МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« K_3 Вычисление арифметического выражения »

С тудент группы	ИКБО-02-20	Чуков Д.А.
Руководитель практики	Ассистент	Люлява Д.В.
Работа представлена	«___»_____2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

Введение

Объекто-ориентированное программирование - парадигма написания программ, пришедшая на замену функциональному подходу и представляющая из себя подход декомпозиции задачи на отдельные объекты, между которыми и происходит взаимодействие. Преимущество ООП очевидно при необходимости программирования больших систем. Первоначально, когда обширные проекты пытались создавать императивным подходом, то код разрастался до совершенно не поддерживаемых габаритов. В случае появления бага, его невозможно было исправить,так как в такой системе весь код представляет из себя единое целое, и сбой в одной строчке по принципу домино ломает всю программу. ООП позволяет решить эту проблему и обеспечить надежное проектирование, разработку, отладку и поддержку программы благодаря трём основным принципам - инкапсуляции, наследованию и полиморфизму. Инкапсуляция отделяет свойства и методы одного объектаот другого и декларирует, что всякое взаимодействие между ними возможно только по заранее созданным методам. Такой подход гарантирует безопасность, в случае сбоя в одном из объектов, это никак не повредит второй объект, так проще локализовать баг и соответсвенно исправить ошибку. Наследование же оберегает программиста от написания одного и того же кода при описании свойств и методов нескольких объектов похожих классов, вместо этого они объявляются родственными и каждый дочерний класс наследует, и соответсвенно может использовать, те или иные методы родительского класса в зависимости от модификатора наследования. Это делает код чище и избавляет от повторений. Полиморфизм в свою очередь обеспечивает возможность функциям обрабатывать данные разных типов и в сочетании с наследованием дает мощнейщий инструмент проектирования гибких программ со сложной логикой при наименьшем количестве кода. Именно по всем этим причинам в данной работе используется именно ООП подход, что в итоге позволило разработать эффективную и устойчивую программу.

Постановка задачи

Разработать программу, в которой на вход подается целочисленное арифметическое выражение без скобок.
Все операции имеют одинаковый приоритет.
В выражении используются следующие операции: +, -, * и % (целочисленный остаток от деления).
После выполнения каждой операции в составе выражения выдается промежуточный результат.
Работа завершается после выполнения последней операции.

Пример.
Исходное целочисленное арифметическое выражение: 5 + 6 – 1 * 3
Форма вывода промежуточных результатов:
5 + 6 = 11
11 – 1 = 10
10 * 3 = 30

Использовать объекты:
1. Для ввода исходного целочисленного арифметического выражения.
2. Для отработки операции.
3. Для вывода очередного промежуточного результата.

Написать программу, реализующую следующий алгоритм:
1. Ввод арифметического выражения.
2. Определение первого операнда.
3. Выдача «signal_1» и передача первого операнда.
4. Цикл до конца арифметического выражения
   4.1. Определение очередной операции.
   4.2. Выдача «signal_2» и передача символа операции.
   4.3. Определение очередного операнда.
   4.4. Выдача «signal_3» и передача очередного операнда.
5. Конец цикла.

В обработчиках объекта отработки операции реализовать:
При получении «signal_1»: сохранить значение первого операнда.
При получении «signal_2»: сохранить символ операции.
При получении «signal_3»: выполнить операцию, выдать «signal_4» и передать строку для вывода.

Описание входных данных

Целочисленное арифметическое выражение.
Операнды и знаки операций разделены пробелом.

Описание выходных данных

В перовой строке:
«первый операнд»«знак первой операции»«второй операнд» = «результат»

В последующих строках:
«результат предыдущей операции»«знак очередной операции»«очередной операнд» = «результат»

Разделитель один пробел

Метод решения

Для решения задачи воспользуемся объектом app класса Application; объектами стандартного потока ввода и вывода - cin, cout; стандартными функциями getline, to_string, stoi, pow.
Все вводы происходят с клавиатуры, все выводы - на экран.

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	Base			Базовый класс в иерархии классов. Содержит основные поля и методы		
		Application	public		2	
		Operation_handler	public		3	
		Output_device	public		4	
2	Application			Класс корневого		

				объекта (приложения).		
3	Operation_handler			Класс объектов, подчинённых корневому объекту класса Application.		Обеспечивает обработку необходимой операции в соответствии с посланным от корневого объекта сигналом
4	Output_device			Класс объектов, подчинённых корневому объекту класса Application.		Обеспечивает вывод промежуточного результата на экран в соответсвии с посланным от объекта отработки операции сигналом

Класс Base

Методы:

- segmentation; - метод расчленения исходного выражения на отдельные операнды и символы операций, разделенные пробелами. **Модификация метода из КЛ_3_3**

Класс Application

Методы:

- buildTree - метод построения дерева иерархии, создан с опорой на учебный материал [6].
- execute - метод отработки программы
- signal_1 - метод сигнала, который используется для установки связи с методом обработчика handler_1 объекта класса Operation_handler
- signal_2 - метод сигнала, который используется для установки связи с методом обработчика handler_2 объекта класса Operation_handler
- signal_3 - метод сигнала, который используется для установки связи с методом обработчика handler_3 объекта класса Operation_handler

Класс Operation_handler

Поля:

- operation_symbol - символ очередной операции. Тип - символьный. Модификатор доступа - private.
- first_operand - очередной первый операнд. Тип - строковый. Модификатор доступа - private.

- econd_operand - очередной второй операнд. Тип - строковый. Модификатор доступа - private.
- temp_result - промежуточный результат вычислений . Тип - строковый. Модификатор доступа - private.

Методы:

- Operation_handler - параметризованный конструктор класса Operation_handler
- handler_1 - метод обработчика, сохранящий значение первого операнда.
- handler_2 - метод обработчика, сохраняющий символ операции.
- handler_3 - метод обработчика, выполняющий операцию, выдающий метод сигнала signal_4 и передающий строку для вывода.
- signal_4 - метод сигнала, который используется для установки связи с методом обработчика handler_4 объекта класса Output_device

Класс Output_device

Поля:

- line_break - логический флаг, который обеспечивает переход на новую строку только для второго и всех последующих выводов. Тип - логический, Модификатор доступа - private. По умолчанию имеет значение логическая ложь.

Методы:

- Output_device - параметризованный конструктор класса Output_device
- handler_4 - метод обработчика, выводящий результат промежуточных вычислений на экран

Описание алгоритма

Функция: main

Функционал: основной код программы

Параметры: без параметров

Возвращаемое значение: целое - индикатор корректной работы программы

№	Предикат	Действия	№ перехода	Комментарий
1		Создание объекта app класса Application с помощью конструктора, аргументы не	2	

		передаются		
2		Вызов метода buildTree объекта app	3	
3		Вызов метода execute объекта app	4	
4		Возврат нуля	∅	

Класс объекта: Base

Модификатор доступа: protected

Метод: segmentation

Функционал: метод расчленения исходного выражения на отдельные операнды и символы операций, разделенные пробелами.

Параметры: ссылка на строку s

Возвращаемое значение: тип с пустым множеством значением

№	Предикат	Действия	№ перехода	Комментарий
1	Передана пустая строка	Возврат ""	∅	if (s == "")
			2	
2		объявление строковой переменной result	3	
3		объявление целочисленной переменной difference и инициализация единицей	4	чтобы отрезать строку вместе со следующим пробелом
4		объявление константной ссылки на переменную start_delimiter строкового типа и инициализация ""	5	
5		объявление константной ссылки на переменную stop_delimiter строкового типа и инициализация " "	6	
6		объявление беззнаковой переменной first_delimiter_pos и инициализация значением, возвращенным вызовом метода find через ссылку s с параметром start_delimiter	7	ищем начало отсечения. По сути можно не делать, т.к. в кажом случае это нулевой индекс, но пусть
7		объявление беззнаковой переменной last_delimiter_pos и инициализация значением,	8	ищем конец отсечения - первая встреча разделителя,

		возвращенным вызовом метода find_first_of через ссылку s с параметром stop_delimiter, first_delimiter_pos		следующего за индексом first_delimiter_pos
8	В строке остался только последний операнд	присваивание полю last_delimiter значения, возвращенного вызовом метода find_last_of через ссылку s с параметром в виде значения, возвращенного методом back через ссылку s, + 1	9	if (last_delimiter_pos == pow(2, 32) - 1)
			10	
9		присваивание значению difference ноль	10	потому что в этом случае далее пробела нет
10		присваивание значению переменной result значения,возвращенного вызовом метода substr через ссылку s с параметрами first_delimiter_pos, last_delimiter_pos - first_delimiter_pos	11	
11		присваивание значению переданной по ссылке строки s значения, возвращенного вызовом метода substr через ссылку s с параметром last_delimiter_pos + difference	12	
12		Возврат строкового значения переменной result	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: buildTree

