СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

Курсовая работа выполненена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы зафиксированы и соответствуют требованиям, изложенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине "Объектно-ориентированное программирование" [2-3] и методике разработки объектно-ориентированных программ [4-6].

Объектно-ориентированное программирование (ООП) представляет собой мощный и широко применяемый подход к созданию программного обеспечения. ООП предлагает набор принципов, методов и инструментов, которые способствуют созданию модульных, гибких и легко поддерживаемых приложений. Каждому программисту важно владеть объектно-ориентированным программированием по многим причинам.

Во-первых, из-за его широкого распространения. ООП является одним из наиболее часто используемых методов разработки программного обеспечения. Знание ООП расширяет возможности программиста и делает его более востребованным на рынке труда. Многие компании и проекты используют объектно-ориентированный подход, что делает владение этим навыком особенно ценным.

Во-вторых, благодаря модульности и возможности повторного использования кода. ООП позволяет структурировать программу в виде модулей, называемых классами, которые объединяют данные и методы, связанные с определенной функциональностью. Это облегчает повторное использование кода и упрощает поддержку и развитие программы.

В-третьих, ООП способствует созданию более понятного и читаемого кода. Использование классов и объектов в ООП позволяет представлять концепции реального мира непосредственно в коде, что облегчает его понимание.

В-четвертых, принципы ООП играют важную роль. Основные концепции объектно-ориентированного программирования, часто называемые "тремя китами ООП", включают инкапсуляцию, наследование и полиморфизм. Инкапсуляция позволяет скрывать внутренние детали реализации, повышая безопасность и понятность кода. Полиморфизм позволяет использовать один и тот же код для работы с различными типами данных, что упрощает код и повышает его гибкость.

Кроме того, объектно-ориентированный подход улучшает совместную работу и разделение ответственности. Классы могут быть разработаны и реализованы независимо друг от друга, а затем объединены в одно приложение. Это позволяет команде разработчиков работать параллельно над разными модулями системы.

Таким образом, знание и понимание основ объектно-ориентированного программирования имеет большое значение для программиста. ООП - это широко применяемый подход, который обеспечивает модульность, гибкость и легкость поддержки современного программного обеспечения.

1 ПОСТАНОВКА ЗАДАЧИ

Надо моделировать работу калькулятора следующей конструкции:

- в вычислении участвуют целые числа объемом памяти 2 байта;
- допустимые операции: +, -, *, / (целочисленное деление), % (деление с остатком), << (побитовый сдвиг влево), >> (побитовый сдвиг в право);
- операции выполняются последовательно, для выполнения операции необходимы два аргумента и знак операции;
- после выполнения каждой операции фиксируется и выводится результат;
- последовательность операций и аргументов образует выражение;
- результат отображается в 16, 10 и 2-ой системе счисления;
- при возникновении переполнения выдается Overflow;
- при попытке деления на 0 выдается Division by zero;
- при вводе знака “C” калькулятор приводиться в исходное состояние, первый аргумент выражения принимает значение 0 и готов для ввода очередного выражения;
- при вводе знака “Off” калькулятор завершает работу.

Нажатие на клавиши калькулятора моделируется посредством клавиатурного ввода. Ввод делится на команды:

- «целое число» - первый аргумент выражения, целое не отрицательное число, можно последовательно вводить несколько раз, предыдущее значение меняется. При вводе не первым аргументом выражения - игнорируется;
- «знак операции» «целое число» - второе и последующие операции выражения;
- «C» - приведение калькулятора в исходное состояние;
- «Off» - завершение работы калькулятора.

Вывод результата моделируется посредством вывода на консоли. Результат

выводиться в следующей форме:

```
«выражение»     HEX «16-ое число»  DEC «10-ое число»  BIN «2-ое число»
```

«16-ое число» выводиться в верхнем регистре с лидирующими нулями (пример 01FA).

«10-ое число» (пример 1765).

«2-ое число» выводиться разбивкой по четыре цифры с лидирующими нулями (пример 0000 0100 0111 0101).

Построить систему, которая использует объекты:

1. Объект «система».
2. Объект для чтения команд. После чтения очередной команды объект выдает сигнал с текстом, содержащим команду. Все команды синтаксический корректны (моделирует пульт управления калькулятора).
3. Объект для выполнения арифметических операции. После завершения выдается сигнал с текстом результата. Если произошло переполнение или деление на нуль, выдается сигнал об ошибке. После выдачи сообщения калькулятор переводится посредством соответствующего сигнала в исходное положение.
4. Объект для выполнения операции побитового сдвига. После завершения выдается сигнал с текстом результата.
5. Объект для выполнения операции «C».
6. Объект для вывода очередного результата на консоль.

Написать программу, реализующую следующий алгоритм:

1. Вызов метода объекта «система» build_tree_objects ( ).
    1.1. Построение дерева иерархии объектов.
    1.2. Установка связей сигналов и обработчиков между объектами.
2. Вызов метода объекта «система» exec_app ( ).
    2.1. Приведение всех объектов в состояние готовности.

2.2. Цикл для обработки вводимых команд.

2.2.1. Выдача сигнала объекту для ввода команды.

2.2.2. Отработка команды.

2.3. После ввода команды «Off» завершить работу.

Все приведенные сигналы и соответствующие обработчики должны быть реализованы.

Все сообщения на консоль выводятся с новой строки.

В набор поддерживаемых команд добавить команду «SHOWTREE» и по этой команде вывести дерево иерархии объектов системы с отметкой о готовности и завершить работу программы.

1.1 Описание входных данных

Построчно множество команд, в любом количестве. Перечень команд:

```
«целое не отрицательное число»
«знак операции» «целое число»
C
```

Последняя команда присутствует всегда:

```
Off
```

Пример ввода:

```
5
+ 5
<< 1
/ 0
+ 5
C
7
8
/ -3
C
9
% -4
+ 7
* 11
Off
```

1.2 Описание выходных данных

Построчно выводиться результат каждой операции по форме:

```
«выражение»     HEX «16-ое число»  DEC «10-ое число»  BIN «2-ое число»
```

