Оглавление

[**Техническое задание** 3](#_Toc74511825)

[**Описание решаемой задачи** 3](#_Toc74511826)

[**Описание алгоритма** 3](#_Toc74511827)

[**Тесты** 19](#_Toc74511828)

# **Техническое задание**

Исследование паттерна State

# **Описание решаемой задачи**

Написание программы, которая симулирует секвентальную коробку передач.

Код был написан на языке C++ и при помощи паттерна **State**.

# **Описание алгоритма**

\*Листинг программы\*

#include <string>

#include <iostream>

class State;

class StateContext;

class FirstGear;

class SecondGear;

class ThirdGear;

class FourthGear;

class FithGear;

class SixthGear;

class GearN;

class GearR;

class State

{

std::string name;

public:

State(const std::string& name)

: name(name) {}

std::string GetName()

{

return name;

}

virtual void MaxRPM(StateContext\*) = 0;

virtual void MinRPM(StateContext\*) = 0;

};

class StateContext

{

private:

State\* state;

public:

StateContext(State\* state)

: state(state) {}

void MaxRPM()

{

std::cout << "Gearing up " << state->GetName() << "..." << std::endl;

state->MaxRPM(this);

}

void MinRPM()

{

std::cout << "Gearing down " << state->GetName() << "..." << std::endl;

state->MinRPM(this);

}

void SetState(State\* s)

{

std::cout << "Changing state from " << state->GetName()

<< " to " << s->GetName() << "..." << std::endl;

delete state;

state = s;

}

State\* GetState()

{

return state;

}

~StateContext()

{

delete state;

}

};

class FirstGear : public State

{

public:

FirstGear() : State("Gear 1") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class SecondGear : public State

{

public:

SecondGear() : State("Gear 2") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class ThirdGear : public State

{

public:

ThirdGear() : State("Gear 3") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class FourthGear : public State

{

public:

FourthGear() : State("Gear 4") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class FithGear : public State

{

public:

FithGear() : State("Gear 5") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class SixthGear : public State

{

public:

SixthGear() : State("Gear 6") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class GearN : public State

{

public:

GearN() : State("Neutral") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class GearR : public State

{

public:

GearR() : State("Reverse") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

void GearN::MaxRPM(StateContext\* state)

{

state->SetState(new FirstGear());

}

void GearN::MinRPM(StateContext\* state)

{

state->SetState(new GearR());

}

void FirstGear::MaxRPM(StateContext\* state)

{

state->SetState(new SecondGear());

}

void FirstGear::MinRPM(StateContext\* state)

{

state->SetState(new GearN());

}

void SecondGear::MaxRPM(StateContext\* state)

{

state->SetState(new ThirdGear());

}

void SecondGear::MinRPM(StateContext\* state)

{

state->SetState(new FirstGear());

}

void ThirdGear::MaxRPM(StateContext\* state)

{

state->SetState(new FourthGear());

}

void ThirdGear::MinRPM(StateContext\* state)

{

state->SetState(new SecondGear());

}

void FourthGear::MaxRPM(StateContext\* state)

{

state->SetState(new FithGear());

}

void FourthGear::MinRPM(StateContext\* state)

{

state->SetState(new ThirdGear());

}

void FithGear::MaxRPM(StateContext\* state)

{

state->SetState(new SixthGear());

}

void FithGear::MinRPM(StateContext\* state)

{

state->SetState(new FourthGear());

}

void SixthGear::MaxRPM(StateContext\* state)

{

std::cout << "Nothing happens" << std::endl;

}

void SixthGear::MinRPM(StateContext\* state)

{

state->SetState(new FithGear());

}

void GearR::MinRPM(StateContext\* state)

{

std::cout << "Nothing happens" << std::endl;

}

void GearR::MaxRPM(StateContext\* state)

{

state->SetState(new GearN());

}

int main()

{

StateContext\* sc = new StateContext(new GearN());

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MaxRPM();

delete sc;

return 0;

}

1) В данной программе использовались библиотеки string, iostream:

#include <string>

#include <iostream>

2) Создаём по классу для каждого сосотяния:

class State;

class StateContext;

class FirstGear;

class SecondGear;

class ThirdGear;

class FourthGear;

class FithGear;

class SixthGear;

class GearN;

class GearR;