Функционал: метод построения дерева иерархии

Параметры: без параметров

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода SetName текущего объекта с параметром "app"	2	
2		Создание объекта класса Operation_handler с помощью конструктора, в качестве	3	

		параметров - указатель на текущий объект this, строка "obj_op_handler", целочисленное число 2, целочисленное число 1. Адрес объект присваиваем указателю obj_op_handler класса Base		
3		Создание объекта класса Output_device с помощью конструктора, в качестве параметров - указатель на текущий объект this, строка "obj_out", целочисленное число 3, целочисленное число 1. Адрес объект присваиваем указателю obj_out класса Base	4	
4		Вызов метода set_connection через указатель this с параметрами: ссылка на метод сигнала signal_1 из области видимости класса Application, преобразованная с помощью макроопределения SIGNAL_D, obj_op_handler, ссылка на метод обработчика handler_1 из области видимости класса Operation_handler, преобразованная с помощью макроопределения HANDLER_D	5	
5		Вызов метода set_connection через указатель this с параметрами: ссылка на метод сигнала signal_2 из области видимости класса Application, преобразованная с помощью макроопределения SIGNAL_D, obj_op_handler, ссылка на метод обработчика handler_2 из области видимости класса Operation_handler, преобразованная с помощью макроопределения HANDLER_D	6	
6		Вызов метода set_connection через указатель this с параметрами: ссылка на метод сигнала signal_3 из области видимости класса Application, преобразованная с помощью макроопределения SIGNAL_D, obj_op_handler, ссылка на метод обработчика handler_3 из области видимости класса Operation_handler, преобразованная с помощью макроопределения HANDLER_D	7	
7		Вызов метода set_connection через указатель obj_op_handler с параметрами: ссылка на метод сигнала signal_4 из области видимости класса Operation_handler, преобразованная с помощью макроопределения SIGNAL_D, obj_out, ссылка на метод обработчика	∅	

		handler_4 из области видимости класса Output_device, преобразованная с помощью макроопределения HANDLER_D		

Класс объекта: Application

Модификатор доступа: public

Метод: signal_1

Функционал: метод сигнала, который используется для установки связи с методом обработчика handler_1 объекта класса Operation_handler

Параметры: ссылка на строку nothing

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Класс объекта: Application

Модификатор доступа: public

Метод: signal_2

Функционал: метод сигнала, который используется для установки связи с методом обработчика handler_2 объекта класса Operation_handler

Параметры: ссылка на строку nothing

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Класс объекта: Application

Модификатор доступа: public

Метод: signal_3

Функционал: метод сигнала, который используется для установки связи с методом обработчика handler_3 объекта класса Operation_handler

Параметры: ссылка на строку nothing

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий

1			∅	

Класс объекта: Operation_handler

Модификатор доступа: public

Метод: Operation_handler

Функционал: Параметризованный конструктор класса Operation_handler

Параметры: parent - указатель на головной объект класса Base, строковое name - наименование текущего объекта, целочисленное class_number - номер класса наследника, целочисленное state_number - число, характеризующее готовность объекта к работе

Возвращаемое значение: не возвращает значения

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов конструктора базового класса Base с параметрами parent, name, class_number, state_number	∅	

Класс объекта: Operation_handler

Модификатор доступа: public

Метод: handler_1

Функционал: метод обработчика, сохранящий значение первого операнда.

Параметры: строка first_operand

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1		Присваивание значению поля first_operand текущего объекта значение параметра first_operand	2	
2		Присваивание значению поля temp_result текущего объекта значение параметра first_operand	∅	

Класс объекта: Operation_handler

Модификатор доступа: public

Метод: handler_2

Функционал: метод обработчика, сохраняющий символ операции.

Параметры: строка symbol

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1		Присваивание значению поля operation_symbol текущего объекта значение первого элемента строки symbol	∅	Symbol имеет тип string (что по сути массив из char), operation_symbol - char, все символы операций состоят из одного символа, поэтому это допустимое решение

Класс объекта: Operation_handler

Модификатор доступа: public

Метод: handler_3

Функционал: метод обработчика, выполняющий операцию, выдающий метод сигнала signal_4 и передающий строку для вывода.

Параметры: строка second_operand

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1		Присваивание значению поля second_operand текущего объекта значение параметра second_operand	2	
2		Присваивание значению поля first_operand текущего объекта значение поля temp_result текущего объекта	3	
3	Был введен символ операции +	Присваивание значению поля temp_result текущего объекта преобразованного к типу string с помощью функции to_string значения суммы значений поля temp_result, преобразованного к типу int с помощью функции stoi и параметра second_operand, преобразованного к типу int с помощью функции stoi	4	
	Был введен символ операции -	Присваивание значению поля temp_result текущего объекта преобразованного к типу string с помощью функции to_string значения разности значений поля temp_result, преобразованного к типу int с помощью функции stoi и параметра second_operand,	4	

		преобразованного к типу int с помощью функции stoi		
	Был введен символ операции *	Присваивание значению поля temp_result текущего объекта преобразованного к типу string с помощью функции to_string значения произведения значений поля temp_result, преобразованного к типу int с помощью функции stoi и параметра second_operand, преобразованного к типу int с помощью функции stoi	4	
	Был введен символ операции %	Присваивание значению поля temp_result текущего объекта преобразованного к типу string с помощью функции to_string значения целочисленного остатка от деления значений поля temp_result, преобразованного к типу int с помощью функции stoi и параметра second_operand, преобразованного к типу int с помощью функции stoi	4	
			4	
4		объявление строки output и инициализация значением: first_operand + " " + operation_symbol + " " + second_operand + " = " + temp_result	5	
5		Вызов метода emit_signal текущего объекта с параметрами ссылка на метод сигнала signal_4 из области видимости класса Operation_handler, преобразованная с помощью макроопределения SIGNAL_D, строка output	∅	

Класс объекта: Operation_handler

Модификатор доступа: public

Метод: signal_4

Функционал: метод сигнала, который используется для установки связи с методом обработчика handler_4 объекта класса Output_device

Параметры: ссылка на строку output

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Класс объекта: Output_device

Модификатор доступа: public

Метод: Output_device

Функционал: Параметризованный конструктор класса Output_device

Параметры: parent - указатель на головной объект класса Base, строковое name - наименование текущего объекта, целочисленное class_number - номер класса наследника, целочисленное state_number - число, характеризующее готовность объекта к работе

Возвращаемое значение: не возвращает значения

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов конструктора базового класса Base с параметрами parent, name, class_number, state_number	∅	

Класс объекта: Output_device

Модификатор доступа: public

Метод: handler_4

Функционал: метод обработчика, выводящий результат промежуточных вычислений на экран

Параметры: строка temp_result

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1	Это НЕ первый вывод промежутчного результата	Переход на новую строку	2	if (line_break)
			2	
2		Вывод параметра temp_result	3	
3		Присваивание полю line_break текущего объекта значения true	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: execute

Функционал: метод отработки программы

Параметры: без параметров

Возвращаемое значение: тип с пустым множеством значений

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление строковых переменных number1, number2, symbol	2	
2		Объявление строки expression	3	
3		Вызов функции getline с параметрами cin, expression	4	
4		Присваивание значению number1 результата, возвращенного вызовом функции segmentation текущего объекта с параметром expression	5	
5		Вызов метода emit_signal текущего объекта с параметрами: ссылка на метод сигнала signal_1 из области видимости класса Application, преобразованная с помощью макроопределения SIGNAL_D, строка number1	6	
6	Арифметическое выражение НЕ посчитано до конца		7	while (expression != "")
			∅	
7		Присваивание значению symbol результата, возвращенного вызовом функции segmentation текущего объекта с параметром expression	8	
8		Вызов метода emit_signal текущего объекта с параметрами: ссылка на метод сигнала signal_2 из области видимости класса Application, преобразованная с помощью макроопределения SIGNAL_D, строка symbol	9	
9		Присваивание значению number2 результата, возвращенного вызовом функции segmentation текущего объекта с параметром expression	10	
10		Вызов метода emit_signal текущего объекта с параметрами: ссылка на метод сигнала signal_3 из области	6	

		видимости класса Application, преобразованная с помощью макроопределения SIGNAL_D, строка number2		

