МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Наследование в С++»**

*Вариант № 10*

Выполнил:

Студент группы

КТбо2-8

Макаров С. В.

Проверил:

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# **1 ФОРМУЛИРОВКА ЗАДАНИЯ**

Согласно варианту задания, описать класс Element (элемент логической схемы) с двумя входами и одним выходом, и полем, хранящим название элемента. Определить функцию, которая преобразует входные двоичные значения в выходное. На его основе реализовать классы AND и OR - двоичные вентили, которые реализуют логическое умножение и сложение соответственно. В дополнительном классе Sсheme создать массив элементов (до 10) и обеспечить подачу двоичных сигналов на их входы с выводом выходных значений. Входные сигналы хранятся в файле.

Варианты предполагают создание иерархии классов с виртуальными функциями. Если есть необходимость, то в производном классе можно объявлять дополнительные компоненты. Проектировать иерархию надо так, чтобы гипотетическое добавление нового производного класса (типа) происходило бы без существенного изменения уже существующих классов, т.е. процент повторного использования кода должно быть большим. Безусловно, виртуальные функции должны вызываться правильным образом. По-прежнему запрещается использование контейнеров STL.

# **2 СПЕЦИФИКАЦИЯ КЛАССОВ**

class LogicalElement

{

public:

LogicalElement() = delete;

LogicalElement(const std::string& \_name);

virtual bool Execute(bool left, bool right) const = 0;

virtual LogicalElement\* Copy() const = 0;

std::string GetName() const;

protected:

std::string name;

};

class And : public LogicalElement

{

public:

And();

virtual bool Execute(bool left, bool right) const override;

virtual LogicalElement\* Copy() const override;

};

class Or : public LogicalElement

{

public:

Or();

virtual bool Execute(bool left, bool right) const override;

virtual LogicalElement\* Copy() const override;

};

class Xor : public LogicalElement

{

public:

Xor();

virtual bool Execute(bool left, bool right) const override;

virtual LogicalElement\* Copy() const override;

};

class Scheme

{

friend class SchemeFileBuilder;

public:

Scheme();

Scheme(const Scheme& other);

~Scheme();

Scheme& operator=(const Scheme& other);

BinaryData Execute(const BinaryData& data) const;

int CountOfElements() const;

private:

void \_DeleteElements();

void \_CopyElements(int \_size, LogicalElement\*\* \_elements);

LogicalElement\*\* elements;

int size;

};

class SchemeFileBuilder

{

public:

SchemeFileBuilder() = delete;

SchemeFileBuilder(const char\* filename, ElementPattern \_pattern);

~SchemeFileBuilder();

Scheme Build();

bool IsOkay();

private:

std::fstream file;

ElementPattern pattern;

void \_ValidateSize(int size) const;

LogicalElement\*\* \_ReadSchemeFromFile(int size);

void \_ToUniqueFormat(std::string& str) const;

bool \_IsArrayOkay(int size, LogicalElement\*\* \_array) const;

};

class ElementPattern

{

public:

ElementPattern();

ElementPattern(int \_size, LogicalElement\*\* pattern);

ElementPattern(const ElementPattern& other);

~ElementPattern();

ElementPattern& operator=(const ElementPattern& other);

LogicalElement\* FindByName(const std::string& name) const;

private:

void \_DeleteElements();

void \_CopyElements(int \_size, LogicalElement\*\* \_elements);

int size;

LogicalElement\*\* table;

};

class BinaryData

{

public:

BinaryData();

BinaryData(bool\* \_buffer, int \_size);

BinaryData(const BinaryData& other);

~BinaryData();

BinaryData& operator=(const BinaryData& other);

bool\* GetBuffer() const;

int GetSize() const;

bool operator[](int index) const;

private:

void \_CopyData(int \_size, bool\* \_buf);

void \_DeleteData();

bool\* buffer;

int size;

};

class BinaryInputParser

{

public:

static bool ValidateString(const std::string& str);

static BinaryData Parse(const std::string& str, int size);

private:

static bool \_ConvertToBool(char letter);

};

class ConsoleInteractor

{

public:

void Start() const;

private:

ElementPattern \_GetDefaultPapptern() const;

};

# **3 ИСПОЛЬЗУЕМЫЕ МАТЕМАТИЧЕСКИЕ ЗАВИСИМОСТИ И АЛГОРИТМЫ**

В данной работе использовались логические формулы: AND, OR, XOR.