Если произошло переполнение:

```
«выражение»     Overflow
```

Если произошло переполнение:

```
«выражение»     Division by zero
```

Пример вывода:

```
5 + 5     HEX 000A  DEC 10  BIN 0000 0000 0000 1010
5 + 5 << 1     HEX 0014  DEC 20  BIN 0000 0000 0001 0100
5 + 5 << 1 / 0     Division by zero
0 + 5     HEX 0005  DEC 5  BIN 0000 0000 0000 0101
8 / -3     HEX FFFE  DEC -2  BIN 1111 1111 1111 1110
9 % -4     HEX 0001  DEC 1  BIN 0000 0000 0000 0001
9 % -4 + 7     HEX 0008  DEC 8  BIN 0000 0000 0000 1000
9 % -4 + 7 * 11     HEX 0058  DEC 88  BIN 0000 0000 0101 1000
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- функция main для основная функция;
- объект стандартного потока ввода/ вывода;
- условный оператор.

Класс cl_application:

- свойства/поля:
    - поле строка:
        - наименование — s_cmd;
        - тип — Строковый;
        - модификатор доступа — private;
- функционал:
    - метод signal_app_to_reader — сигнал приложения к считывателю;
    - метод handler_app_from_reader — обработчик приложения от считывателя;
    - метод build_tree_objects — создание объектов и их связей;
    - метод exec_app — запуск приложения;
    - метод show_tree — вывод дерева иерархии объектов системы с отметкой о готовности.

Класс cl_reader:

- функционал:
    - метод handler_reader_from_app — считывание команд;
    - метод signal_reader_to_all — сигнал от reader.

Класс cl_calc:

- функционал:
    - метод signal_calc_to_screen — сигнал к экрану;

- метод handler_calc_from_reader — обработка арифметических операций.

Класс cl_cancel:

- функционал:
  - метод handler_cancel_from_reader — обработчик сигнала сброса.

Класс cl_shift:

- функционал:
  - метод handler_shift_from_reader — обработка арифметических операций;
  - метод signal_shift_to_screen — сигнал к screen.

Класс cl_screen:

- функционал:
  - метод handler_screen_from_all — вывод результата;
