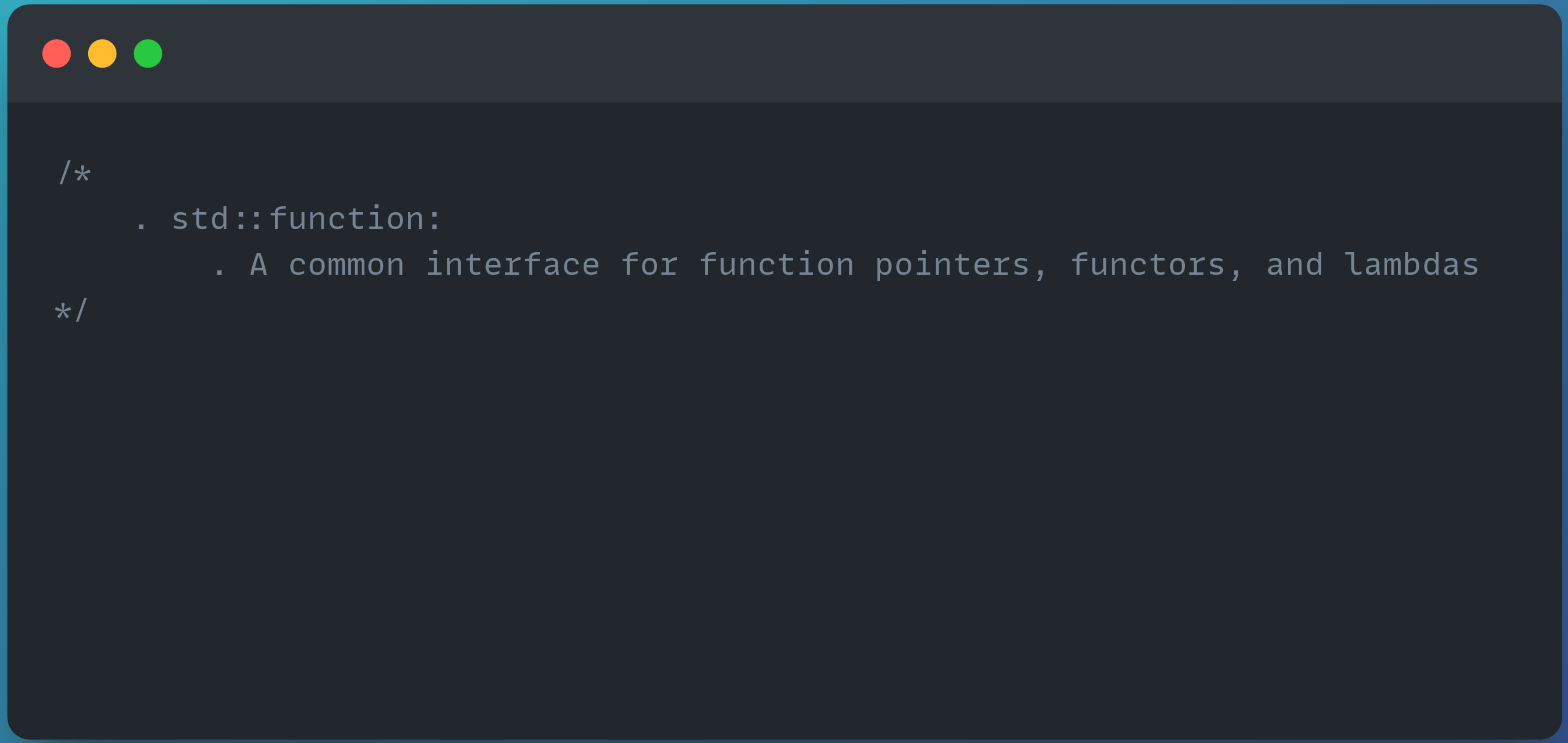
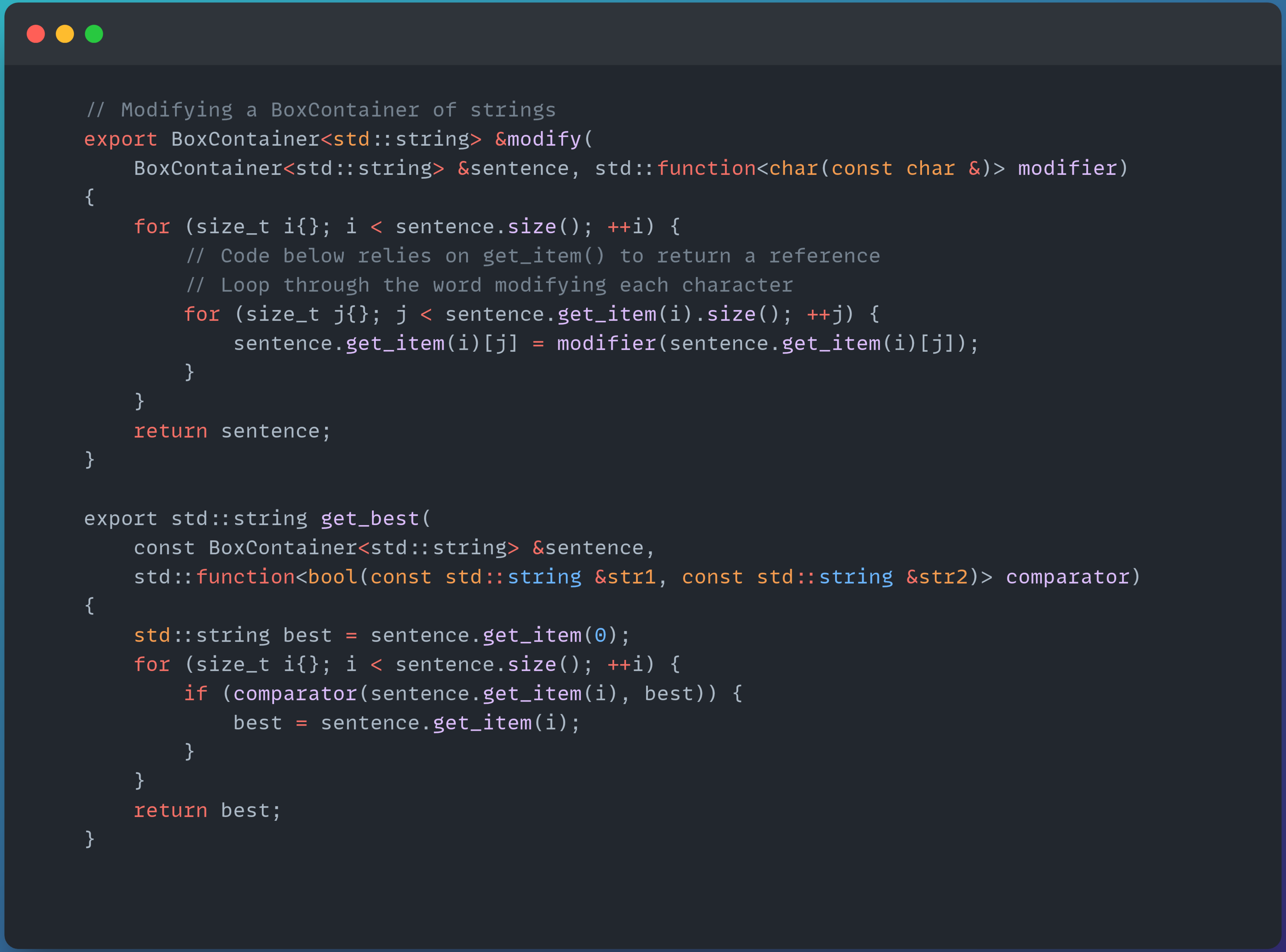
std::function



```
/*
    . std::function:
        . A common interface for function pointers, functors, and lambdas
*/
```

std::function



```cpp
// Modifying a BoxContainer of strings
export BoxContainer<std::string> &modify(
    BoxContainer<std::string> &sentence, std::function<char(const char &)> modifier)
{
    for (size_t i{}; i < sentence.size(); ++i) {
        // Code below relies on get_item() to return a reference
        // Loop through the word modifying each character
        for (size_t j{}; j < sentence.get_item(i).size(); ++j) {
            sentence.get_item(i)[j] = modifier(sentence.get_item(i)[j]);
        }
    }
    return sentence;
}

export std::string get_best(
    const BoxContainer<std::string> &sentence,
    std::function<bool(const std::string &str1, const std::string &str2)> comparator)
{
    std::string best = sentence.get_item(0);
    for (size_t i{}; i < sentence.size(); ++i) {
        if (comparator(sentence.get_item(i), best)) {
            best = sentence.get_item(i);
        }
    }
    return best;
}
```

std::function

```cpp
//Usage
std::function<char(const char &)> my_modifier;

// Function pointer
my_modifier = std_function::encrypt;
fmt::println("A encrypted becomes : {}", my_modifier('A'));// D

// Functor
std_function::Decrypt decrypt;
my_modifier = decrypt;
fmt::println("D decrypted becomes : {}", my_modifier('D'));


// Lambda function
my_modifier = [](const char &param) { return static_cast<char>(param + 3); };
fmt::println("A encrypted becomes : {}", my_modifier('A'));// D
```