Блок-схема алгоритма

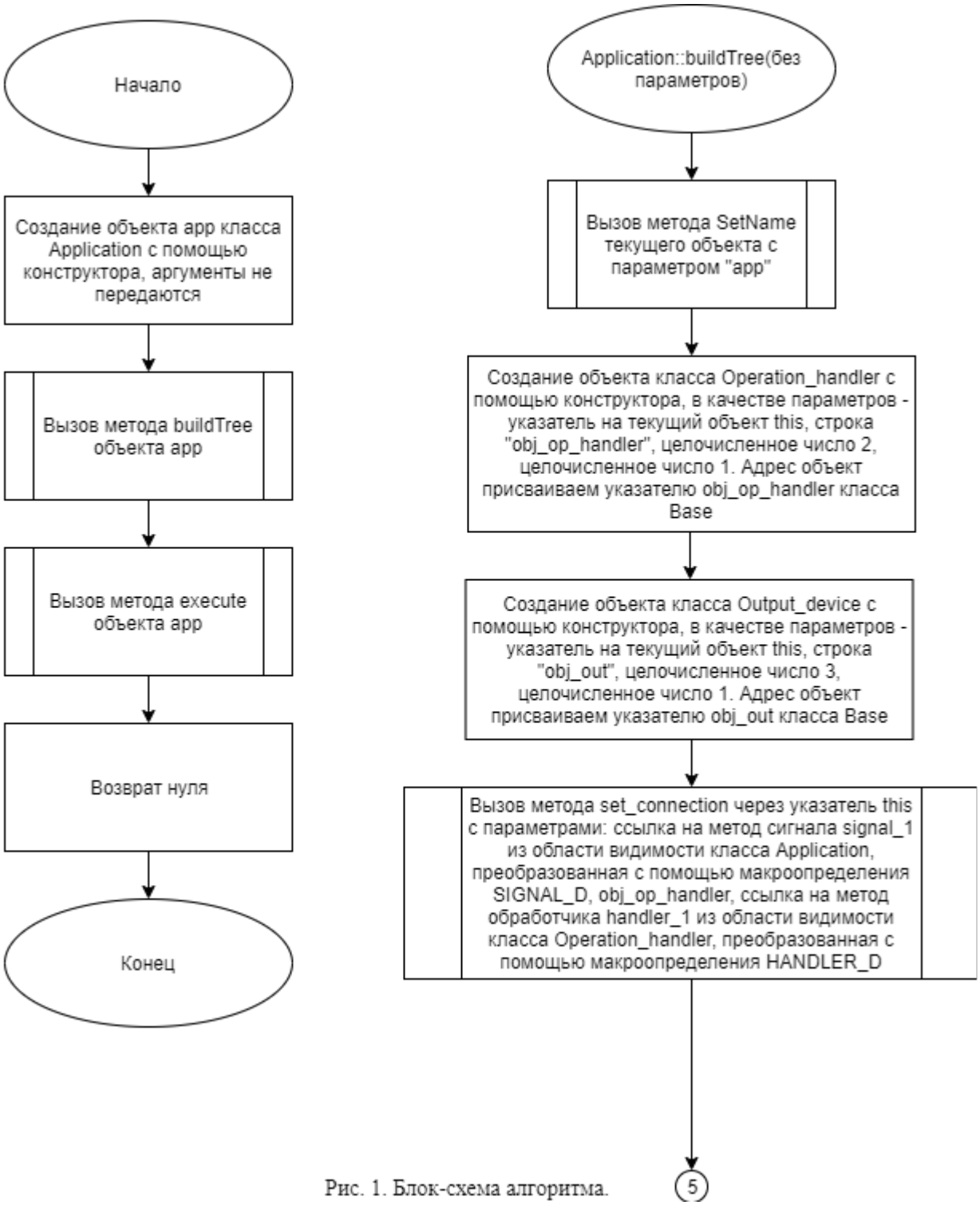

Рис. 1. Блок-схема алгоритма.

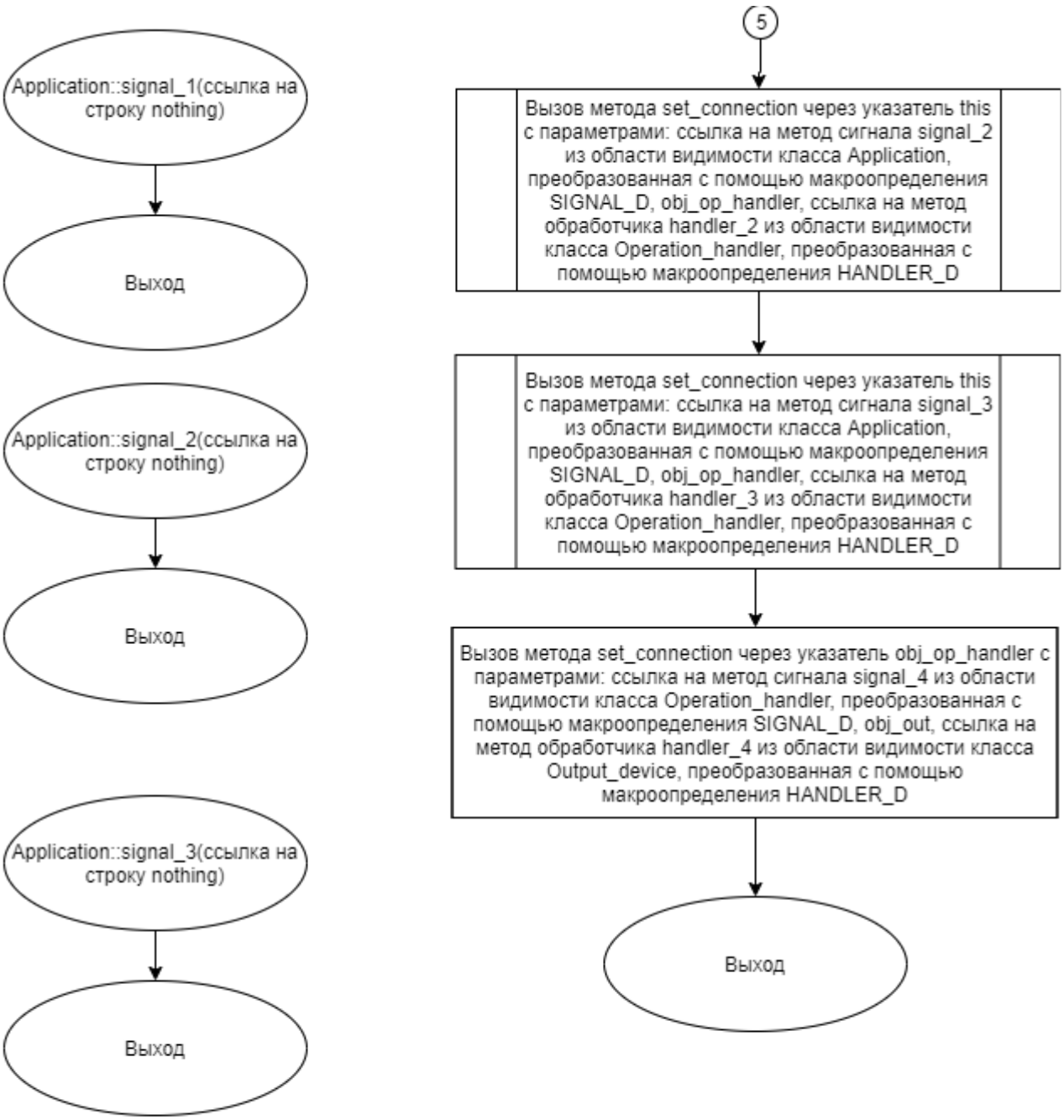

Рис. 2. Блок-схема алгоритма.

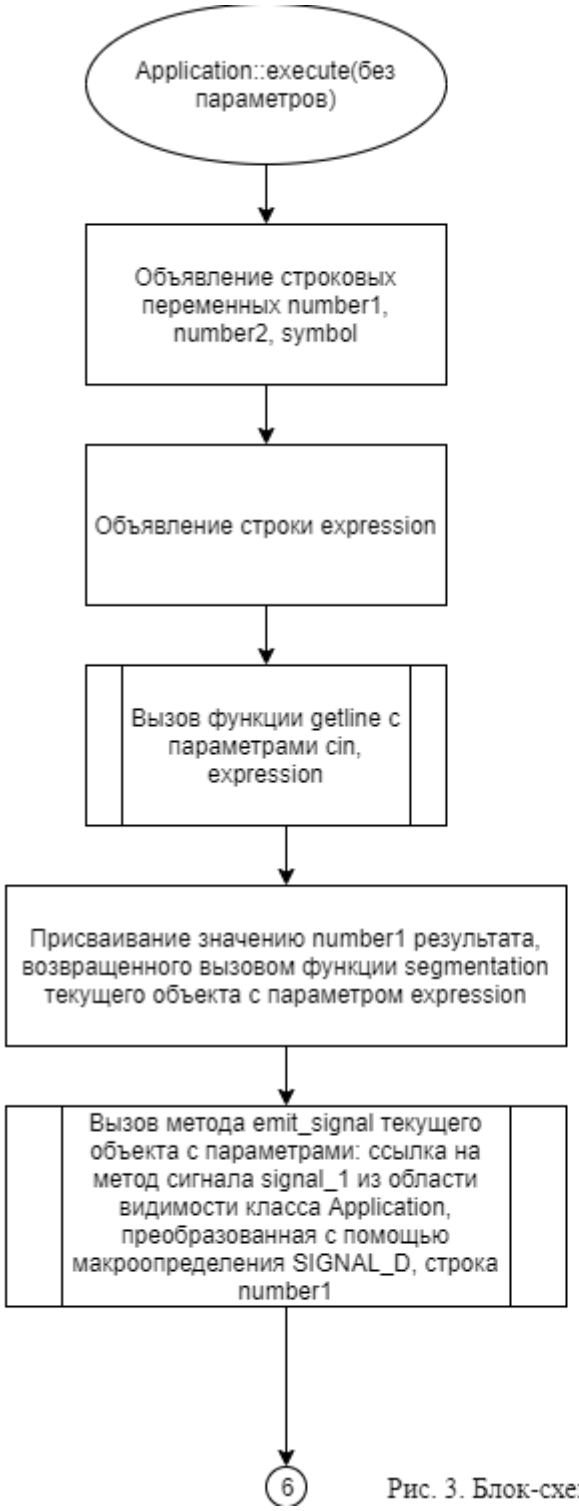

Рис. 3. Блок-схема алгоритма.