  - метод toBinary — перевод в 2-ую систему счисления.

Класс cl_base:

- свойства/поля:
  - поле :
    - наименование — s_expression;
    - тип — Строковый;
    - модификатор доступа — public;
  - поле :
    - наименование — s_operation;
    - тип — Строковый;
    - модификатор доступа — public;
  - поле :
    - наименование — s_operand_2;
    - тип — Строковый;

        - модификатор доступа — public;
    - поле :
        - наименование — i_result;
        - тип — Целочисленный;
        - модификатор доступа — public;
    - поле :
        - наименование — f;
        - тип — Целочисленный;
        - модификатор доступа — public;

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_application			Класс корневого объекта (объект "Система")	
2	cl_reader			Класс, отвечающий за ввод	
3	cl_calc			Класс, отвечающий за арифметический подсчёт	
4	cl_cancel			Класс, отвечающий за сброс	
5	cl_shift			Класс, отвечающий за побитовый сдвиг	
6	cl_screen			Класс, отвечающий за вывод	
7	cl_base			Базовый класс	
		cl_application	public		1
		cl_reader	public		2
		cl_calc	public		3
		cl_cancel	public		4
		cl_shift	public		5
		cl_screen	public		6

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм метода signal_app_to_reader класса cl_application

Функционал: Сигнал приложения к считывателю.

Параметры: string& msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода signal_app_to_reader класса cl_application

№	Предикат	Действия	№ перехода
1			∅

3.2 Алгоритм метода handler_app_from_reader класса cl_application

Функционал: Обработчик приложения от считывателя.

Параметры: string msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода handler_app_from_reader класса cl_application

№	Предикат	Действия	№ перехода
1	Команда равна Off	Оканчивание ввода	∅

№	Предикат	Действия	№ перехода
			2
2	Команда равна SHOWTREE	Отправка сигнала	∅
			∅

3.3 Алгоритм функции main

Функционал: Основная функция.

