Отчет к четвертой лабораторной работе

Этот код реализует множество с тремя различными внутренними представлениями: массив, `std::vector` и `std::list`. В зависимости от размера множества, используется одна из этих реализаций.

Полный код программы

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

// Интерфейс реализации
class SetImpl {
public:
    virtual ~SetImpl() = default;
    virtual void add(int value) = 0;
    virtual void remove(int value) = 0;
    virtual bool contains(int value) const = 0;
    virtual size_t size() const = 0;
    virtual int get(size_t index) const = 0;
    virtual std::string name() const = 0;
};

// Реализация через массив
class ArraySet : public SetImpl {
    int data[100];
    size_t currentSize = 0;

public:
    void add(int value) override {
        if (currentSize < 100 && !contains(value)) {
            data[currentSize++] = value;
        }
    }

    void remove(int value) override {
        for (size_t i = 0; i < currentSize; ++i) {
            if (data[i] == value) {
                data[i] = data[--currentSize];
                return;
            }
        }
    }

    bool contains(int value) const override {
        for (size_t i = 0; i < currentSize; ++i) {
            if (data[i] == value) {
                return true;
```

```
            }
        }
        return false;
    }

    size_t size() const override {
        return currentSize;
    }

    int get(size_t index) const override {
        if (index < currentSize) {
            return data[index];
        }
        throw std::out_of_range("Index out of range");
    }

    std::string name() const override {
        return "Array Implementation";
    }
};

// Реализация через std::vector
class VectorSet : public SetImpl {
    std::vector<int> data;

public:
    void add(int value) override {
        if (!contains(value)) {
            data.push_back(value);
        }
    }

    void remove(int value) override {
        auto it = std::find(data.begin(), data.end(), value);
        if (it != data.end()) {
            data.erase(it);
        }
    }

    bool contains(int value) const override {
        return std::find(data.begin(), data.end(), value) != data.end();
    }

    size_t size() const override {
        return data.size();
    }

    int get(size_t index) const override {
        if (index < data.size()) {
            return data[index];
        }
        throw std::out_of_range("Index out of range");
    }
```

```
    std::string name() const override {
        return "Vector Implementation";
    }
};

// Реализация через std::list
class ListSet : public SetImpl {
    std::list<int> data;

public:
    void add(int value) override {
        if (!contains(value)) {
            data.push_back(value);
        }
    }

    void remove(int value) override {
        data.remove(value);
    }

    bool contains(int value) const override {
        return std::find(data.begin(), data.end(), value) != data.end();
    }

    size_t size() const override {
        return data.size();
    }

    int get(size_t index) const override {
        if (index >= data.size()) {
            throw std::out_of_range("Index out of range");
        }
        auto it = data.begin();
        std::advance(it, index);
        return *it;
    }

    std::string name() const override {
        return "List Implementation";
    }
};

class Set {
    SetImpl* impl;

    void switchImpl(int element) {
        size_t currentSize = impl->size();
        SetImpl* newImpl = nullptr;

        if (currentSize + element <= 100) {
            newImpl = new ArraySet();
        } else if (currentSize + element <= 1000) {
            newImpl = new VectorSet();
        } else {
```

```
            newImpl = new ListSet();
        }

        // Перенос данных
        for (size_t i = 0; i < currentSize; ++i) {
            newImpl->add(impl->get(i));
        }

        delete impl;
        impl = newImpl;
    }

public:
    Set() : impl(new ArraySet()) {}

    ~Set() {
        delete impl;
    }

    void add(int value) {
        impl->add(value);
        switchImpl(1);
    }

    void remove(int value) {
        impl->remove(value);
        switchImpl(-1);
    }

    bool contains(int value) const {
        return impl->contains(value);
    }

    size_t size() const {
        return impl->size();
    }

    std::string name() const {
        return impl->name();
    }
};

int main() {
    Set mySet;

    for (int i = 0; i < 150; ++i) {
        mySet.add(i);
    }

    std::cout << "Set contains 50: " << mySet.contains(50) << std::endl;
    std::cout << "Set size: " << mySet.size() << std::endl;
    std::cout << "Set implementation: " << mySet.name() << std::endl;

    mySet.remove(50);
```

```
    std::cout << "Set contains 50: " << mySet.contains(50) << std::endl;

    for (int i = 51; i < 120; ++i) {
        mySet.remove(i);
    }

    std::cout << "Set size: " << mySet.size() << std::endl;
    std::cout << "Set implementation: " << mySet.name() << std::endl;

    return 0;
}
```

Описание классов и методов

Интерфейс `SetImpl`

Этот интерфейс определяет основные методы для всех реализаций множества.

Методы:

- `virtual ~SetImpl() = default`: Виртуальный деструктор.
- `virtual void add(int value) = 0`: Добавляет значение в множество.
- `virtual void remove(int value) = 0`: Удаляет значение из множества.
- `virtual bool contains(int value) const = 0`: Проверяет, содержится ли значение в множестве.
- `virtual size_t size() const = 0`: Возвращает размер множества.
- `virtual int get(size_t index) const = 0`: Возвращает элемент по индексу.
- `virtual std::string name() const = 0`: Возвращает имя реализации.

Класс `ArraySet`

Реализует множество на основе массива фиксированного размера.

Методы:

- `void add(int value) override`: Добавляет значение в массив, если оно еще не содержится и массив не заполнен.
- `void remove(int value) override`: Удаляет значение из массива, заменяя его последним элементом.
- `bool contains(int value) const override`: Проверяет, содержится ли значение в массиве.
- `size_t size() const override`: Возвращает текущий размер массива.
- `int get(size_t index) const override`: Возвращает элемент по индексу, если индекс корректен.
- `std::string name() const override`: Возвращает строку "Array Implementation".

Класс `VectorSet`

Реализует множество на основе `std::vector`.

Методы:

- `void add(int value) override`: Добавляет значение в вектор, если оно еще не содержится.
- `void remove(int value) override`: Удаляет значение из вектора.
- `bool contains(int value) const override`: Проверяет, содержится ли значение в векторе.
- `size_t size() const override`: Возвращает текущий размер вектора.
- `int get(size_t index) const override`: Возвращает элемент по индексу, если индекс корректен.
- `std::string name() const override`: Возвращает строку "Vector Implementation".

Класс `ListSet`

Реализует множество на основе `std::list`.

Методы:

- `void add(int value) override`: Добавляет значение в список, если оно еще не содержится.
- `void remove(int value) override`: Удаляет значение из списка.
- `bool contains(int value) const override`: Проверяет, содержится ли значение в списке.
- `size_t size() const override`: Возвращает текущий размер списка.
- `int get(size_t index) const override`: Возвращает элемент по индексу, если индекс корректен.
- `std::string name() const override`: Возвращает строку "List Implementation".

Класс `Set`

Класс для работы с множеством, который автоматически переключается между различными реализациями в зависимости от размера множества.

Методы:

- `Set()`: Конструктор, инициализирующий множество реализацией на основе массива.
- `~Set()`: Деструктор, освобождающий текущую реализацию множества.
- `void add(int value)`: Добавляет значение в множество и при необходимости переключает реализацию.
- `void remove(int value)`: Удаляет значение из множества и при необходимости переключает реализацию.
- `bool contains(int value) const`: Проверяет, содержится ли значение в множестве.
- `size_t size() const`: Воз

вращает текущий размер множества.

- `std::string name() const`: Возвращает имя текущей реализации множества.

Пример использования:

```
int main() {
    Set mySet;

    for (int i = 0; i < 150; ++i) {
        mySet.add(i);
    }

    std::cout << "Set contains 50: " << mySet.contains(50) << std::endl;
    std::cout << "Set size: " << mySet.size() << std::endl;
    std::cout << "Set implementation: " << mySet.name() << std::endl;

    mySet.remove(50);
    std::cout << "Set contains 50: " << mySet.contains(50) << std::endl;

    for (int i = 51; i < 120; ++i) {
        mySet.remove(i);
    }

    std::cout << "Set size: " << mySet.size() << std::endl;
    std::cout << "Set implementation: " << mySet.name() << std::endl;

    return 0;
}
```

Заключение

В данном отчете представлен код для реализации множества с тремя различными внутренними представлениями: массив, `std::vector` и `std::list`. В зависимости от размера множества, используется одна из этих реализаций. Были объяснены классы и методы, их параметры и функциональность. Программа автоматически переключает реализацию множества при добавлении и удалении элементов, обеспечивая оптимальную производительность в зависимости от размера множества.