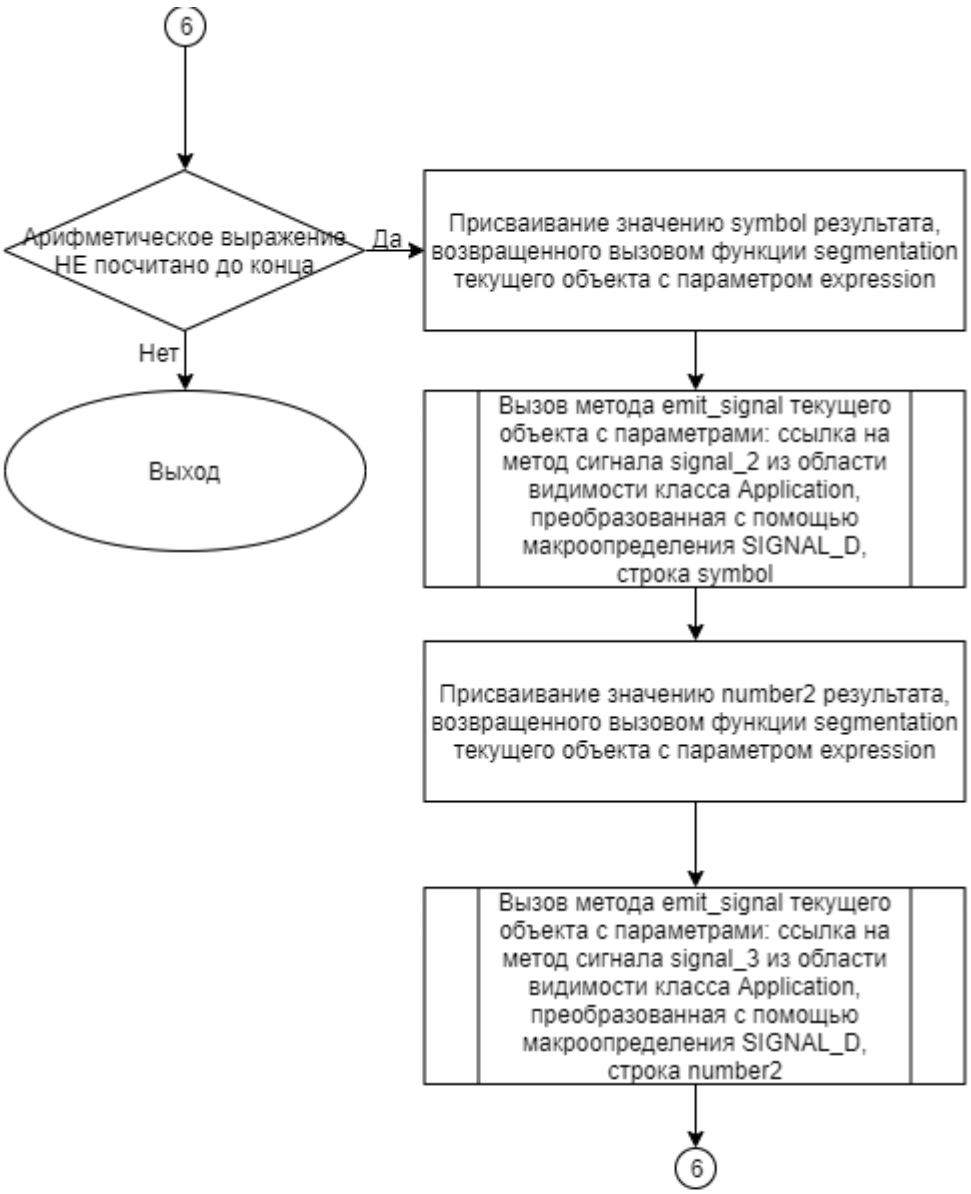

Рис. 4. Блок-схема алгоритма.

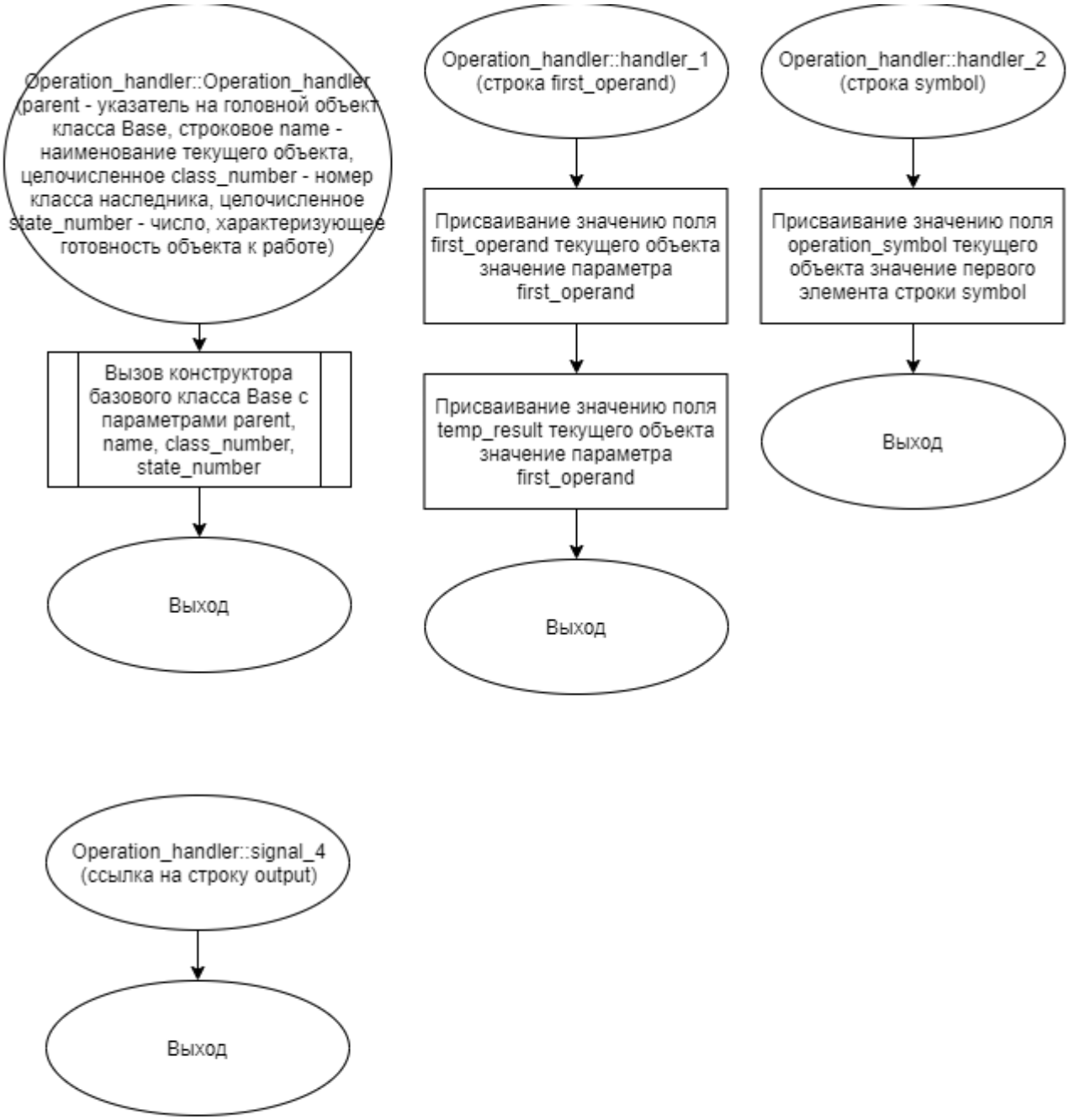

Рис. 5. Блок-схема алгоритма.