3) Класс *State-*класс, в котором находятся подклассы, и функция для считывания имени состояния.

class State

{

std::string name;

public:

State(const std::string& name)

: name(name) {}

std::string GetName()

{

return name;

}

virtual void MaxRPM(StateContext\*) = 0;

virtual void MinRPM(StateContext\*) = 0;

};

4) Класс *StateContext -* [интерфейс](https://ru.wikipedia.org/wiki/Интерфейс), который должен реализовать каждое из конкретных состояний.

class StateContext

{

private:

State\* state;

public:

StateContext(State\* state)

: state(state) {}

void MaxRPM()

{

std::cout << "Gearing up " << state->GetName() << "..." << std::endl;

state->MaxRPM(this);

}

void MinRPM()

{

std::cout << "Gearing down " << state->GetName() << "..." << std::endl;

state->MinRPM(this);

}

void SetState(State\* s)

{

std::cout << "Changing state from " << state->GetName()

<< " to " << s->GetName() << "..." << std::endl;

delete state;

state = s;

}

State\* GetState()

{

return state;

}

~StateContext()

{

delete state;

}

};

5) Блок классов FirstGear…GearR- является блоком, в котором для каждого состояния определяется что будет выводится в терминале при переходе в одно из этих состояний и действия, которые с ними можно производить:

class FirstGear : public State

{

public:

FirstGear() : State("Gear 1") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class SecondGear : public State

{

public:

SecondGear() : State("Gear 2") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class ThirdGear : public State

{

public:

ThirdGear() : State("Gear 3") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class FourthGear : public State

{

public:

FourthGear() : State("Gear 4") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class FithGear : public State

{

public:

FithGear() : State("Gear 5") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class SixthGear : public State

{

public:

SixthGear() : State("Gear 6") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class GearN : public State

{

public:

GearN() : State("Neutral") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

class GearR : public State

{

public:

GearR() : State("Reverse") {}

virtual void MaxRPM(StateContext\* state);

virtual void MinRPM(StateContext\* state);

};

6) Блок, в котором для каждого класса содержится информация о том, в какое состояние будет осуществлён переход при определённых действиях

void GearN::MaxRPM(StateContext\* state)

{

state->SetState(new FirstGear());

}

void GearN::MinRPM(StateContext\* state)

{

state->SetState(new GearR());

}

void FirstGear::MaxRPM(StateContext\* state)

{

state->SetState(new SecondGear());

}

void FirstGear::MinRPM(StateContext\* state)

{

state->SetState(new GearN());

}

void SecondGear::MaxRPM(StateContext\* state)

{

state->SetState(new ThirdGear());

}

void SecondGear::MinRPM(StateContext\* state)

{

state->SetState(new FirstGear());

}

void ThirdGear::MaxRPM(StateContext\* state)

{

state->SetState(new FourthGear());

}

void ThirdGear::MinRPM(StateContext\* state)

{

state->SetState(new SecondGear());

}

void FourthGear::MaxRPM(StateContext\* state)

{

state->SetState(new FithGear());

}

void FourthGear::MinRPM(StateContext\* state)

{

state->SetState(new ThirdGear());

}

void FithGear::MaxRPM(StateContext\* state)

{

state->SetState(new SixthGear());

}

void FithGear::MinRPM(StateContext\* state)

{

state->SetState(new FourthGear());

}

void SixthGear::MaxRPM(StateContext\* state)

{

std::cout << "Nothing happens" << std::endl;

}

void SixthGear::MinRPM(StateContext\* state)

{

state->SetState(new FithGear());

}

void GearR::MinRPM(StateContext\* state)

{

std::cout << "Nothing happens" << std::endl;

}

void GearR::MaxRPM(StateContext\* state)

{

state->SetState(new GearN());

}

7) Функция main, в которой задаётся начальное состояние и определяется цепочка действий, подающихся поочерёдно.

int main()

{

StateContext\* sc = new StateContext(new GearN());

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MaxRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MinRPM();

sc->MaxRPM();

