- Exercises

Let's solve a few exercises on policy and traits in this lesson.

WE'LL COVER THE FOLLOWING

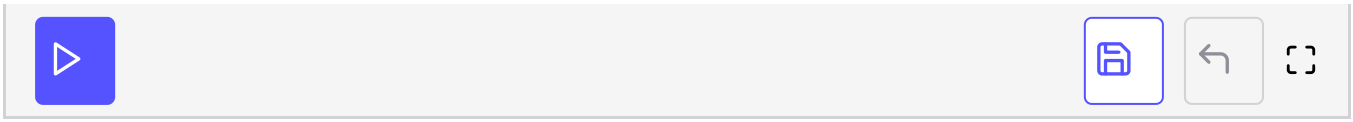
- Problem Statement 1
- Problem Statement 2
- Problem Statement 3

Problem Statement 1

Extend the example below in such a way such that the type `MyInt` can be used as a key to an `std::map`.

```
// 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;
}

// write your functions here
```

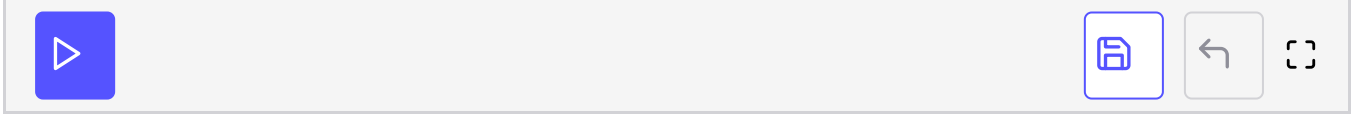
```
int main(){

// uncomment these line after implementing these functionalities

/*
    std::cout << std::endl;

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

    std::cout << "MyUnorderedMap: ";
    MyUnorderedMap 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 << std::endl;

    typedef std::map<MyInt, int, MySmaller> MyOrderedMap;

    std::cout << "MyOrderedMap: ";
    MyOrderedMap myMap2{{MyInt(-2), -2}, {MyInt(-1), -1}, {MyInt(0), 0}, {MyInt(1), 1}};

    for(auto m : myMap2) std::cout << '{' << m.first << ", " << m.second << "}";

    std::cout << "\n\n";
*/
}
```



Problem Statement 2

Define a `std::map<std::string, int>`, in which the keys are sorted in a decreasing way.

```
#include <iostream>
#include <map>

int main() {

  // your code goes here

}
```

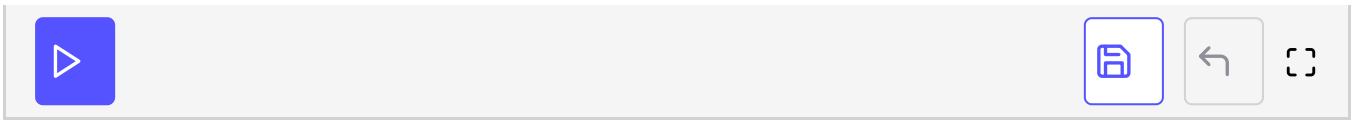


Problem Statement 3

The function template templateTraits.cpp shows for which primary type category each type belongs to. Each type is in exactly one primary type

category. Extend the program so that you get a type for each type category.

```
// 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(){
  // check for all types for calling type-traits fucntions

}
```



We'll look at the solutions of these exercises in the next lesson.