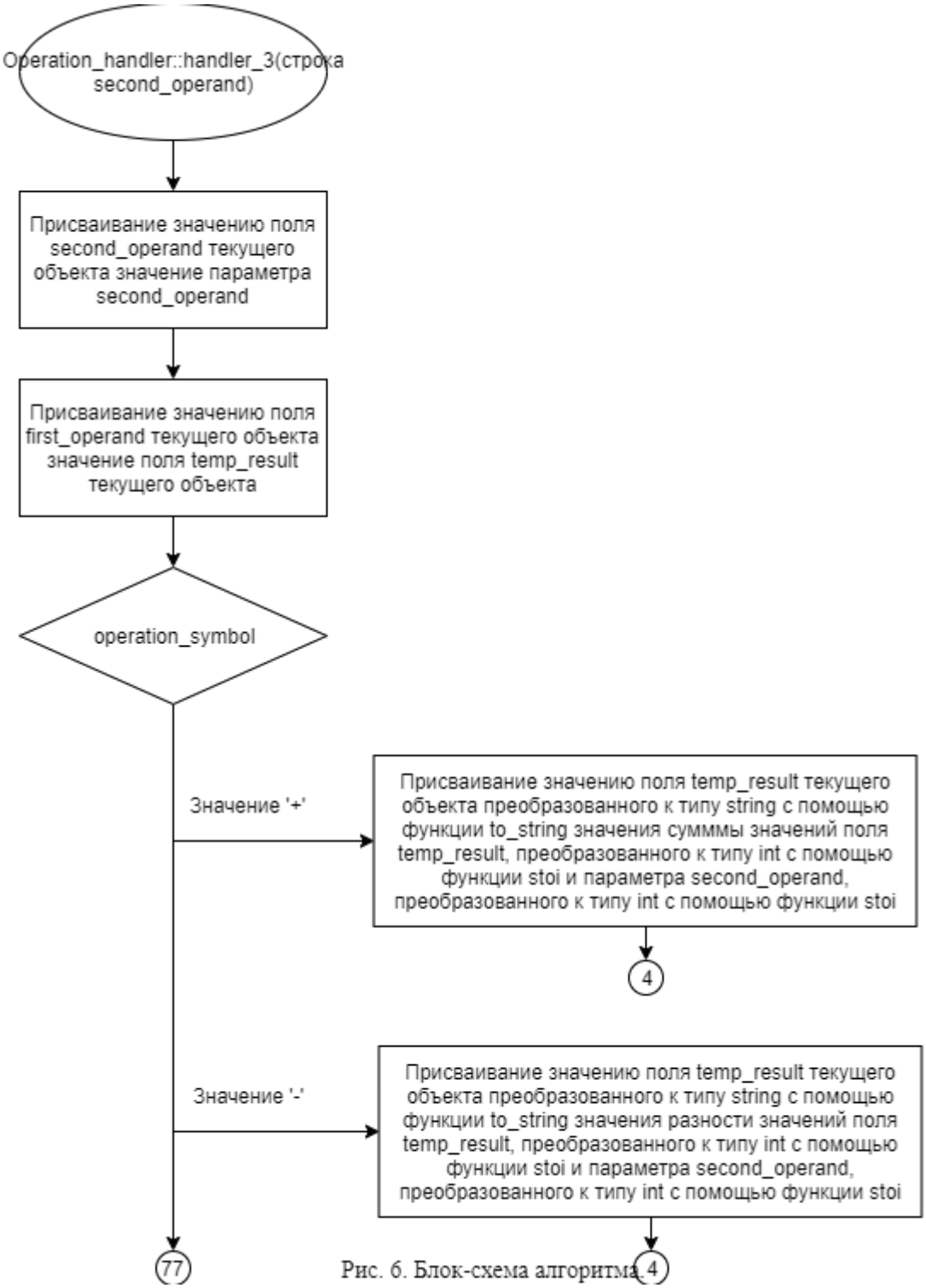

Рис. 6. Блок-схема алгоритма

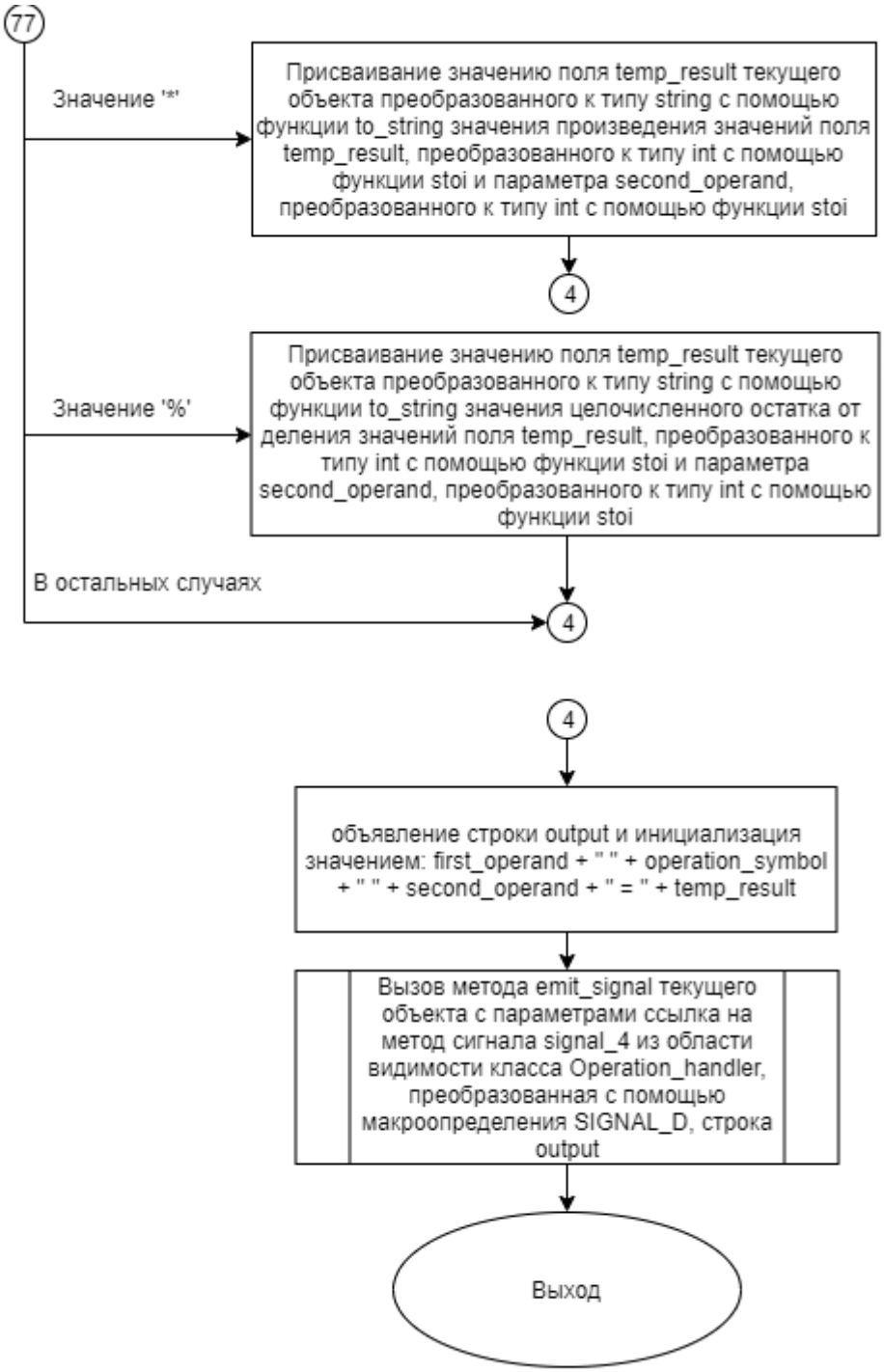

Рис. 7. Блок-схема алгоритма.

