課題

課題1

```
#include <iostream>
#include <vector>
#include <list>

template<typename Container>
void print_elements(const Container& container) {
    for (const auto& element : container) {
        std::cout << element << " ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};
    std::list<char> lst = {'a', 'b', 'c', 'd', 'e'};

    print_elements(vec);
    print_elements(lst);

    return 0;
}
```

課題2

```
#include <iostream>
#include <vector>
#include <list>

template<typename T, typename Container = std::vector<T>>
class Stack {
public:
    void push(const T& value) {
        data.push_back(value);
    }

    void pop() {
        if (!is_empty()) {
            data.pop_back();
        }
    }

    T top() const {
        if (!is_empty()) {
            return data.back();
        } else {
            throw std::runtime_error("Stack is empty");
        }
    }

    bool is_empty() const {
        return data.empty();
    }

private:
    Container data;
};

int main() {
    Stack<int> int_vector_stack;
    int_vector_stack.push(1);
    int_vector_stack.push(2);
    std::cout << int_vector_stack.top() << std::endl; // Output: 2
    int_vector_stack.pop();
    std::cout << int_vector_stack.top() << std::endl; // Output: 1

    Stack<char, std::list<char>> char_list_stack;
    char_list_stack.push('a');
    char_list_stack.push('b');
    std::cout << char_list_stack.top() << std::endl; // Output: b
    char_list_stack.pop();
    std::cout << char_list_stack.top() << std::endl; // Output: a

    return 0;
}
```

課題3

```
#include <iostream>
#include <vector>
#include <list>
#include <deque>
#include <chrono>

constexpr int NUM_OPERATIONS = 100000;

template <typename T, typename Container>
class Stack {
public:
    void push(const T& value) {
        data.push_back(value);
    }

    void pop() {
        if (!is_empty()) {
            data.pop_back();
        }
    }

    T& top() {
        return data.back();
    }

    bool is_empty() const {
        return data.empty();
    }

private:
    Container data;
};

template<typename StackType>
void measure_performance() {
    StackType stack;
    auto start_time = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < NUM_OPERATIONS; ++i) {
        stack.push(i);
    }

    for (int i = 0; i < NUM_OPERATIONS; ++i) {
        stack.pop();
    }

    auto end_time = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>
(end_time - start_time).count();

    std::cout << "Time taken: " << duration << " microseconds" <<
std::endl;
}

int main() {
    std::cout << "Using std::vector:" << std::endl;
    measure_performance<Stack<int, std::vector<int>>>();

    std::cout << "Using std::list:" << std::endl;
    measure_performance<Stack<int, std::list<int>>>();

    std::cout << "Using std::deque:" << std::endl;
    measure_performance<Stack<int, std::deque<int>>>();

    return 0;
}
```

課題4

```
#include <iostream>
#include <string>
#include <map>
#include <fstream>

class PhoneBook {
public:
    void add_entry(const std::string& name, const std::string& number) {
        phone_book[name] = number;
    }

    void remove_entry(const std::string& name) {
        phone_book.erase(name);
    }

    std::string find_number(const std::string& name) const {
        auto it = phone_book.find(name);
        if (it != phone_book.end()) {
            return it->second;
        } else {
            return "Name not found";
        }
    }

    void display() const {
        for (const auto& entry : phone_book) {
            std::cout << entry.first << ": " << entry.second << std::endl;
        }
    }

    void load_from_file(const std::string& file_path) {
        std::ifstream input_file(file_path);

        if (!input_file) {
            std::cerr << "Error opening input file" << std::endl;
            return;
        }

        std::string name, number;
        while (input_file >> name >> number) {
            add_entry(name, number);
        }

        input_file.close();
    }

private:
    std::map<std::string, std::string> phone_book;
};

int main() {
    PhoneBook phone_book;
    phone_book.load_from_file("phone_numbers.txt");
    phone_book.display();

    std::cout << std::endl << "Adding a new entry (John, 555-1234):" <<
std::endl;
    phone_book.add_entry("John", "555-1234");
    phone_book.display();

    std::cout << std::endl << "Removing an entry (John):" << std::endl;
    phone_book.remove_entry("John");
    phone_book.display();

    std::cout << std::endl << "Searching for a number by name (Alice): ";
    std::string number = phone_book.find_number("Alice");
    std::cout << number << std::endl;

    return 0;
}
```