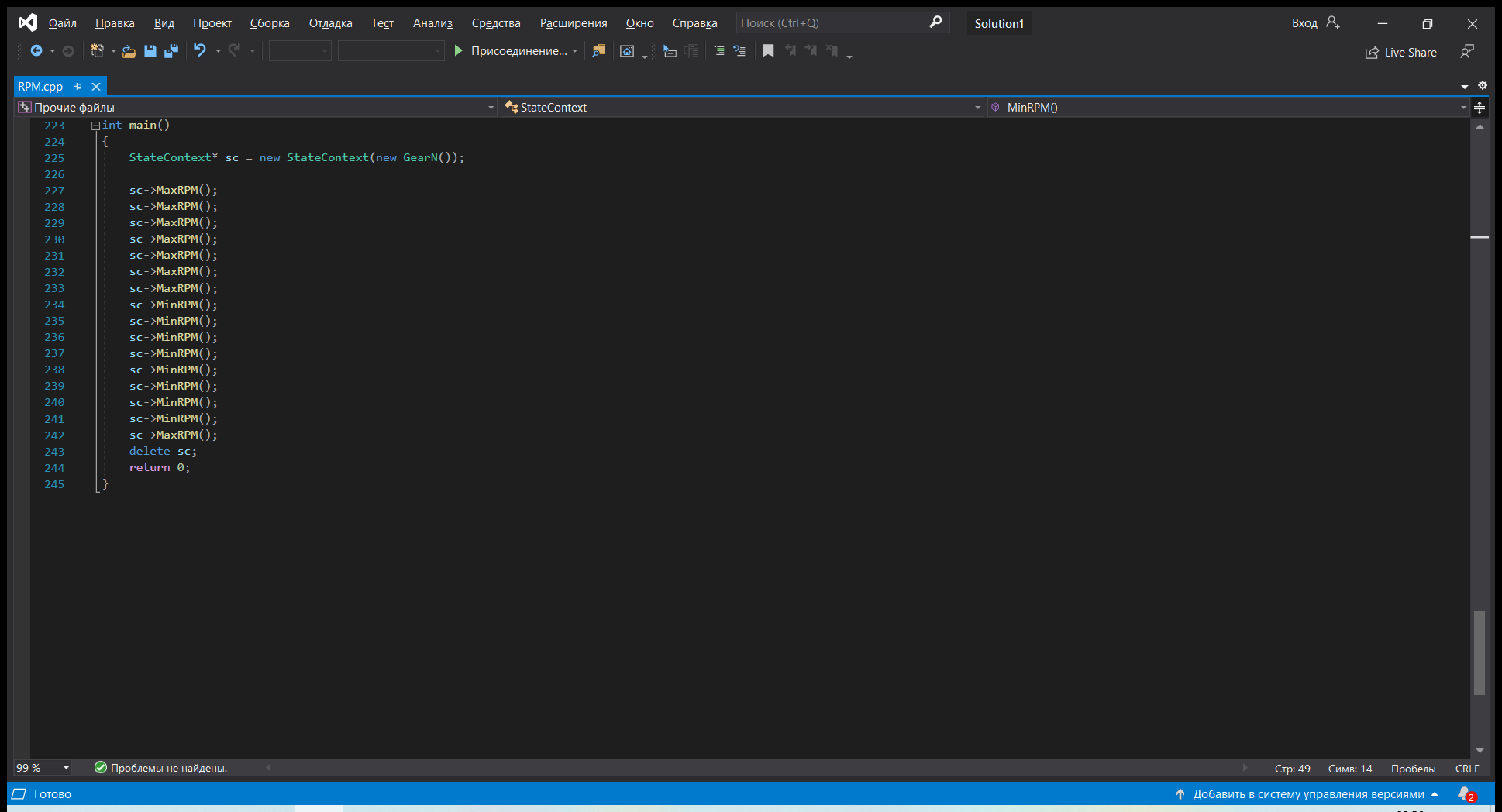
delete sc;