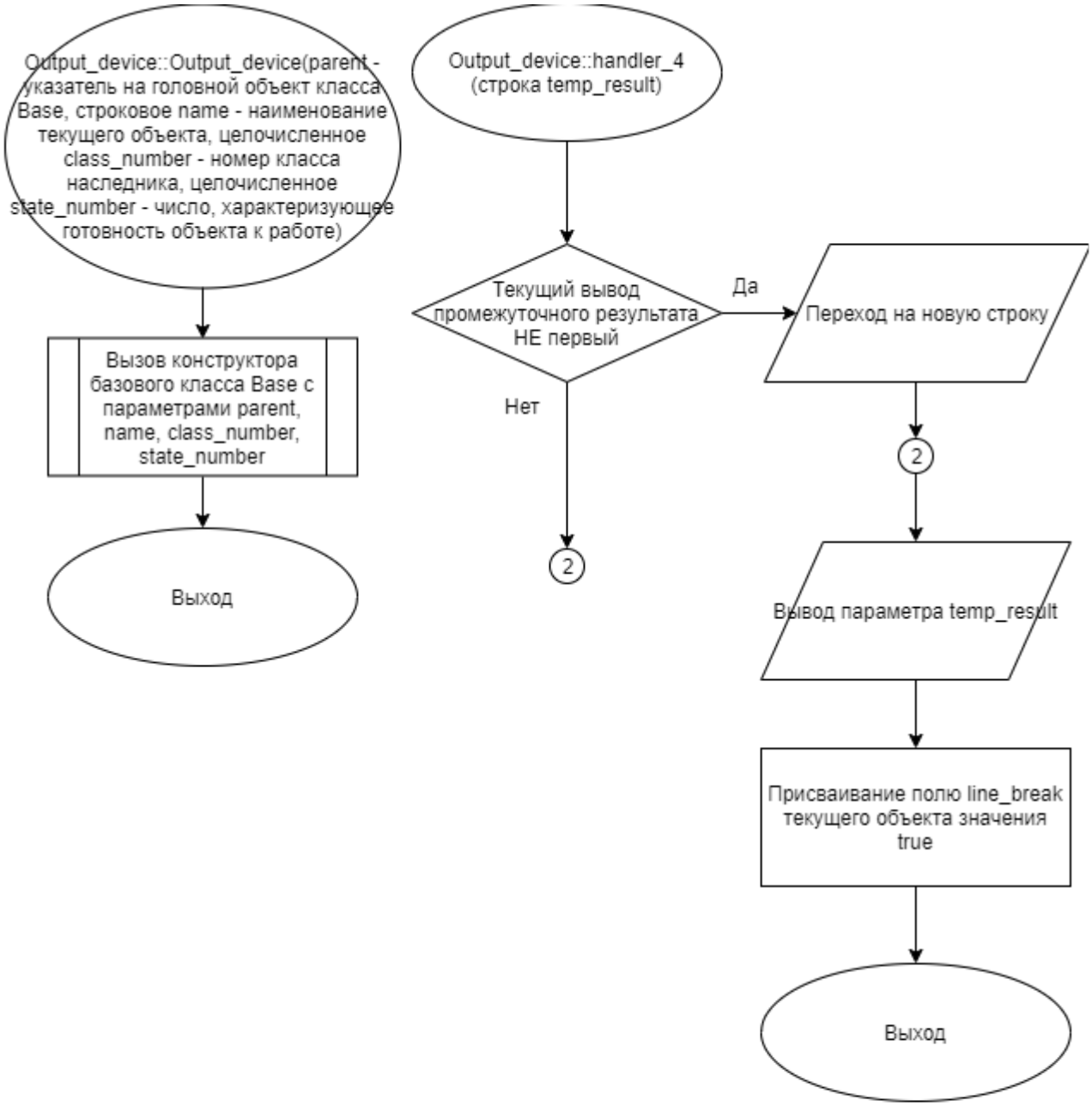

Рис. 8. Блок-схема алгоритма.

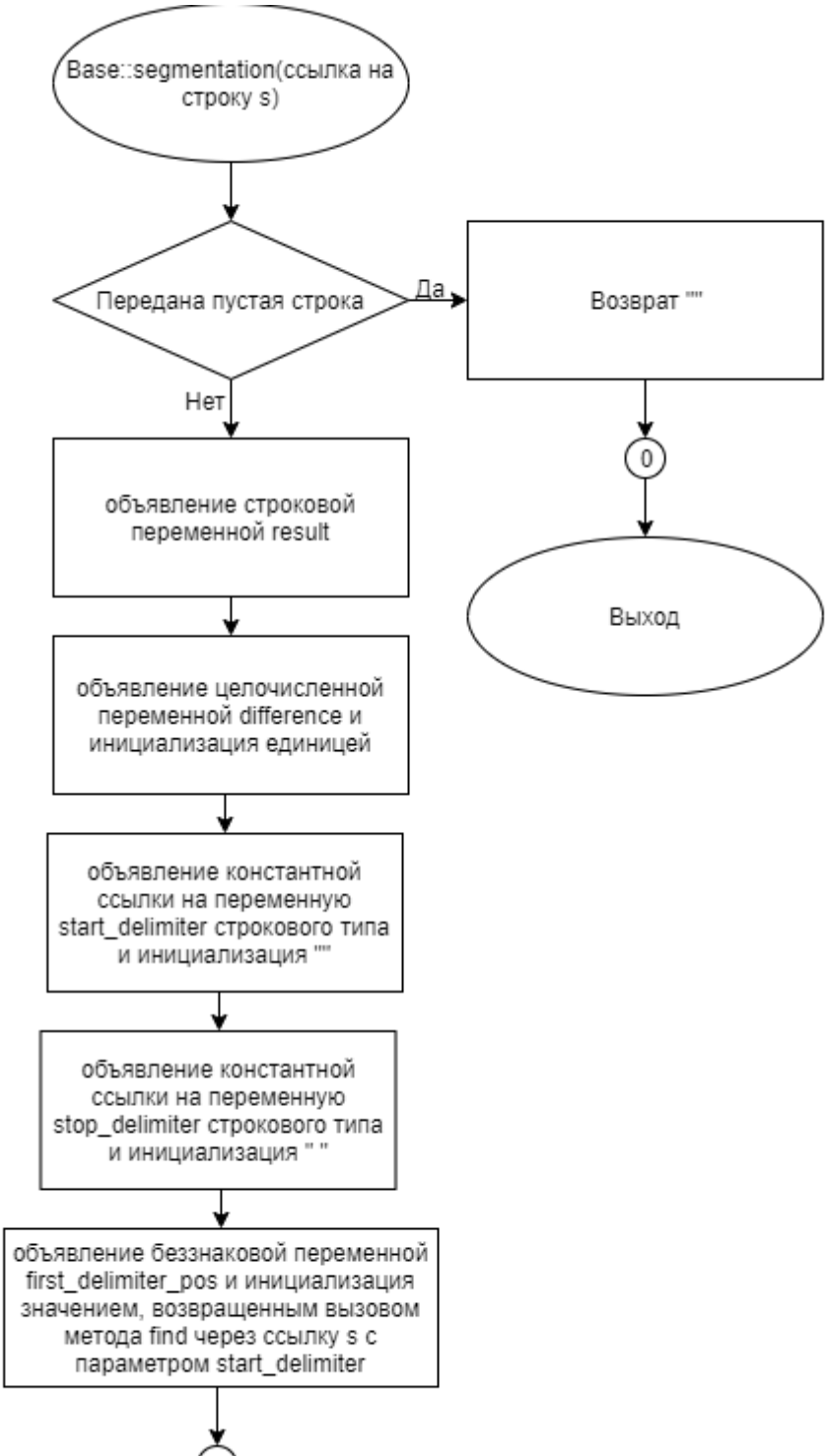

Рис. 9. Блок-схема алгоритма.

