Functors

```
. Functors:
    .#1: Functor basics:

    .#2: Standard functors

    .#3: Functors with parameters

    .#4: Functors and lambdas
        . Showing that lambdas are implemented as functors behind the scenes.
```

Functors: The basics.

```cpp
// The functors
export class Encrypt {
public:
    char operator()(const char &param) {
        return static_cast<char>(param + 3);
    }
};

// The function pointers
export char encrypt(const char &param) {  // Callback function
    return static_cast<char>(param + 3);
}

export template <typename Modifier>
std::string &modify(std::string &str_param, Modifier modifier) {
    for (size_t i{}; i < str_param.size(); ++i) {
        str_param[i] = modifier(str_param[i]);  // Calling the callback
    }
    return str_param;
}
```

Functors: The basics.

```
//Usage
// Modifying through functors
std::string msg{"Hello world!"};
Encrypt encryptor;
Decrypt decryptor;

fmt::print("Original message: {}\n", msg);
modify(msg, encryptor);
fmt::print("Encrypted message: {}\n", msg);
modify(msg, decryptor);
fmt::print("Decrypted message: {}\n", msg);

// Modifying through function pointers
std::string msg2{"Hello world!"};
fmt::print("Original message: {}\n", msg2);
modify(msg2, encrypt);
fmt::print("Encrypted message: {}\n", msg2);
modify(msg2, decrypt);
fmt::print("Decrypted message: {}\n", msg2);
```

Functors: Built-in

```cpp
std::plus<int> adder;
std::minus<int> substracter;
std::greater<int> compare_greater;

fmt::println("10 + 7: {}", adder(10, 7));            // 17
fmt::println("10 - 7: {}", substracter(10, 7));       // 3
fmt::println(" 10 > 7: {}", compare_greater(10, 7));  // true

fmt::println("---------------------------------");

BoxContainer<std::string> quote;
quote.add("The");
quote.add("sky");
quote.add("is");
quote.add("blue");
quote.add("my");
quote.add("friend");

std::greater<std::string> string_comparator{};

// fmt::println("quote : {}", quote);
std::cout << "quote: " << quote << "\n";
//  Built in functor
fmt::println("greater string : {}", get_best(quote, string_comparator));
```

Functors with parameters

```cpp
// A functor can take parameters and internally
// store them as member variables
export template<typename T>
requires std::is_arithmetic_v<T>
class IsInRange {
public:
    IsInRange(T min, T max) : min_inclusive{ min }, max_inclusive{ max } {}
    bool operator()(T value) const {
        return ((value ⩾ min_inclusive) && (value ⩽ max_inclusive));
    }

private:
    T min_inclusive;
    T max_inclusive;
};

export template<typename T, typename RangePicker>
requires std::is_arithmetic_v<T>
T range_sum(const BoxContainer<T> &collection, RangePicker is_in_range) {
    T sum{};
    for (size_t i{}; i < collection.size(); ++i) {
        if (is_in_range(collection.get_item(i)))
            sum += collection.get_item(i);
    }
    return sum;
}
```

Functors with parameters

```
//Usage
BoxContainer<double> doubles;
doubles.add(10.1);
doubles.add(20.2);
doubles.add(30.3);
doubles.add(15);

std::cout << "doubles: " << doubles << "\n";
fmt::println("range_sum :{} ", range_sum(doubles, IsInRange<double>(10.0, 15.5)));
fmt::println("range_sum :{} ", range_sum(doubles, IsInRange<double>(10.0, 41.5)));
```

Lambdas use functors behind the scenes

```
int result = [](int x, int y) -> int { return x + y; }(7, 3);
```

```
class __lambda_5_22_7292729
{
  public:
  inline  int operator()(int x, int y) const
  {
    return x + y;
  }

};
```