# **4 ДИАГРАММА КЛАССОВ**

Uml диаграмма классов изображена на рисунке 1.

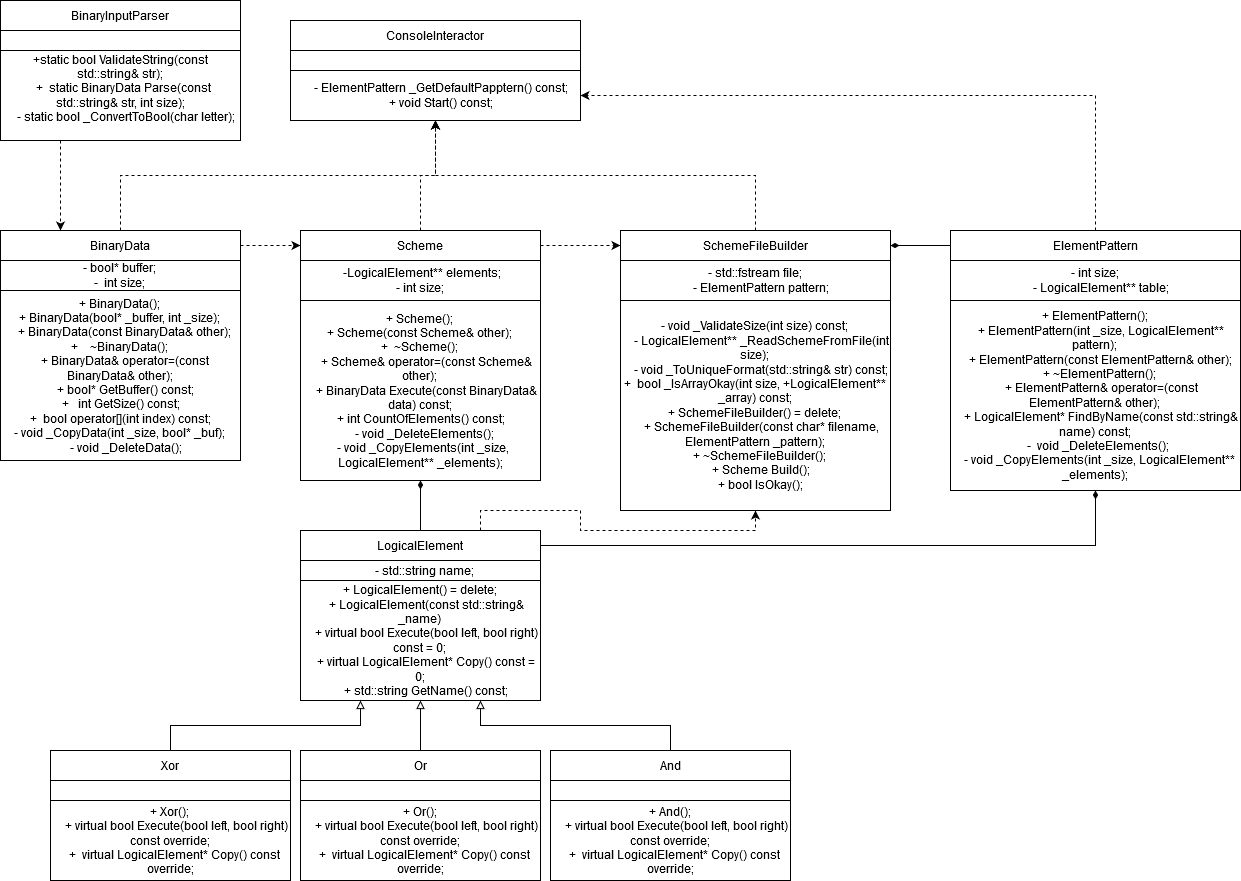


Рисунок 1 – Диаграмма классов

**ЛИСТИНГ ПРОГРАММЫ**

class LogicalElement

{

public:

LogicalElement() = delete;

LogicalElement(const std::string& \_name);

virtual bool Execute(bool left, bool right) const = 0;

virtual LogicalElement\* Copy() const = 0;

std::string GetName() const;

protected:

std::string name;

};

LogicalElement::LogicalElement(const std::string& \_name) : name(\_name)

{}

std::string LogicalElement::GetName() const

{

return name;

}

class And : public LogicalElement

{

public:

And();

virtual bool Execute(bool left, bool right) const override;

virtual LogicalElement\* Copy() const override;

};

class Or : public LogicalElement

{

public:

Or();

virtual bool Execute(bool left, bool right) const override;

virtual LogicalElement\* Copy() const override;

};

class Xor : public LogicalElement

{

public:

Xor();

virtual bool Execute(bool left, bool right) const override;

virtual LogicalElement\* Copy() const override;

};

And::And() : LogicalElement("AND")

{}

bool And::Execute(bool left, bool right) const

{

return left & right;

}

LogicalElement\* And::Copy() const

{

return new And();

}

Or::Or() : LogicalElement("OR")

{}

bool Or::Execute(bool left, bool right) const

{

return left | right;

}

LogicalElement\* Or::Copy() const

{

return new Or();

}

Xor::Xor() : LogicalElement("XOR")

{}

bool Xor::Execute(bool left, bool right) const

{

return left xor right;

}

LogicalElement\* Xor::Copy() const

{

return new Xor();

}

class Scheme

{

friend class SchemeFileBuilder;

public:

Scheme();

Scheme(const Scheme& other);

~Scheme();

Scheme& operator=(const Scheme& other);

BinaryData Execute(const BinaryData& data) const;

int CountOfElements() const;

private:

void \_DeleteElements();

void \_CopyElements(int \_size, LogicalElement\*\* \_elements);

LogicalElement\*\* elements;

int size;

};

Scheme::Scheme() : elements(nullptr), size(0)

{}

Scheme::Scheme(const Scheme& other)

: Scheme()

{

\_DeleteElements();

\_CopyElements(other.size, other.elements);

}

Scheme::~Scheme()

{

\_DeleteElements();

}

Scheme& Scheme::operator=(const Scheme& other)

{

if (this == &other)

{

return \*this;

}

\_DeleteElements();

\_CopyElements(other.size, other.elements);

return \*this;

}

BinaryData Scheme::Execute(const BinaryData& data) const

{

// Длина массива даты должна быть вдвое больше длины массива элементов, а длина результата равна ему.

int res\_size = size;

bool\* res\_data = new bool[res\_size];

std::fill(res\_data, res\_data + res\_size, false);

for (int i = 0; i < res\_size; i++)

{

bool r = data[i \* 2], l = data[i \* 2 + 1];

res\_data[i] = elements[i]->Execute(l, r);

}

return BinaryData(res\_data, res\_size);

}

int Scheme::CountOfElements() const

{

return size;

}

void Scheme::\_DeleteElements()

{

if (elements == nullptr)

{

return;

}

for (int i = 0; i < size; i++)

{

delete elements[i];

}

delete[] elements;

elements = nullptr;

size = 0;

}

void Scheme::\_CopyElements(int \_size, LogicalElement\*\* \_elements)

{

if (elements != nullptr)

{

return;

}

size = \_size;

elements = new LogicalElement \* [size];

for (int i = 0; i < size; i++)

{

elements[i] = \_elements[i]->Copy();

}

}

class SchemeFileBuilder

{

public:

SchemeFileBuilder() = delete;

SchemeFileBuilder(const char\* filename, ElementPattern \_pattern);

~SchemeFileBuilder();

Scheme Build();

bool IsOkay();

private:

std::fstream file;

ElementPattern pattern;

void \_ValidateSize(int size) const;

LogicalElement\*\* \_ReadSchemeFromFile(int size);

void \_ToUniqueFormat(std::string& str) const;

bool \_IsArrayOkay(int size, LogicalElement\*\* \_array) const;