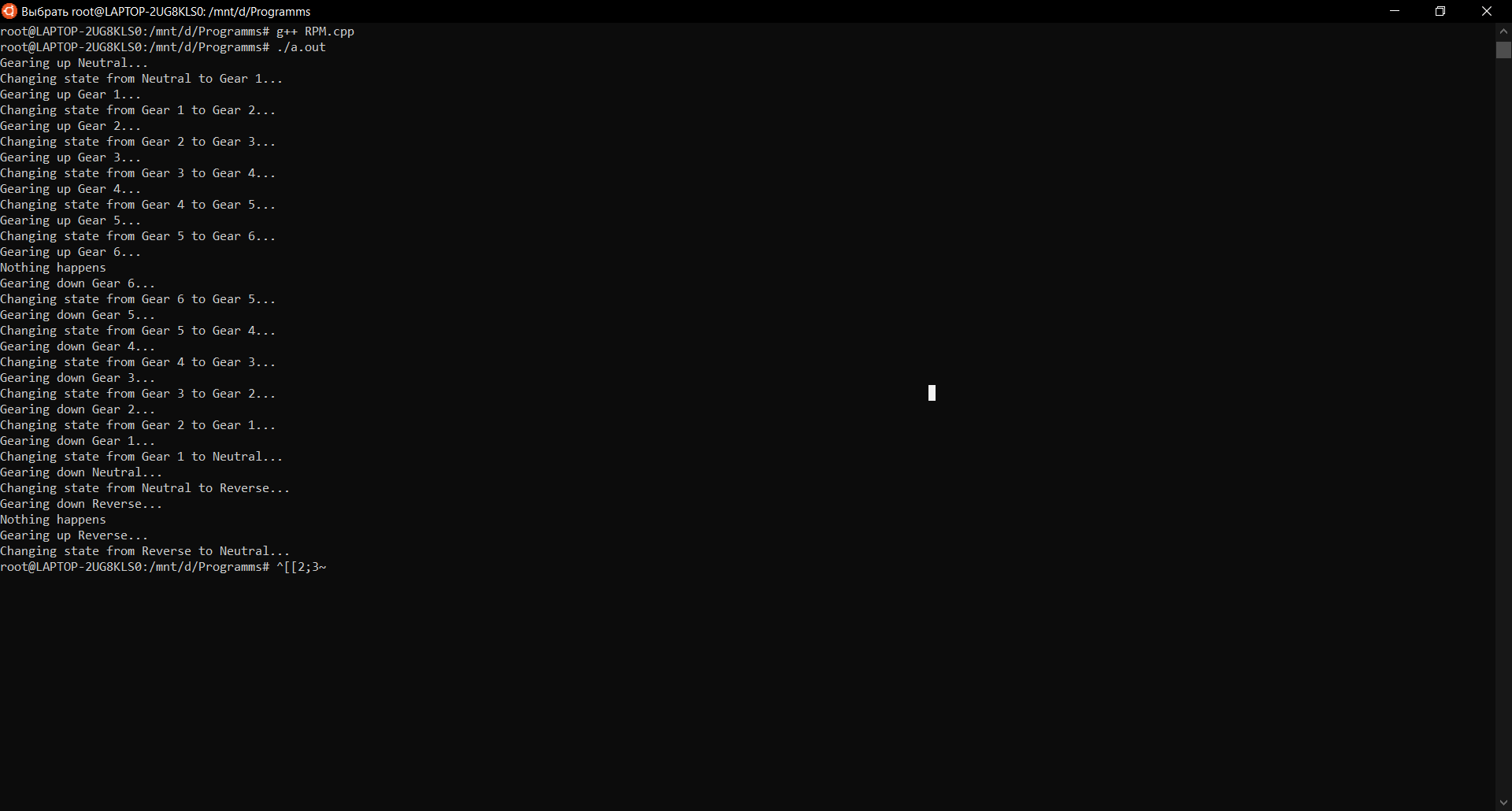
return 0;

