- Examples

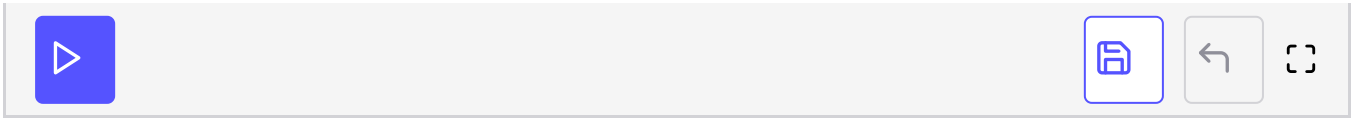
In this lesson, we will look at a few examples of CRTP.

WE'LL COVER THE FOLLOWING

Example 1

```
// crtpEquality.cpp

#include <iostream>
#include <string>

template<class Derived>
class Equality{};

template <class Derived>
bool operator == (Equality<Derived> const& op1, Equality<Derived> const & op2){
  Derived const& d1 = static_cast<Derived const&>(op1);
  Derived const& d2 = static_cast<Derived const&>(op2);
  return !(d1 < d2) && !(d2 < d1);
}

template <class Derived>
bool operator != (Equality<Derived> const& op1, Equality<Derived> const & op2){
  Derived const& d1 = static_cast<Derived const&>(op1);
  Derived const& d2 = static_cast<Derived const&>(op2);
  return !(d1 == d2);
}

struct Apple:public Equality<Apple>{
  Apple(int s): size{s}{};
  int size;
};

bool operator < (Apple const& a1, Apple const& a2){
  return a1.size < a2.size;
}
```

```
struct Man:public Equality<Man>{
  Man(const std::string& n): name{n}{}
  std::string name;
};

bool operator < (Man const& m1, Man const& m2){
  return m1.name < m2.name;
}


int main(){

  std::cout << std::boolalpha << std::endl;

  Apple apple1{5};
  Apple apple2{10};
  std::cout << "apple1 == apple2: " << (apple1 == apple2) << std::endl;

  Man man1{"grimm"};
  Man man2{"jaud"};
  std::cout << "man1 != man2: " << (man1 != man2) << std::endl;

  std::cout << std::endl;

}
```

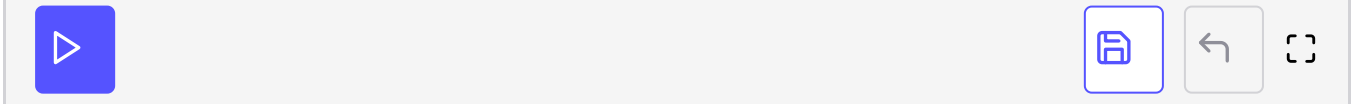


Explanation

- For the classes `Apple` and `Man`, we implemented the smaller operator (lines 28 and 37). We will only use the class `Man` for simplicity. The class `Man` is public derived (lines 32 - 35) from the class `Equality<Man>`.
- For classes of the kind `Equality<Derived>`, we implemented the equality (lines 9 - 14) and the inequality operator (lines 16 - 21). The inequality operator uses the equality operator (line 20).
- The equality operator relies on the fact that the smaller operator is implemented for `Derived` (line 13). The equality operator and inequality operator convert their operands: `Derived const&: Derived const& d1 = static_cast<Derived const&>(op1)`.
- Now, we can compare `Apple` and `Man` for equality and inequality in the main program.

Example 2

```
// templateCRTP.cpp


#include <iostream>

template <typename Derived>
struct Base{
  void interface(){
    static_cast<Derived*>(this)->implementation();
  }

  void implementation(){
    std::cout << "Implementation Base" << std::endl;
  }
};

struct Derived1: Base<Derived1>{
  void implementation(){
    std::cout << "Implementation Derived1" << std::endl;
  }
};

struct Derived2: Base<Derived2>{
  void implementation(){
    std::cout << "Implementation Derived2" << std::endl;
  }
};

struct Derived3: Base<Derived3>{};

template <typename T>
void execute(T& base){
    base.interface();
}


int main(){

  std::cout << std::endl;

  Derived1 d1;
  execute(d1);

  Derived2 d2;
  execute(d2);

  Derived3 d3;
  execute(d3);

  std::cout << std::endl;

}
```



Explanation

*In the function template `execute` (lines 30-33), we used static polymorphism.

In the function template `execute` (lines 30-33), we used static polymorphism.

- On each argument, we invoked the base method `base.interface`. The method `Base::interface` in lines 7 - 9 is the key point of the CRTP idiom. The methods dispatch to the implementation of the derived class: `static_cast<Derived*>(this)->implementation()`.This is possible since the method is instantiated when called.
- Now, the derived classes `Derived1`, `Derived2`, and `Derived3` are fully defined. Therefore, the method `Base::interface` can use the details of its derived classes. Especially interesting is the method `Base::implementation` (lines 11 - 13).
- This method plays the role of a default implementation of the static polymorphism for the class `Derived3` (line 28).

Now that we have completed the section on, **High Performance** in Embedded Programming with Modern C++, we will discuss how to work with **Reduced Resources** in embedded programming in the next lesson.