**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по заданию на получение доп. баллов**
**по дисциплине «Объектно-ориентированное программирование»**

Студент гр. 3381 _____ Сычев Н.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

**Цель работы.**

Цель задания — реализовать класс Mask, который позволяет работать с масками для обработки контейнеров с элементами, используя индексированные операции маскирования, трансформации и срезов.

**Задание.**

Реализовать класс Mask. У данного класса в качестве параметра шаблона задается размер маски, а в конструкторе задается маска, которая может содержать только 1 и 0.

Например:

Mask<4> mask = {1,1,0,1};

У класса Mask сделать метод size, который возвращает размер маски, и метод at, который возвращает значение по индексу в маске. Если при создании объекта количество элементов не соответствует размеру, то должна быть ошибка компиляции. Если при обращении в элементы вне диапазона, то должно бросаться исключение.

Также добавить в класс Mask метод slice, который может принять любой контейнер по ссылке (предполагается, что у контейнера есть метод at), и видоизменяющий контейнер (не создающий новый), удаляющий все элементы у которых по индексу маски стоит 0. Если размер маски больше размера контейнера, то маска обрабатывается до размера контейнера. Если размер маски меньше размера контейнера, то по достижении лимита маски, она начинает обрабатываться заново. Например, для массива чисел [1 2 3 4 5 6 7] и маски [1 0 0] должен получиться массив [1 4 7].

И добавить метод transform, который принимает любой контейнер (как метод slice) и функцию преобразования. Данный метод возвращает контейнер того же размера, который подобен оригинальному, но элементы попадающие под маску (значение маски 1) видоизменены функцией переданной в метод.

Реализовать метод slice_and_transfrom, который принимает те же аргументы, что и transform, и возвращает контейнер, содержащий только видоизмененные элементы функций преобразования (согласно маске)

**Выполнение работы.**

1) Конструктор класса Mask:

Конструктор принимает список инициализации (через std::initializer_list<int>) для создания маски. Внутри конструктора проверяется: Размер переданного списка должен совпадать с параметром шаблона N. Все элементы маски должны быть либо 0, либо 1. Если это не так, выбрасывается исключение.

2) Метод size():

Возвращает размер маски (размер массива mask_data), который равен значению параметра шаблона N.

3) Метод at(index):

Возвращает значение элемента маски по индексу. Если индекс выходит за пределы массива, выбрасывается исключение std::out_of_range.

4) Метод slice(Container& container):

Изменяет переданный контейнер по ссылке, удаляя все элементы, которые соответствуют нулю в маске. Если размер маски меньше размера контейнера, маска повторяется циклично. Контейнер изменяется на месте, элементы, которые не удовлетворяют маске, удаляются (используется метод resize для сужения контейнера).

5) Метод transform(Container& container, std::function<int(int)> func):

Применяет переданную функцию трансформации к элементам контейнера, которые соответствуют единице в маске. Возвращает новый контейнер того же размера, но с преобразованными элементами (элементы, соответствующие единице в маске, изменяются с использованием переданной функции).

6) Метод slice_and_transform(Container& container, std::function<int(int)> func): Сначала применяется метод slice, затем на результат применяется метод transform. Возвращает новый контейнер, содержащий только те элементы, которые были видоизменены согласно маске и преобразованной функции.

Разработанный программный код см. в приложении А.

**Вывод**

В рамках выполнения задания был реализован класс Mask, который выполняет несколько ключевых функций с использованием масок, содержащих значения 0 и 1.