}

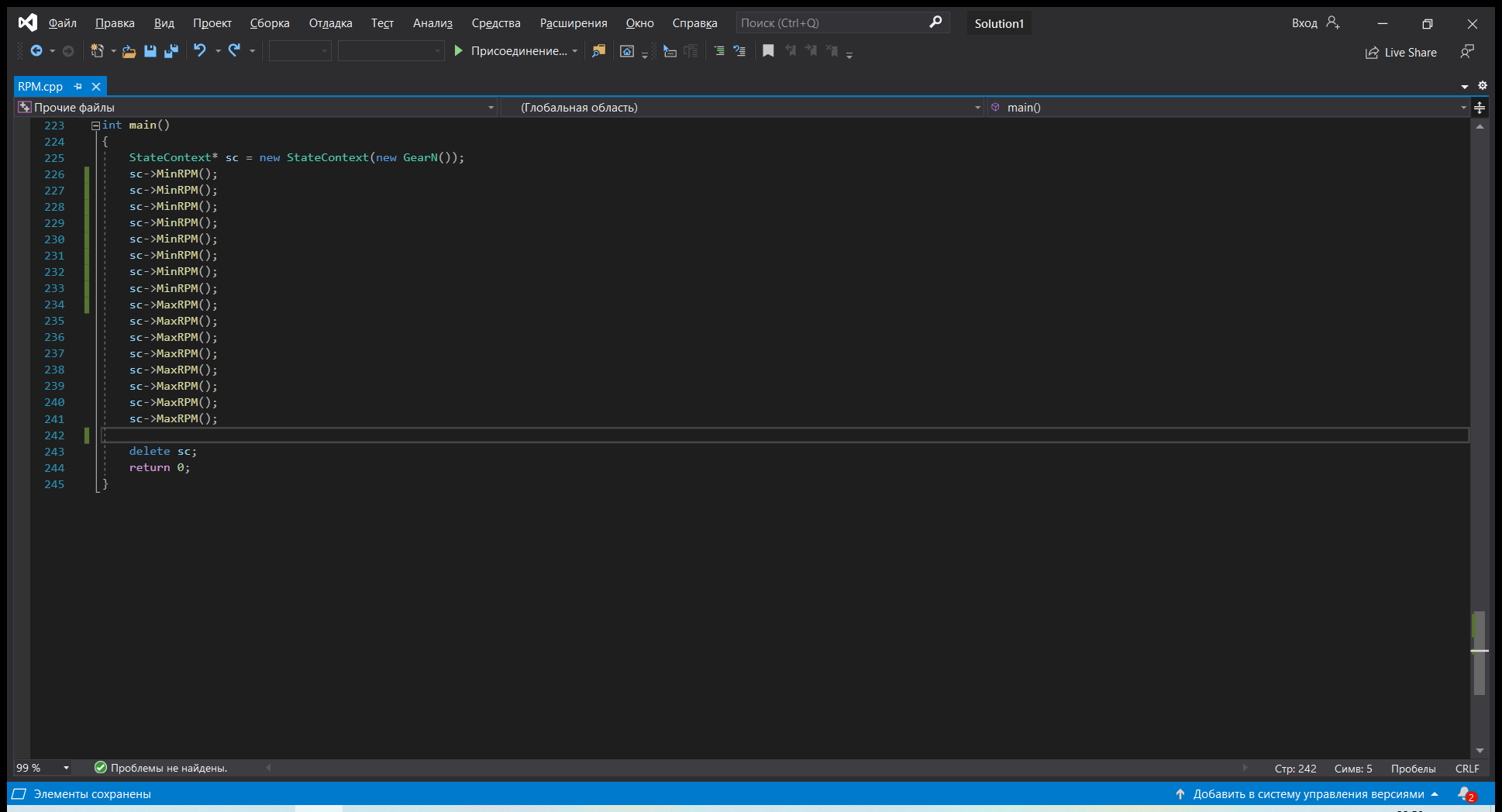
# **Тесты**

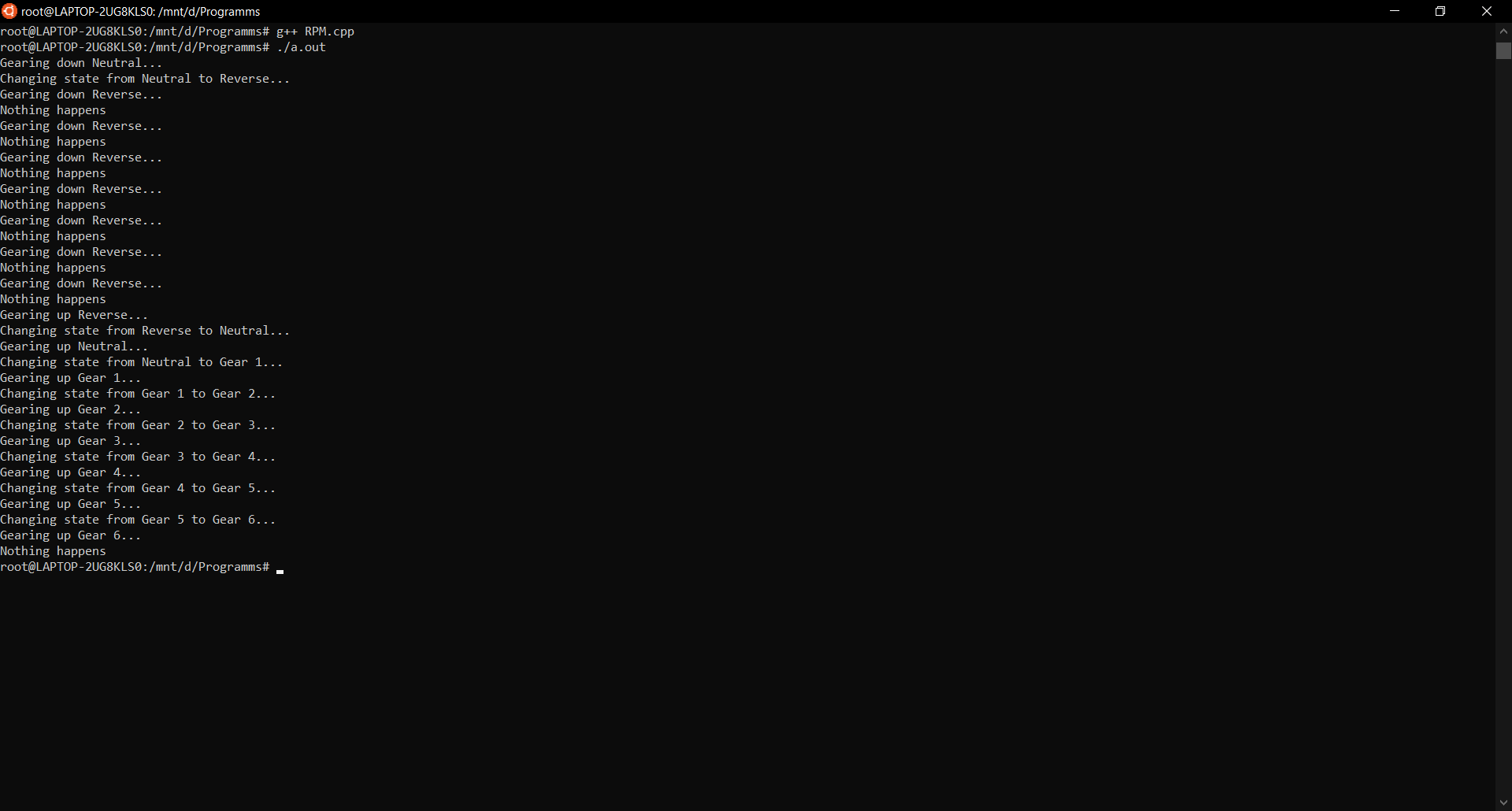
При данной цепочке действий осуществляются следующие переходы состояний.





Попробуем поменять цепочку действий.