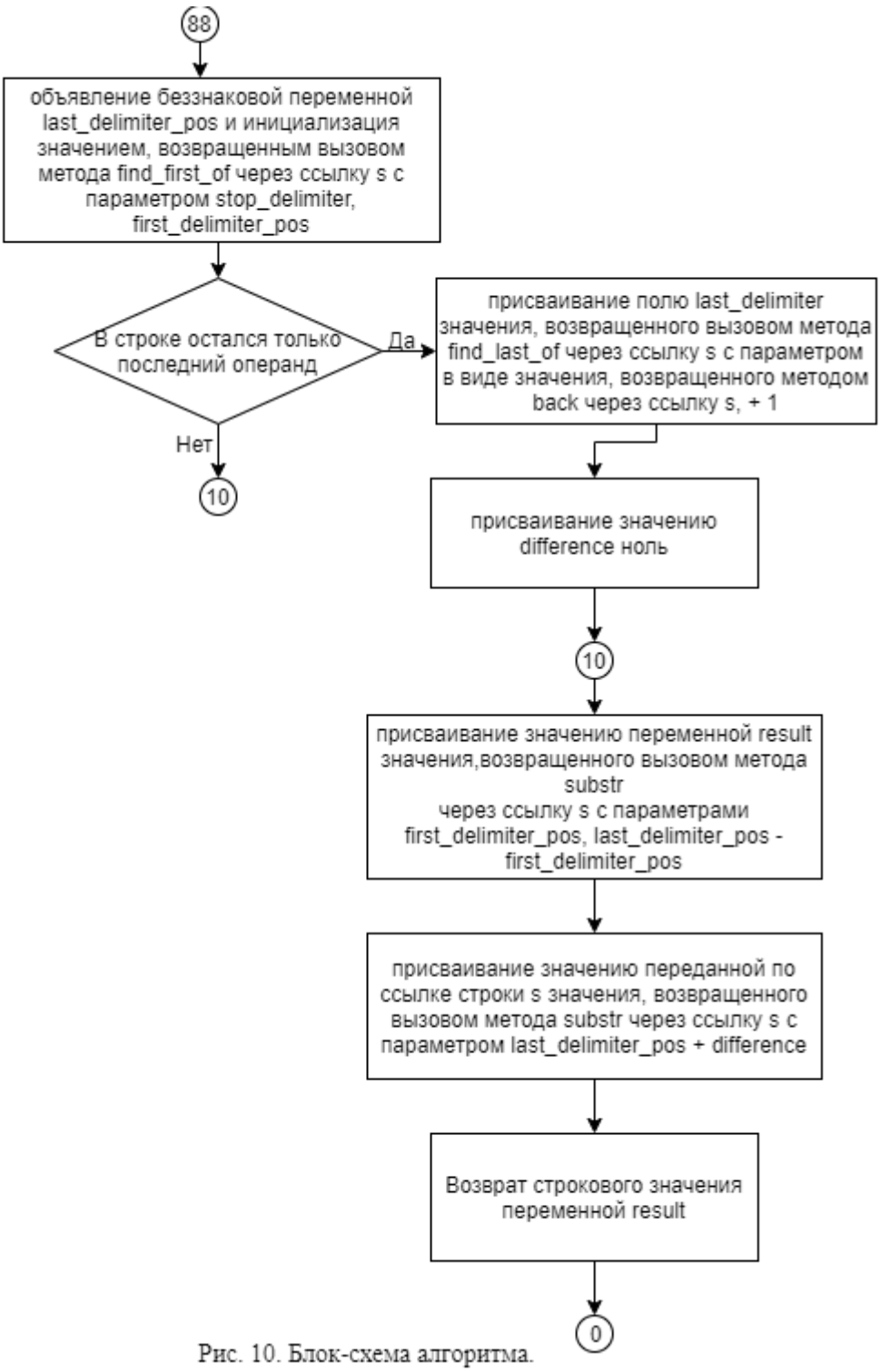

Рис. 10. Блок-схема алгоритма.

Код программы

Файл Application.cpp

```
#include "Application.h"

using namespace std;

Application::Application(Base* parent, int class_number, int state_number) :
Base(parent,"",class_number,state_number) {
}

void Application::buildTree() {
        setName("app");
        Base* obj_op_handler = new Operation_handler(this, "obj_op_handler",
2, 1);
        Base* obj_out = new Output_device(this, "obj_out", 3, 1);

        this->set_connection(SIGNAL_D(&Application::signal_1), obj_op_handler,
HANDLER_D(&Operation_handler::handler_1));
        this->set_connection(SIGNAL_D(&Application::signal_2), obj_op_handler,
HANDLER_D(&Operation_handler::handler_2));
        this->set_connection(SIGNAL_D(&Application::signal_3), obj_op_handler,
HANDLER_D(&Operation_handler::handler_3));
        obj_op_handler->set_connection(SIGNAL_D(&Operation_handler::signal_4),
obj_out, HANDLER_D(&Output_device::handler_4));
};


int Application::read_data() {
        if (this->getName() != "endtree") {
                cout << endl << "Test result" << endl;
                cout << "The object " << this->getName() << " is ready";
                this->read_base_data();
        }
        return 0;
}

void Application::start_emitting_signals() {
        cout << endl << "Emit signals";
        while (true) {
                string object_signal_name;
                string text;
                cin >> object_signal_name;
                if (object_signal_name == "endsignals") {
                        break;
                }
                cin >> text;
                Base* signal = root->search_by_name(object_signal_name);
                ptr_s ptr_this_s = 0;

                /*switch (signal->class_number) {
                case 1: ptr_this_s = SIGNAL_D(&Derived::send_signal); break;
                case 2: ptr_this_s = SIGNAL_D(&Derived2::send_signal); break;
```

```
                case 3: ptr_this_s = SIGNAL_D(&Derived3::send_signal); break;
                }*/
                signal->emit_signal(ptr_this_s, text);
        };
}

void Application::execute() {
        string number1, number2, symbol;
        string expression;
        getline(cin, expression);
        number1 = segmentation(expression); //определение первого операнда
        emit_signal(SIGNAL_D(&Application::signal_1), number1); //send_signal-
1

        while (expression != "") {
                symbol = segmentation(expression); //определение очередной
операции
                emit_signal(SIGNAL_D(&Application::signal_2), symbol);
//send_signal-2
                number2 = segmentation(expression); //определение очередного
операнда
                emit_signal(SIGNAL_D(&Application::signal_3), number2);
//send_signal-3
        }
        //OutputTree();
        //read_data();
}
```

Файл Application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H
#include "Base.h"
#include "Operation_handler.h"
#include "Output_device.h"

class Application : public Base {
        Base* root_parent = 0;
        Base* temp_child = 0;

public:
        Application(Base* parent = 0, int class_number = 1, int state_number =
1);
        void buildTree();
        void execute();
        int read_data();
        void start_emitting_signals();
        void signal_1(string& nothing) {};
        void signal_2(string& nothing) {};
        void signal_3(string& nothing) {};

};

#endif
```

Файл Base.cpp

```
#include "Base.h"

int Base::i;
Base* Base::root;

Base::Base(Base* parent, string name, int class_number, int state_number) {
	i = 0;
	setParent(parent);
	setName(name);
	setPath();
	this->class_number = class_number;
	this->state_number = state_number;

}

void Base::setName(string name) {
	this->name = name;
}

string Base::getName() {
	return this->name;
}

void Base::setPath() {
	if (parent) {
		this->path += parent->getPath();
	}
	if (name != "") {
		this->path += "/" + this->name;
	}
}

string Base::getPath() {
	return this->path;
}

void Base::setParent(Base* parent) {
	this->parent = parent;
	if (parent) {
		parent->children.push_back(this);
	}
}

void Base::OutputTree() {
	if (!i) { //условие при котором "шапка" выводится только один раз
		cout << "Object tree";
		cout << endl << name;
	}
	if (children.empty()) {
		return;
	}
	children_iterator = children.begin();
	while (children_iterator != children.end()) {
		tabulation(1);
```

```
                cout << endl << tabulation(0);
                cout << (*children_iterator)->getName();
                (*children_iterator)->OutputTree();
                tabulation(-1);
                children_iterator++;
        }
}

string Base::tabulation(int a) {
        string this_tabulation = "";
        string tab = "    ";
        if (a == 1) {
                ++i;
        }
        if (a == -1) {
                --i;
        }
        for (int j = 0; j < i; j++) {
                this_tabulation.append(tab);
        }
        return this_tabulation;
}

Base* Base::getParent() {
        return parent;
};

void Base::read_base_data() {
        children_iterator = children.begin();
        while (children_iterator != children.end()) {
                cout << endl;
                cout << "The object " << (*children_iterator)->getName();
                if ((*children_iterator)->state_number > 0) {
                        cout << " is ready";
                }
                else { cout << " is not ready"; }
                (*children_iterator)->read_base_data();
                children_iterator++;
        }
}

string Base::segmentation(string& s) {
        if (s == "") {
                return "";
        }
        string result;
        int difference = 1; //to substract together with space
        const string& start_delimiter = ""; //there has been "/"
        const string& stop_delimiter = " "; //there has been "/"
        unsigned first_delimiter_pos = s.find(start_delimiter); //there has
been + 1
        unsigned last_delimiter_pos = s.find_first_of(stop_delimiter,
first_delimiter_pos);
        if (last_delimiter_pos == pow(2, 32) - 1) {
                last_delimiter_pos = s.find_last_of(s.back()) + 1;
                difference = 0;
```

```
        }
        result = s.substr(first_delimiter_pos, last_delimiter_pos -
first_delimiter_pos);
        s = s.substr(last_delimiter_pos + difference);
        return (result);
};

Base* Base::search(string path) {
        if (path.substr(0, 2) == "//") { //если адрес передан через //, то
ищем по имени
                string temp = path;
                segmentation(temp); //убираем первый слэш
                return search_by_name(segmentation(temp)); //убираем второй и
вызываем поиск по имени
        };
        string local = segmentation(path); //достаем имя верхней директории
        if (local == this->getName() && path == "") {
                return this;
        }
        if (local == this->getName()) {
                local = segmentation(path);
        }
        children_iterator = children.begin();
        while (children_iterator != children.end()) {
                if (local == (*children_iterator)->getName()) {
                        if (path == "") {
                                return *children_iterator;
                        }
                        else {
                                return (*children_iterator)->search(path);
                        }
                }
                children_iterator++;
        }
        return NULL;
}

Base* Base::search_by_name(string name) {
        if (name == this->getName()) return this;
        children_iterator = children.begin();
        while (children_iterator != children.end()) {
                if (name == (*children_iterator)->getName()) {
                        return (*children_iterator);
                }
                if ((*children_iterator)->search_by_name(name) != NULL) {
                        return (*children_iterator)->search_by_name(name);
                }
                children_iterator++;
        }
        return NULL;
}

Base* Base::search_by_path(string path) {
        if (search(path)) {
                string temp = "";
                temp = search(path)->getPath() + " " + "Object name: " +
search(path)->getName();
                cout << endl << temp;
                return search(path);
        }
```

```
        else {
                string temp = "";
                temp = path + " " + "Object not found";
                cout << endl << temp;
                return NULL;
        }
}

void Base::set_connection(ptr_s p_signal, Base* p_ob_handler, ptr_h p_handler)
{
        o_sh* new_struct = new o_sh;
        if (connections.size() > 0) {
                connections_iterator = connections.begin();

                while (connections_iterator != connections.end()) {
                        if ((*connections_iterator)->p_signal == p_signal &&
(*connections_iterator)->p_Base == p_ob_handler && (*connections_iterator)-
>p_handler == p_handler) {
                                return;
                        }
                        connections_iterator++;
                }
        }
        new_struct->p_signal = p_signal;
        new_struct->p_Base = p_ob_handler;
        new_struct->p_handler = p_handler;
        connections.push_back(new_struct);
};

void Base::delete_connection(ptr_s p_signal, Base* p_ob_handler, ptr_h
p_handler) {
        Base* signal = 0;
        Base* handler = 0;
        if (connections.empty()) return;

        connections_iterator = connections.begin();
        while (connections_iterator != connections.end()) {
                if ((*connections_iterator)->p_signal == p_signal &&
((*connections_iterator))->p_Base == p_ob_handler
                        && ((*connections_iterator))->p_handler == p_handler)
{
                        signal = this;
                        handler = (*connections_iterator)->p_Base;
                        connections.erase(connections_iterator);
                        break;
                }
                connections_iterator++;
        }
};

void Base::emit_signal(ptr_s s_ignal, string& s_command) {
        if (connections.empty()) return;
        (this->*s_ignal) (s_command); //посылаем строковый сигнал на
преобразование
        connections_iterator = connections.begin();
```

```
        while (connections_iterator != connections.end()) {
                if ((*connections_iterator)->p_signal == s_ignal) {
                        ptr_h p_handler = (*connections_iterator)-
>p_handler; //метод обработчика
                        Base* ob_handler = (*connections_iterator)-
>p_Base; //указатель на объект-обработчик
                        (ob_handler->*p_handler)(s_command); //вызов метода
обработчика с параметром в виде уже преобразованного сигнала
                }
                connections_iterator++;
        }
};
```

Файл Base.h

```
#ifndef BASE_H
#define BASE_H
#include <iostream>
#include <vector>
#include <cmath>
#include <string>
#include <tuple>
#define SIGNAL_D(  signal_f  ) ( ( void ( Base::* ) ( string & ) )
( ( signal_f  ) ) )
#define HANDLER_D( handler_f ) ( ( void ( Base::* ) ( string ) ) ( ( handler_f
) ) )
using namespace std;
class Base;
typedef void (Base::* ptr_s)(string&);
typedef void (Base::* ptr_h)(string);

class Base {
        static int i;
        string name;
        Base* parent;
        int state_number;
        string path;
        vector<Base*> children;
        vector<Base*>::iterator children_iterator;

        struct o_sh {
                ptr_s p_signal;
                Base* p_Base;
                ptr_h p_handler;
        };
        vector<o_sh*> connections;
        vector<o_sh*>::iterator connections_iterator;

public:
        static Base* root;
        int class_number;
        Base(Base*, string, int class_number, int state_number);
        string getPath();
        string getName();
        Base* getParent();
        Base* search_by_name(string);
        Base* search_by_path(string);
        void OutputTree();

        void show_all_connections();
        void set_connection(ptr_s, Base* p_ob_handler, ptr_h);
        void delete_connection(ptr_s, Base* p_ob_handler, ptr_h);
        void emit_signal(ptr_s, string& s_command);

protected:
```

```
        string tabulation(int);
        void setName(string);
        void setPath();
        void setParent(Base*);
        Base* search(string);
        void read_base_data();
        string segmentation(string&);
};

#endif
```

Файл main.cpp

```
#include "Application.h"

int main() {
        Application app;
        app.buildTree();
        app.execute();
        return 0;
}
```

Файл Operation_handler.cpp

```
#include "Operation_handler.h"

Operation_handler::Operation_handler(Base* parent, string name, int
class_number, int state_number) : Base(parent, name, class_number,
state_number) {}

void Operation_handler::handler_1(string first_operand) {
        this->first_operand = first_operand;
        temp_result = first_operand;
};

void Operation_handler::handler_2(string symbol) {
        this->operation_symbol = symbol[0];
};

void Operation_handler::handler_3(string second_operand) {
        this->second_operand = second_operand;
        first_operand = temp_result;
        switch (operation_symbol) {
        case '+': temp_result = to_string(stoi(temp_result) +
stoi(second_operand)); break; //+
        case '-': temp_result = to_string(stoi(temp_result) -
stoi(second_operand)); break; //-
        case '*': temp_result = to_string(stoi(temp_result) *
stoi(second_operand)); break; //*
        case '%': temp_result = to_string(stoi(temp_result) %
stoi(second_operand)); break; //%
        }
        string output = first_operand + " " + operation_symbol + " " +
second_operand + " = " + temp_result;
        emit_signal(SIGNAL_D(&Operation_handler::signal_4), output);
};

void Operation_handler::signal_4(string& output) {};
```

Файл Operation_handler.h

```
#ifndef OPERATION_HANDLER_H
#define OPERATION_HANDLER_H
#include "Base.h"

class Operation_handler : public Base {
        char operation_symbol;
        string first_operand;
        string second_operand;
        string temp_result;
public:
        Operation_handler(Base*, string, int, int);
        void handler_1(string);
        void handler_2(string);
        void handler_3(string);
        void signal_4(string&);
};
#endif
```

Файл Output_device.cpp

```
#include "Output_device.h"

Output_device::Output_device(Base* parent, string name, int class_number, int
state_number) : Base(parent, name, class_number, state_number) {}

void Output_device::handler_4(string temp_result) {
        if (line_break) {
                cout << endl;
        }
        cout << temp_result;
        line_break = true;
};
```

Файл Output_device.h

```
#ifndef OUTPUT_DEVICE_H
#define OUTPUT_DEVICE_H
#include "Base.h"

class Output_device : public Base {
        bool line_break = false;
public:
        Output_device(Base*, string, int, int);
        void handler_4(string);
};

#endif
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
5 + 6 - 1 * 3	5 + 6 = 11 11 - 1 = 10 10 * 3 = 30	5 + 6 = 11 11 - 1 = 10 10 * 3 = 30
15 - 5 * 9 % 15 - 404 + 505	15 - 5 = 10 10 * 9 = 90 90 % 15 = 0 0 - 404 = -404 -404 + 505 = 101	15 - 5 = 10 10 * 9 = 90 90 % 15 = 0 0 - 404 = -404 -404 + 505 = 101

Заключение

В результате выполнения работы я приобрел навыки написания программ в ООП стиле на примере реализации объекто-ориентированного подхода в языке программирования C++. В том числе на конкретных учебных задачах я научился разрабатывать базовый класс для объектов, определять общий функционал для используемых в рамках приложения объектов, разрабатывать операции добавления, удаления, изменения позиции объекта в рамках иерархического дерева, построению дерева иерархии объектов, переключению состояния объектов и определению их готовности к работе, выводить на печать дерево иерархии объектов, искать указатель на объект по координате на дереве иерархии объектов или по имени, при уникальности наименований объектов.

Также мною были получены навыки проектирования, написания, отладки и модификации программ в ООП стиле. Я изучил все стандартные конструкции и основные библиотеки языка C++, ознакомился с соответвующей документацией, освоил весь материал, изложенный в учебных пособиях по данной теме. При возникновении трудностей обращался за консультацией к профессиональному сообществу на различных интернет-ресурсах, к своим семинаристам и лектору. Все полученые знания не раз применил на практике при решении задач.

Объекто-ориентированный подход является наиболее распространенной методикой разработки на данный момент, и навык написания программм в данной парадигме является основополагающим для любого современного программиста. Данная курсовая работа помогла мне развить соответсвующие компетенции и приобрести необходимый опыт, который будет очень востребован при построении будущей карьеры и устройстве на работу.

Список используемой литературы (источников)

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://online-edu.mirea.ru/mod/resource/view.php?id=247030 (дата обращения 05.05.2021).

4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).