};

SchemeFileBuilder::SchemeFileBuilder(const char\* filename, ElementPattern \_pattern)

: file(filename, std::ios::in), pattern(\_pattern)

{

if (file.fail() || file.bad())

{

throw std::invalid\_argument("File with entered name doesn't exist");

}

}

SchemeFileBuilder::~SchemeFileBuilder()

{

file.close();

}

Scheme SchemeFileBuilder::Build()

{

Scheme result;

int size;

LogicalElement\*\* elements = nullptr;

file >> size;

\_ValidateSize(size);

elements = \_ReadSchemeFromFile(size);

if (!\_IsArrayOkay(size, elements))

{

throw new std::runtime\_error("builded scheme is not okay, check file");

}

result.elements = elements;

result.size = size;

file.seekg(0); // Восстановить указатель в файле

return result;

}

bool SchemeFileBuilder::IsOkay()

{

return file.good();

}

void SchemeFileBuilder::\_ValidateSize(int size) const

{

if (size <= 0 && size % 2 == 1)

{

std::stringstream err;

err << "Invalid argument [" << size << "] in input file";

throw new std::invalid\_argument(err.str());

}

}

LogicalElement\*\* SchemeFileBuilder::\_ReadSchemeFromFile(int size)

{

LogicalElement\*\* elements = new LogicalElement \* [size];

std::string type;

for (int i = 0; i < size; i++)

{

file >> type;

\_ToUniqueFormat(type);

elements[i] = pattern.FindByName(type);

}

return elements;

}

void SchemeFileBuilder::\_ToUniqueFormat(std::string& str) const

{

for (auto& i : str)

{

i = toupper(i);

}

}

bool SchemeFileBuilder::\_IsArrayOkay(int size, LogicalElement\*\* \_array) const

{

for (int i = 0; i < size; i++)

{

if (\_array[i] == nullptr)

{

return false;

}

}

return true;

}

class ElementPattern

{

public:

ElementPattern();

ElementPattern(int \_size, LogicalElement\*\* pattern);

ElementPattern(const ElementPattern& other);

~ElementPattern();

ElementPattern& operator=(const ElementPattern& other);

LogicalElement\* FindByName(const std::string& name) const;

private:

void \_DeleteElements();

void \_CopyElements(int \_size, LogicalElement\*\* \_elements);

int size;

LogicalElement\*\* table;

};

ElementPattern::ElementPattern() : table(nullptr), size(0)

{}

ElementPattern::ElementPattern(int \_size, LogicalElement\*\* pattern)

: size(\_size), table(pattern)

{}

ElementPattern::ElementPattern(const ElementPattern& other)

: ElementPattern()

{

\_DeleteElements();

\_CopyElements(other.size, other.table);

}

ElementPattern::~ElementPattern()

{

\_DeleteElements();

}

ElementPattern& ElementPattern::operator=(const ElementPattern& other)

{

if (this == &other)

{

return \*this;

}

\_DeleteElements();

\_CopyElements(other.size, other.table);

return \*this;

}

LogicalElement\* ElementPattern::FindByName(const std::string& name) const

{

for (int i = 0; i < size; i++)

{

std::string type = table[i]->GetName();

if (name.compare(type) == 0)

{

return table[i]->Copy();

}

}

return nullptr;

}

void ElementPattern::\_DeleteElements()

{

if (table == nullptr)

{

return;

}

for (int i = 0; i < size; i++)

{

delete table[i];

}

delete[] table;

table = nullptr;

size = 0;

}

void ElementPattern::\_CopyElements(int \_size, LogicalElement\*\* \_elements)

{

if (table != nullptr)

{

return;

}

size = \_size;

table = new LogicalElement \* [size];

for (int i = 0; i < size; i++)

{

table[i] = \_elements[i]->Copy();

}

}

class BinaryData

{

public:

BinaryData();

BinaryData(bool\* \_buffer, int \_size);

BinaryData(const BinaryData& other);

~BinaryData();