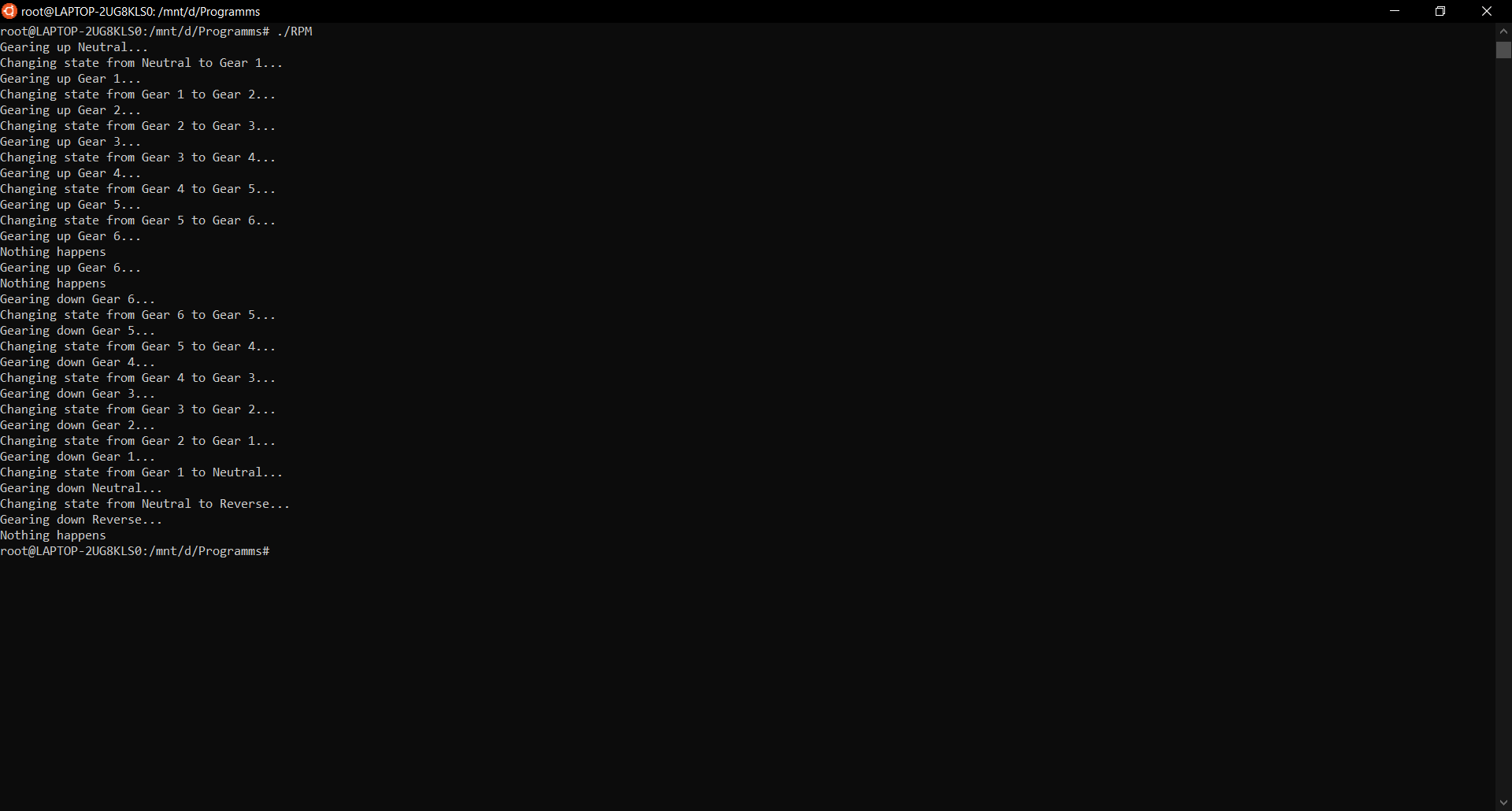
Как мы видим результат изменился.

Теперь проведём тестирование покрытия кода. Проводим мы его с помощью gcov.

1. Сначала откомпилируем программу и поместим её в исполняемый файл: g++ --coverage RPM.cpp -o RPM
2. Затем вызовем файл, который мы создали:

./RPM

Произойдёт отработка программы.



1. Вводим в терминал команду lcov с несколькими опциями:

lcov -t "RPM" -o RPM.info -c -d .

Пояснения по поводу опций:

* -t <имя> устанавливает имя отчёта, при измерении покрытия кода тестами можно указать имя теста или набора тестов
* -o <имя> устанавливает имя выходного файла с промежуточной информацией
* -c указывает, что lcov должен использовать существующие данные о coverage
* -d <путь> устанавливает каталог, в котором надо искать данные о coverage, и мы передаём текущий каталог “.”

1. Теперь можно сгенерировать отчёт в виде HTML-страницы с помощью утилиты genhtml, входящей в состав пакета программ lcov:

genhtml -o report RPM.info

5) Теперь вы можете перейти в каталог report, открыть файл index.html в браузере и посмотреть отчёт. Вы увидите страницу, на которой показан статичный отчёт о покрытии различных каталогов, и можно перейти по ссылкам для просмотра отдельных каталогов:

