T>::value: " << is_rvalue_reference<T>::value << endl;

  cout << endl;

}

int main(){
  getPrimaryTypeCategory<void>();
}
```



Explanation

In the example above, we have defined the function `getPrimaryTypeCategory` in line 9 which takes a type `T` and determines which type category `T` belongs to. This example uses all 14 primary type categories of the type-traits library.

In the next lesson, we'll solve a few exercises to understand more on policy and traits in idioms and patterns.