## ПРИЛОЖЕНИЕ А
## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
# #include <iostream>

#include <stdexcept>

#include <initializer_list>

#include <vector>

#include <functional>



template<std::size_t N>

class Mask {

public:

    // Конструктор, который принимает список инициализации для маски

    Mask(std::initializer_list<int> list) {

        if (list.size() != N) {

            throw std::invalid_argument("Количество элементов не соответствует
```

```
размеру маски");
        }

        std::size_t i = 0;
        for (int val : list) {
            if (val != 0 && val != 1) {
                throw std::invalid_argument("Элементами маски могут быть только 0 и 1.");
            }
            mask_data[i++] = val;
        }
    }

    // Метод для получения размера маски
    std::size_t size() const {
        return N;
    }

    // Метод для доступа к элементам маски с проверкой диапазона
    int at(std::size_t index) const {
        if (index >= N) {
            throw std::out_of_range("Index out of range");
        }
        return mask_data[index];
    }
```

```
    // Метод slice, который изменяет контейнер
    template<typename Container>
    void slice(Container& container) const {
        std::size_t j = 0;
        for (std::size_t i = 0; i < container.size(); ++i) {
            if (at(i % N) == 1) {
                container[j++] = container.at(i);
            }
        }
        container.resize(j);
    }

    // Метод transform
    template<typename Container>
    Container transform(const Container& container,
std::function<int(int)> func) const {
        Container result = container;
        std::size_t mask_size = size();
        for (std::size_t i = 0; i < mask_size; ++i) {
            if (at(i) == 1) {
                result[i] = func(result[i]);
            }
        }
        return result;
```

```
    }

    // Метод slice_and_transform, который сначала применяет slice, затем transform
    template<typename Container>
    Container slice_and_transform(const Container& container, std::function<int(int)> func) const {
        Container sliced = container;
        slice(sliced);
        return transform(sliced, func);
    }

private:
    int mask_data[N];
};

int main() {
    try {
        // Маска 1: Применяется к вектору
        Mask<3> mask1 = {1, 0, 0};
        std::vector<int> data1 = {1, 2, 3, 4, 5, 6, 7};

        mask1.slice(data1);
        std::cout << "slize mask1{1, 0, 0} ";
        for (int val : data1) {
            std::cout << val << " ";
```

```
        }
        std::cout << std::endl;

        // Маска 2: Применяется к вектору с трансформацией
        Mask<4> mask2 = {0, 1, 1, 0};  // Маска длиной 4
        std::vector<int> data2 = {1, 2, 3, 4, 5, 6, 7};

        auto transformed1 = mask2.transform(data2, [](int x) { return x *
2; });
        std::cout << "transform mask2{0, 1, 1, 0}: ";
        for (int val : transformed1) {
            std::cout << val << " ";
        }
        std::cout << std::endl;

        // Пример 3: Применение slice_and_transform
        Mask<3> mask3 = {1, 0, 1};
        std::vector<int> data3 = {10, 20, 30, 40, 50};
        auto result3 = mask3.slice_and_transform(data3, [](int x) { return
x + 5; });
        std::cout << "slice_and_transform mask3{1, 0, 1}: ";
        for (int val : result3) {
            std::cout << val << " ";
        }
        std::cout << std::endl;
```

```
        // Пример 4: Маска с нулевыми значениями (не изменяет элементы)
        Mask<3> mask4 = {0, 0, 0};
        std::vector<int> data4 = {5, 10, 15, 20, 25};
        mask4.slice(data4);
        std::cout << "slice mask4{0, 0, 0} (ничего не изменяет): ";
        for (int val : data4) {
            std::cout << val << " ";
        }
        std::cout << std::endl;

        // Пример 5: Применение маски с повторяющимися значениями
        Mask<5> mask5 = {1, 1, 0, 0, 1};  // Маска длиной 5
        std::vector<int> data5 = {1, 2, 3, 4, 5, 6, 7};
        auto transformed2 = mask5.transform(data5, [](int x) { return x *
x; });
        std::cout << "transform mask5{1, 1, 0, 0, 1}: ";
        for (int val : transformed2) {
            std::cout << val << " ";
        }
        std::cout << std::endl;

    } catch (const std::exception& e) {
        std::cerr << "Exception: " << e.what() << std::endl;
    }
```

```
    return 0;
}
```