Function pointers and callbacks

```
/*
    . Function pointers:
        .#1: Function pointers
          . syntax and basics
        .#2: Callback functions
        .#3: Function pointer aliases
*/
```

Function pointers: Syntax

```cpp
double (*f_ptr)(double, double) = &add;
double(*f_ptr) (double,double) = add;

double(*f_ptr) (double,double) {&add};
double(*f_ptr) (double,double){add};

auto f_ptr = &add;
auto f_ptr = add;

auto *f_ptr = &add;
auto *f_ptr = add;

// Things gone wroong
double (*f_ptr)(double, double) = nullptr;

fmt::println("add(10,30) : {}", f_ptr(10, 30));
```

Callback functions

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

export char decrypt(const char &param)
{// Callback function
    return static_cast<char>(param - 3);
}


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

```
//Usage
std::string msg{"Hello"};
modify(msg, encrypt);
fmt::println("outcome : {}", msg);
```

Callback functions

```
export std::string get_best(const BoxContainer<std::string> &sentence,
bool (*comparator)(const std::string &str1, const std::string &str2))
{

    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;
}


export bool larger_in_size(const std::string &str1, const std::string &str2) {
    return str1.size() > str2.size();
}

export bool greater_lexicographically(const std::string &str1, const std::string &str2) { return (str1 > str2); }
```

Function pointer type aliases

```cpp
//Function pointer type aliases.
// Using syntax
export using str_comparator = bool (*)(const std::string &str1, const std::string &str2);

//With typedefs
//export typedef bool(*str_comparator) (const std::string& str1, const std::string& str2);
```

```cpp
//Usage
// Using the type alias
str_comparator callback{ larger_in_size};

fmt::println("larger in size : {}",get_best(quote, callback));
```

Template type aliases

```
// Templated type alias
export template<typename T>
using compare_T = bool (*)(const T &, const T &); // Compare two "things" and return a bool

export template<typename T>
T get_best(const BoxContainer<T> &collection, compare_T<T> comparator)
{
    T best = collection.get_item(0);
    for (size_t i{}; i < collection.size(); ++i) {
        if (comparator(collection.get_item(i), best)) {
            best = collection.get_item(i);
        }
    }
    return best;
}

export bool larger_in_size(const std::string &str1, const std::string &str2) {return str1.size() > str2.size();}

export bool greater_lexicographically(const std::string &str1, const std::string &str2) { return (str1 > str2); }

export bool larger_int(const int &param1, const int &param2) { return param1 > param2; }

export template<typename T>
bool smaller(const T &param1, const T &param2){return param1 < param2;}
```