BinaryData& operator=(const BinaryData& other);

bool\* GetBuffer() const;

int GetSize() const;

bool operator[](int index) const;

private:

void \_CopyData(int \_size, bool\* \_buf);

void \_DeleteData();

bool\* buffer;

int size;

};

BinaryData::BinaryData() : buffer(nullptr), size(0)

{}

BinaryData::BinaryData(bool\* \_buffer, int \_size)

: buffer(\_buffer), size(\_size)

{}

BinaryData::BinaryData(const BinaryData& other)

: BinaryData()

{

\_DeleteData();

\_CopyData(other.size, other.buffer);

}

BinaryData::~BinaryData()

{

\_DeleteData();

}

BinaryData& BinaryData::operator=(const BinaryData& other)

{

if (this == &other)

{

return \*this;

}

\_DeleteData();

\_CopyData(other.size, other.buffer);

return \*this;

}

bool\* BinaryData::GetBuffer() const

{

return buffer;

}

int BinaryData::GetSize() const

{

return size;

}

bool BinaryData::operator[](int index) const

{

if (index >= 0 && index < size)

{

return buffer[index];

}

return false;

}

void BinaryData::\_CopyData(int \_size, bool\* \_buf)

{

std::copy(\_buf, \_buf + \_size, buffer);

size = \_size;

}

void BinaryData::\_DeleteData()

{

if (buffer == nullptr)

{

return;

}

delete[] buffer;

size = 0;

buffer = nullptr;

}

class BinaryInputParser

{

public:

static bool ValidateString(const std::string& str);

static BinaryData Parse(const std::string& str, int size);

private:

static bool \_ConvertToBool(char letter);

};

bool BinaryInputParser::ValidateString(const std::string& str)

{

for (auto& letter : str)

{

if (letter < '0' || letter > '1')

{

return false;

}

}

return true;

}

BinaryData BinaryInputParser::Parse(const std::string& str, int size)

{

if (size <= 0)

{

throw new std::invalid\_argument("Invalid size in Parse method");

}

bool\* buf = new bool[size];

std::fill(buf, buf + size, false);

int index = 0;

for (auto l = str.begin(); l != str.end() && index < size; l++, index++)

{

buf[index] = \_ConvertToBool(\*l);

}

return BinaryData(buf, size);

}

bool BinaryInputParser::\_ConvertToBool(char letter)

{

if (letter == '1')

{

return true;

}

return false;

}

class ConsoleInteractor

{

public:

void Start() const;

private:

ElementPattern \_GetDefaultPapptern() const;

};

void ConsoleInteractor::Start() const

{

try

{

cout << "Enter a file name with scheme to build [default = \"scheme.txt\"]\n";

std::string str;

cout << ">>> ";

cin >> str;

SchemeFileBuilder builder(str.c\_str(), \_GetDefaultPapptern());

Scheme scheme = builder.Build();

system("cls");

cout << "\nSuccess!\n\n";

int size\_d = scheme.CountOfElements() \* 2;

cout << "Now it's turn to enter a input signal for scheme\n";

cout << "Signal have to be in format: " << size\_d << " numbers or less. Unwanted symbols will be ignored.\n";

cout << "Signal contains only symbols '1' and '0'\n\n";

cout << ">>> ";

cin >> str;

if (BinaryInputParser::ValidateString(str))

{

BinaryData input = BinaryInputParser::Parse(str, size\_d);

BinaryData output = scheme.Execute(input);

cout << "\bDestination: \n\t";

for (int i = 0; i < input.GetSize(); i++)

{

cout << input[i] << " ";

}

cout << "\n\n";

cout << "\nResult: \n\t";

for (int i = 0; i < output.GetSize(); i++)

{

cout << output[i] << " ";

}

cout << "\n";

}

else

{

cout << "Input error\n";

}

}

catch (const std::exception & e)

{

cerr << e.what() << "\n";

}

}

ElementPattern ConsoleInteractor::\_GetDefaultPapptern() const

{

int size = 3;

LogicalElement\*\* el = new LogicalElement \* [size];

el[0] = new And();

el[1] = new Or();

el[2] = new Xor();

return ElementPattern(size, el);

}