Параметры: Отсутствуют.

Возвращаемое значение: Текст.

Алгоритм функции представлен в таблице 4.

Таблица 4 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта obj класса cl_application	2
2		Вызов метода build_tree_objects()	3
3		Запуск программы при помощи метода exec_app	∅

3.4 Алгоритм метода handler_reader_from_app класса cl_reader

Функционал: Считывание команд.

Параметры: string msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода handler_reader_from_app класса cl_reader

№	Предикат	Действия	№ перехода
1		Инициализация переменной, хранящей команду	2
2	команда равна "C" или	Отправка сигнала	∅

№	Предикат	Действия	№ перехода
	команда равна "Off"		
			3
3	команда равна "+" или команда равна "-" или команда равна "*" или команда равна "%" или команда равна "/" или команда равна "<<" или команда равна ">>"	Отправка сигнала с командой	∅

3.5 Алгоритм метода signal_reader_to_all класса cl_reader

Функционал: Сигнал от reader.

Параметры: string& msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода signal_reader_to_all класса cl_reader

№	Предикат	Действия	№ перехода
1			∅

3.6 Алгоритм метода signal_calc_to_screen класса cl_calc

Функционал: Сигнал к экрану.

Параметры: string& msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода signal_calc_to_screen класса cl_calc

№	Предикат	Действия	№ перехода
1			∅

3.7 Алгоритм метода handler_calc_from_reader класса cl_calc

Функционал: Обработка арифметических операций.

Параметры: string msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода handler_calc_from_reader класса cl_calc

№	Предикат	Действия	№ перехода
1	msg равно "+"	Сложение операндов	7
			2
2	msg равно "-"	Вычитание операндов	7
			3
3	msg равно "*"	Умножение операндов	7
			4
4	msg равно "/"		5
			6
5	Деление на 0?	Оповещение screen о делении на 0	7
		Целочисленное деление	7
6	msg равно "%"	Деление с остатком	7
			7
7		Отправка сигнала	∅

3.8 Алгоритм метода handler_cancel_from_reader класса cl_cancel

Функционал: Обработчик сигнала сброса.

Параметры: string msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода handler_cancel_from_reader класса cl_cancel

№	Предикат	Действия	№ перехода
1	msg равно "C"	Приводит состояние калькулятора к исходному	∅
			∅

3.9 Алгоритм метода build_tree_objects класса cl_application

Функционал: Создание объектов и их связей.

Параметры: Отсутсвуют.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Установка имени древа	2
2		Создание объектов, подчиняющихся этому	3
3		Создание связей между объектами	∅

3.10 Алгоритм метода exec_app класса cl_application

Функционал: Запуск приложения.

Параметры: Отсутствуют.

Возвращаемое значение: Целочисленное.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Устанавливается готовность всех объектов	2
2		Бесконечный цикл с отправкой сигнала	∅

3.11 Алгоритм метода handler_shift_from_reader класса cl_shift

Функционал: Обработка арифметических операций.

Параметры: string msg.

Возвращаемое значение: Отсутсвует.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода handler_shift_from_reader класса cl_shift

№	Предикат	Действия	№ перехода
1	msg равно "<<"	Побитовый сдвиг влево	3
			2
2	msg равно ">>"	Побитовый сдвиг вправо	3
			3
3		Отправка сигнала	∅

3.12 Алгоритм метода signal_shift_to_screen класса cl_shift

Функционал: Сигнал к screen.

Параметры: string& msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода signal_shift_to_screen класса cl_shift

№	Предикат	Действия	№ перехода
1			∅

3.13 Алгоритм метода handler_screen_from_all класса cl_screen

Функционал: Вывод результата.

Параметры: string msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода handler_screen_from_all класса cl_screen

№	Предикат	Действия	№ перехода
1		Создание переменной типа ostringstream, хранящей результат в 16-ой системе счисления	2
2	Команда равна "C" или команда равна "Off"		∅
			3
3	Команда равна "+" или команда равна "-" или команда равна "*" или команда равна "/" или команда равна "%" или команда равна "<<" или команда равна ">>"	Отправка сигнала с командой	∅
			4
4	Выход за границы 2 битного числа	Вывод Overflow	∅
			5
5	Деление на 0	Вывод Division by zero	∅

№	Предикат	Действия	№ перехода
			6
6		Вывод результата в нужной форме	∅

3.14 Алгоритм метода toBinary класса cl_screen

Функционал: Перевод в 2-ую систему счисления.

Параметры: unsigned int i.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода toBinary класса cl_screen

№	Предикат	Действия	№ перехода
1		Перевод в 2-ую систему счисления при помощи библиотеки bitset	2
2		Разделение по тетрадам с помощью циклов	3
3		Вывод результата	∅

3.15 Алгоритм метода show_tree класса cl_application

Функционал: Вывод дерева иерархии объектов системы с отметкой о готовности.

Параметры: string& msg.

Возвращаемое значение: Отсутствует.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода show_tree класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод дерева иерархии объектов системы с отметкой о готовности	2
2		Завершение программы	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-8.

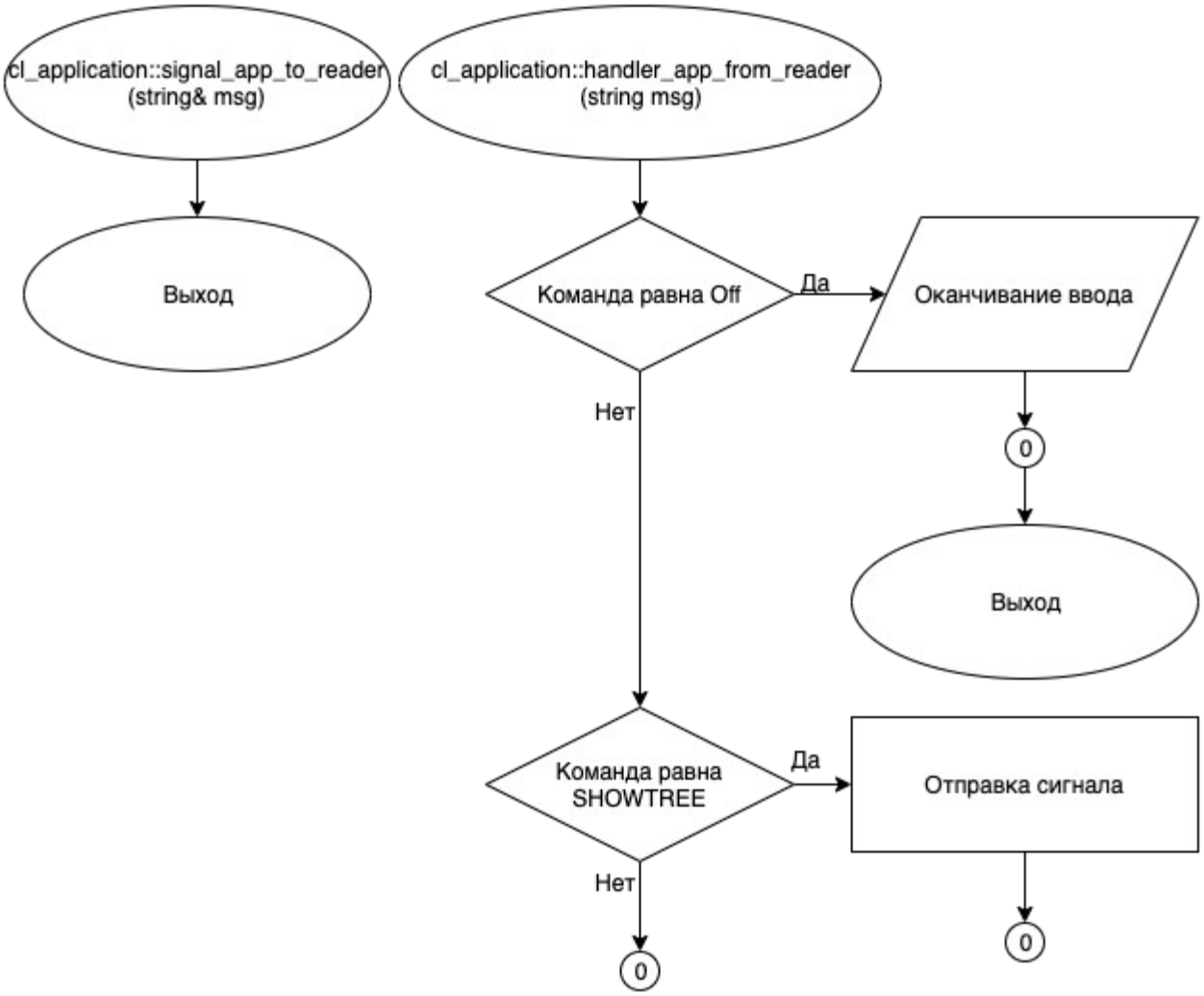

Рисунок 1 – Блок-схема алгоритма

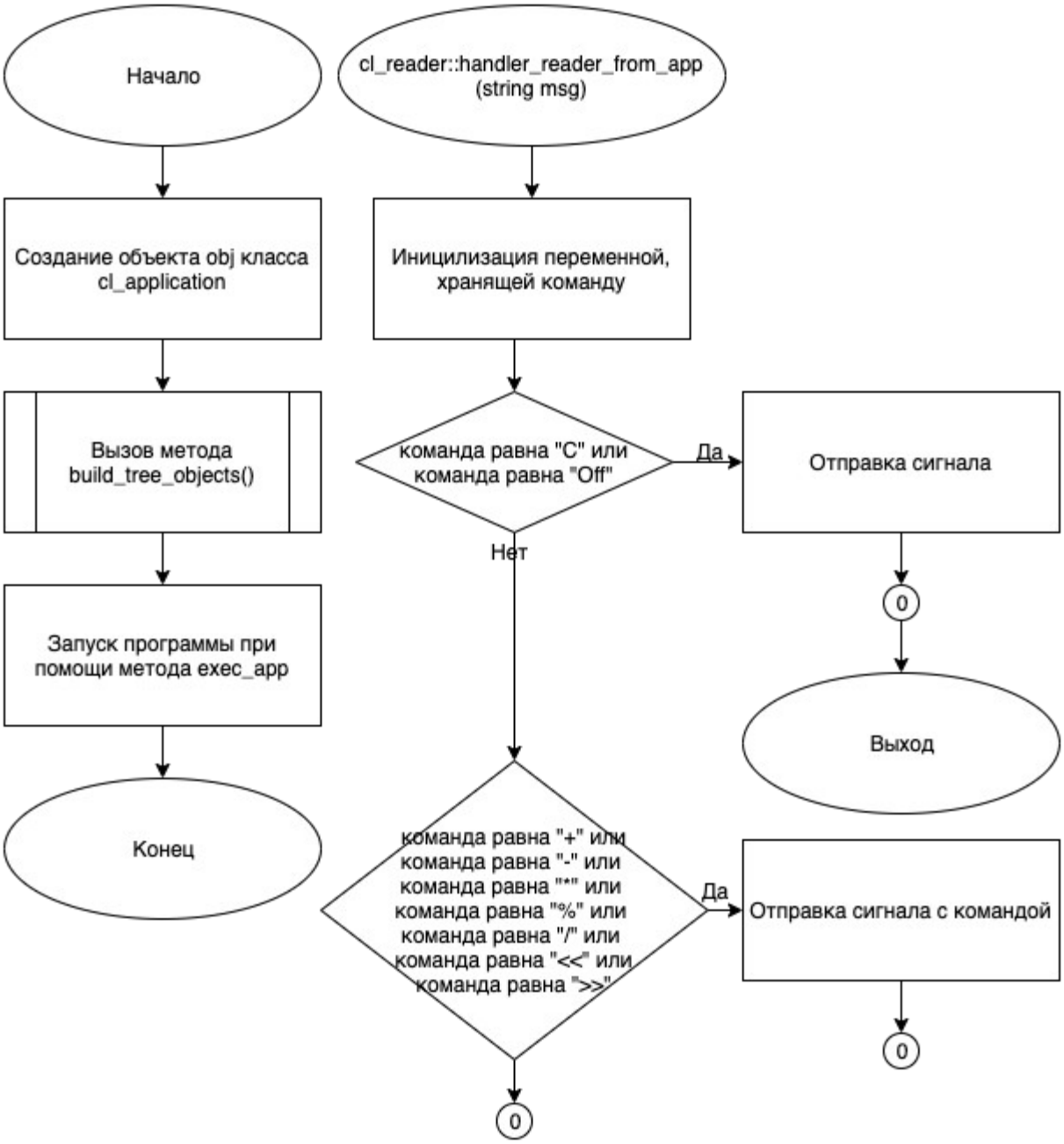

Рисунок 2 – Блок-схема алгоритма

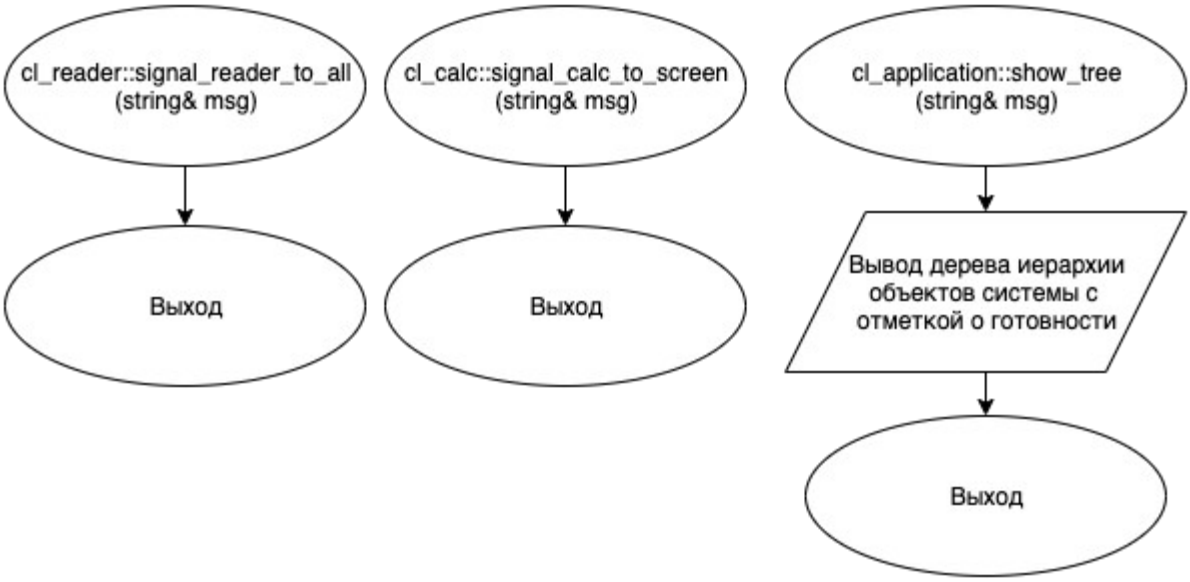

Рисунок 3 – Блок-схема алгоритма

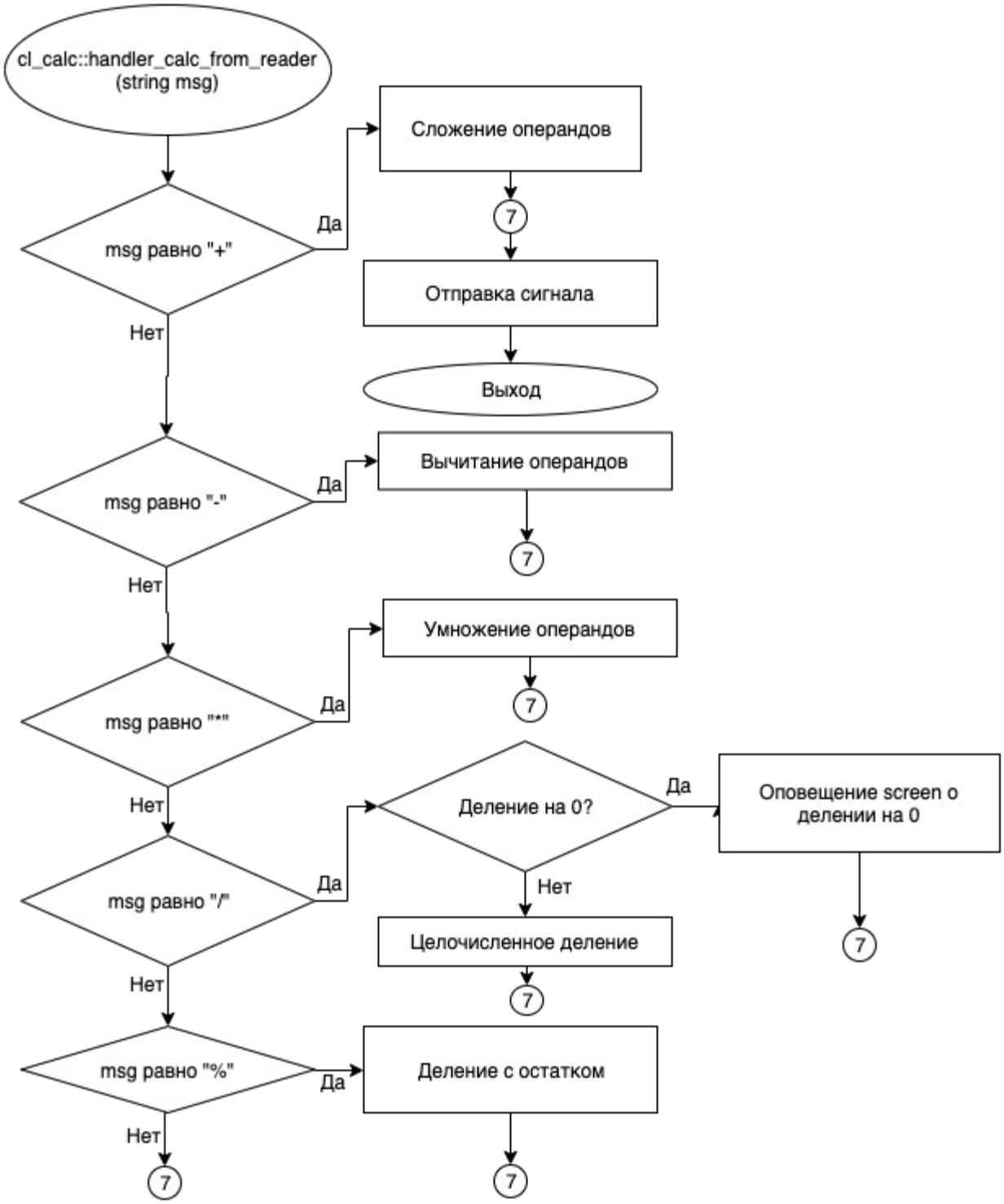

Рисунок 4 – Блок-схема алгоритма

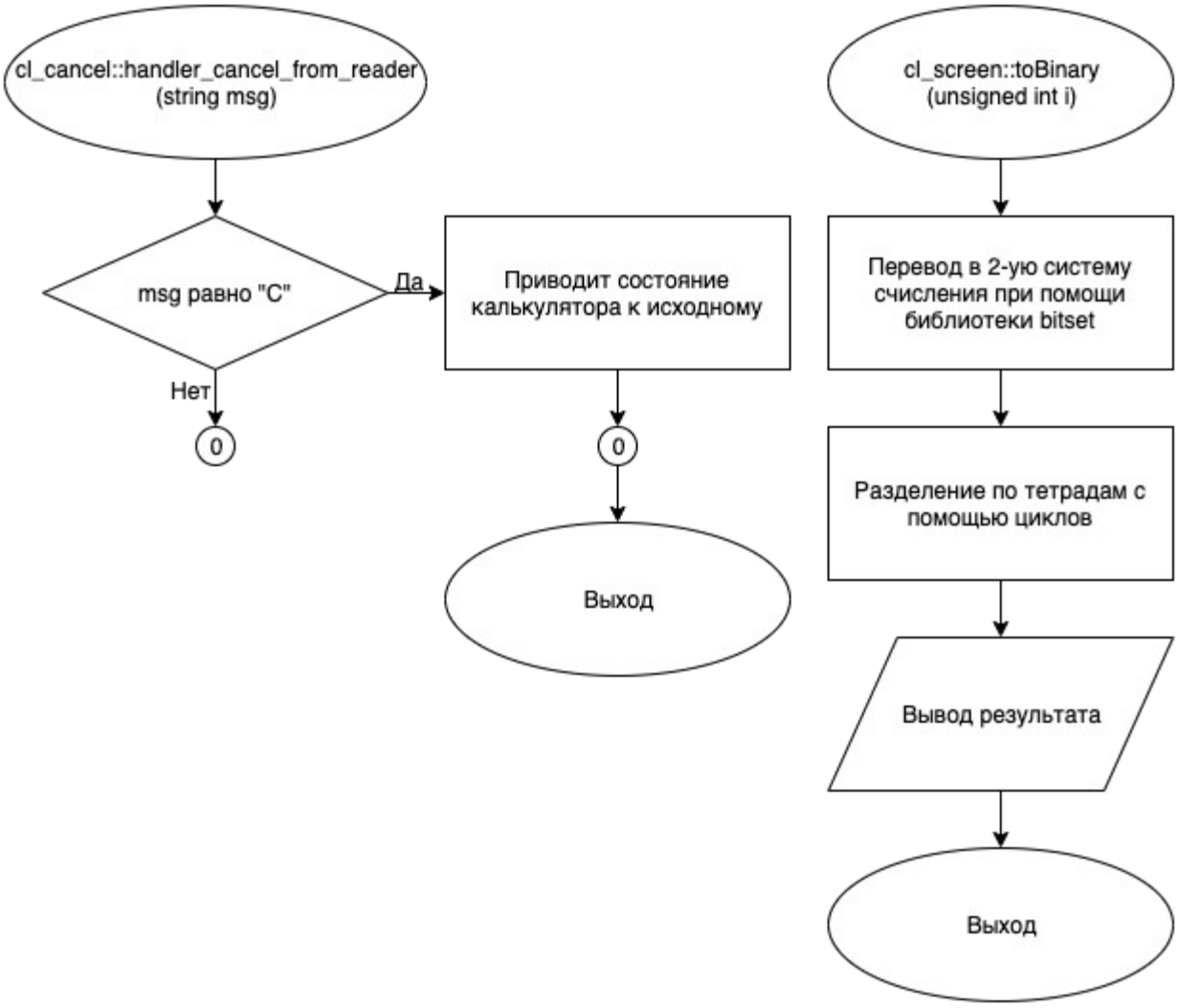

Рисунок 5 – Блок-схема алгоритма

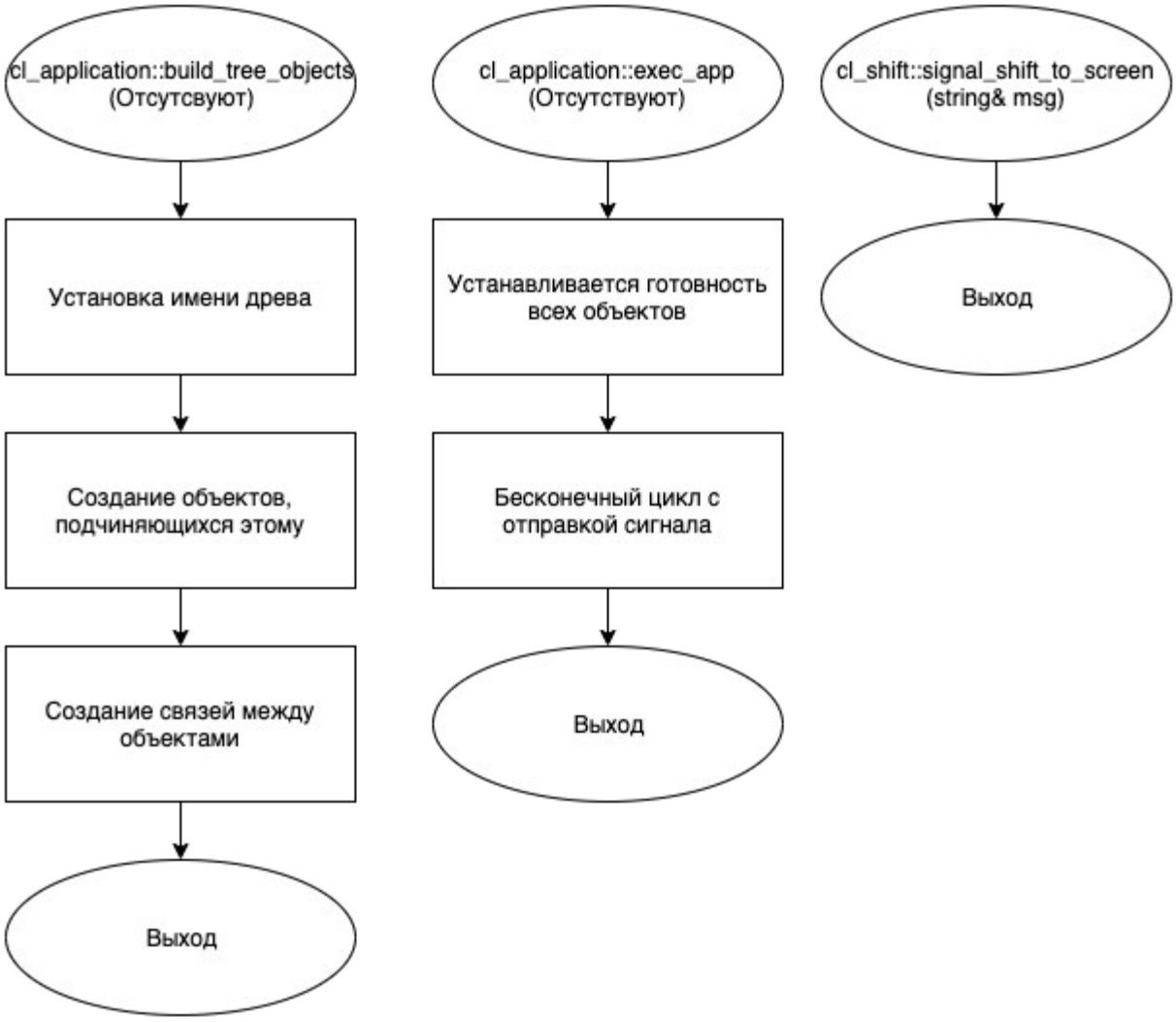

Рисунок 6 – Блок-схема алгоритма

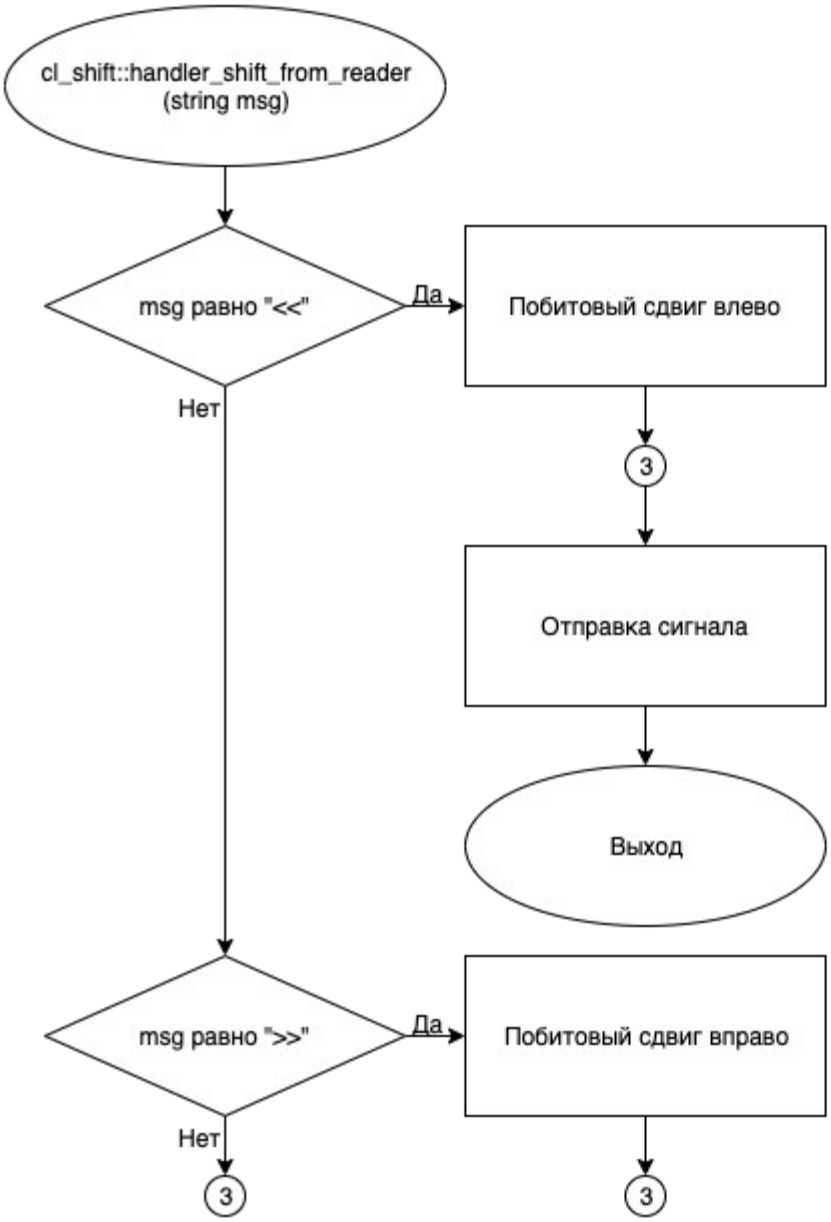

Рисунок 7 – Блок-схема алгоритма

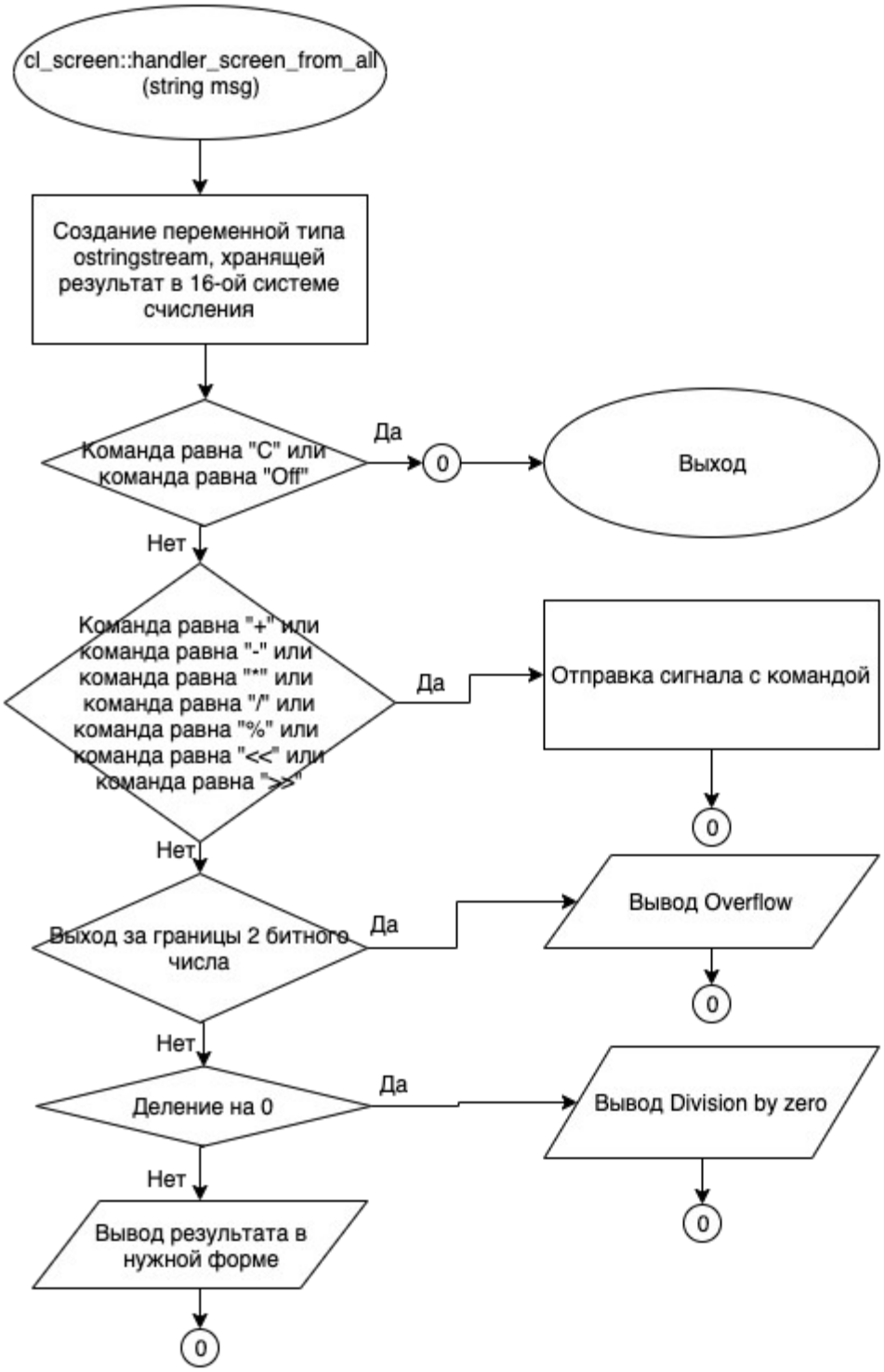

Рисунок 8 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_application.cpp

Листинг 1 – cl_application.cpp

```
#include "cl_application.h"
cl_application::cl_application(cl_base*
p_head_object):cl_base(p_head_object){}

void cl_application::build_tree_objects()
{
   cl_base* p_sub = this;

   set_name("System");

   p_sub = new cl_reader(this, "Reader");
   p_sub = new cl_calc(this, "Calc");
   p_sub = new cl_shift(this, "Shift");
   p_sub = new cl_cancel(this, "Cancel");
   p_sub = new cl_screen(this, "Screen");

   this->set_connection(SIGNAL_D(cl_application::signal_app_to_reader),
                get_object("Reader"),
                HANDLER_D(cl_reader::handler_reader_from_app));

   get_object("Reader")-
>set_connection(SIGNAL_D(cl_reader::signal_reader_to_all),
                                this,

   HANDLER_D(cl_application::handler_app_from_reader));

   get_object("Reader")-
>set_connection(SIGNAL_D(cl_reader::signal_reader_to_all),
                                get_object("Cancel"),

   HANDLER_D(cl_cancel::handler_cancel_from_reader));

   get_object("Reader")-
>set_connection(SIGNAL_D(cl_reader::signal_reader_to_all),
                                get_object("Shift"),

   HANDLER_D(cl_shift::handler_shift_from_reader));

   get_object("Reader")-
```

```
>set_connection(SIGNAL_D(cl_reader::signal_reader_to_all),
                                get_object("Calc"),

   HANDLER_D(cl_calc::handler_calc_from_reader));

   get_object("Calc")-
>set_connection(SIGNAL_D(cl_calc::signal_calc_to_screen),
get_object("Screen"),HANDLER_D(cl_screen::handler_screen_from_all));
}

void cl_application::signal_app_to_reader(std::string& msg){}

void cl_application::handler_app_from_reader(std::string msg)
{

   if(msg == "Off")
   {
      s_cmd = "Off";
      s_operand_2 = "Off";
   }

   if(msg == "SHOWTREE")
   {
      this->emit_signal(SIGNAL_D(cl_application::show_tree), "");
   }
}

int cl_application::exec_app()
{
   std::string s_msg;
   this->turn_on_subtree();

   while(s_cmd != "Off")
   {
      emit_signal(SIGNAL_D(cl_application::signal_app_to_reader), s_msg);
   }

   return 0;
}

void cl_application::show_tree(std::string& msg)
{
   this->tree_traversal();
   exit(0);
}
```

5.2 Файл cl_application.h

Листинг 2 – cl_application.h

```
#ifndef __CL_APPLICATION__H
```

```
#define __CL_APPLICATION__H
#include "cl_base.h"
#include "cl_reader.h"
#include "cl_calc.h"
#include "cl_cancel.h"
#include "cl_shift.h"
#include "cl_screen.h"

class cl_application: public cl_base
{
private:
   std::string s_cmd = "";
public:
   cl_application(cl_base*);

   void build_tree_objects();
   int exec_app();
   void signal_app_to_reader(std::string& msg);
   void handler_app_from_reader(std::string msg);
   void show_tree(std::string& msg);

};
#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base* p_head, std::string s_name){
   this->s_name = s_name;
   this->p_head_object = p_head;
   if(p_head_object != nullptr)
      p_head_object-> p_sub_objects.push_back(this);
}


cl_base::~cl_base()
{
   for(int i = 0; i < p_sub_objects.size(); i++)
   {
      delete p_sub_objects[i];
   }
}


cl_base* cl_base::get_head()
{
```

```
   return p_head_object;
}


std::string cl_base::get_name()
{
   return this-> s_name;
}


bool cl_base::set_name(std::string s_new_name)
{
   if(p_head_object != nullptr)
   {
      for(int i = 0; i < p_head_object -> p_sub_objects.size(); i++)
      {
         if(p_head_object -> p_sub_objects[i] -> get_name() == get_name())
         {
            return false;
         }
      }
   }
   this->s_name = s_new_name;
   return true;
}


cl_base* cl_base::get_object(std::string s_name)
{
   for(int i = 0;i < p_sub_objects.size(); i++)
   {
      if(p_sub_objects[i] -> get_name() == s_name)
      {
         return p_sub_objects[i];
      }
   }
   return nullptr;
}

void cl_base::show_object_tree(int index)
{
   std::cout << std::endl;
   for(int i = 0; i < index; i++)
   {
      std::cout <<"    ";
   }

   std::cout << this->get_name();

   for(int i = 0; i < p_sub_objects.size();i++)
   {
      p_sub_objects[i]->show_object_tree(index + 1);
   }
}
```

```
void cl_base::tree_traversal(int index)
{
   std::cout <<std::endl;
   for(int i = 0; i < index; i++)
   {
      std::cout <<"    ";
   }
   if(object_state == 0)
   {
      std::cout << this->get_name() << " is not ready";
   }
   else
   {
      std::cout << this->get_name() << " is ready";
   }

   for(int i = 0; i < p_sub_objects.size(); i ++)
   {
      p_sub_objects[i]->tree_traversal(index+1);
   }
}


void cl_base::state(int state)
{
   if(state != 0)
   {
      bool flag = true;
      cl_base* p_found = this;
      while(p_found-> p_head_object != nullptr)
      {
         p_found = p_found -> p_head_object;
         if(p_found ->object_state==0)
         {
            flag = false;
         }
      }
      if(flag == true)
      {
         object_state =  state;
      }
   }
   else
   {
      object_state = state;
      for(int i = 0; i < p_sub_objects.size(); i++)
         p_sub_objects[i] -> state(state);
   }
}


cl_base* cl_base::search_sub(std::string s_name)
{
   std::queue<cl_base*> q;
   cl_base* p_found = nullptr;
```

```
        q.push(this);
        while(!q.empty())
        {
            cl_base* p_front = q.front();
            q.pop();
            if(p_front -> get_name() == s_name)
            {
                if(p_found != nullptr)
                {
                    return nullptr;
                }
                else
                {
                    p_found = p_front;
                }
            }
            for(auto p_sub_object: p_front -> p_sub_objects)
            {
                q.push(p_sub_object);
            }
        }
        return p_found;
    }

    cl_base* cl_base::search_branch(std::string name)
    {
        if(get_head() == nullptr)
        {
            return search_sub(name);
        }
        else
        {
            return get_head() -> search_branch(name);
        }
    }

    void cl_base::delete_sub_object(std::string s_name)
    {
        std::vector<cl_base*> & sub = this-> p_sub_objects;
        for(int i = 0; i < sub.size(); i++)
        {
            if(sub[i] -> get_name() == s_name)
            {
                delete sub[i];
                sub.erase(sub.begin() + i);
                return;
            }
        }
    }

    bool cl_base::change_head(cl_base* p_head_object)
    {
        if(get_head() == p_head_object)
        {
            return true;
```

```
        }
        if(get_head() == nullptr || p_head_object == nullptr)
        {
           return false;
        }
        if(p_head_object->get_object(get_name()) != nullptr)
           return false;
        std::queue<cl_base*> search;
        search.push(this);
        while(!search.empty())
        {
           cl_base* check_element =  search.front();
           search.pop();
           if(check_element == p_head_object)
           {
              return false;
           }

           for(int i = 0; i < check_element->p_sub_objects.size(); i++)
           {
              search.push(check_element->p_sub_objects[i]);
           }
        }
        std::vector<cl_base*> & p_sub_objects_old = get_head()->p_sub_objects;
        for(int i = 0; i < p_sub_objects_old.size(); ++i)
        {
           if(p_sub_objects_old[i]->get_name() == get_name())
           {
              p_sub_objects_old.erase(p_sub_objects_old.begin() + i);
              p_head_object->p_sub_objects.push_back(this);
              return true;
           }
        }
        return false;
     }

     cl_base* cl_base::get_object_by_coordinate(std::string s_object_coordinate)
     {
        cl_base* base = this;
        while(base->get_head() != nullptr)
           base = base->get_head();
        if(s_object_coordinate.empty())
        {
           return nullptr;
        }
        if(s_object_coordinate == ".")
        {
           return this;
        }
        if(s_object_coordinate == "/")
        {
           return base;
        }
        if(s_object_coordinate.substr(0, 2) == "//")
        {
```

```
      return search_branch(s_object_coordinate.substr(2));
   }
   if(s_object_coordinate[0] == '.')
   {
      return search_sub(s_object_coordinate.substr(1));
   }
   if(s_object_coordinate[0] != '/')
   {
      size_t ind = s_object_coordinate.find('/');
      cl_base* sub_ptr = this->get_object(s_object_coordinate.substr(0,ind));
      if(sub_ptr == nullptr || ind == std:: string :: npos)
         return sub_ptr;
      return sub_ptr->get_object_by_coordinate(s_object_coordinate.substr(ind
+ 1));
   }
   return base->get_object_by_coordinate(s_object_coordinate.substr(1));
}

void   cl_base::set_connection(TYPE_SIGNAL   p_signal,   cl_base*   p_target,
TYPE_HANDLER handler)
{
   o_sh* p_value;
   for(auto connection : connects)
   {
      if(connection->p_signal == p_signal && connection->p_target == p_target
&& connection->handler == handler)
         return;
   }
   p_value = new o_sh();

   p_value->p_signal = p_signal;
   p_value->p_target = p_target;
   p_value->handler = handler;

   connects.push_back(p_value);
}

void  cl_base::delete_connection(TYPE_SIGNAL  p_signal,  cl_base*  p_target,
TYPE_HANDLER handler)
{
   for(int i = 0; i < connects.size(); i++)
   {
      if(connects[i]->p_signal  ==  p_signal  &&  connects[i]->p_target  ==
p_target && connects[i]->handler == handler)
      {
         delete connects[i];
         connects.erase(connects.begin() + i);
         return;
      }
   }
}

void cl_base::emit_signal(TYPE_SIGNAL p_signal, std::string s_message)
{
   if(this->object_state == 0)
```

```
        {
            return;
        }
        (this->*p_signal)(s_message);
        for(auto connection : connects)
        {
            if(connection->p_signal == p_signal)
            {
                cl_base* p_target = connection->p_target;
                TYPE_HANDLER handler = connection->handler;
                if(p_target->object_state != 0)
                    (p_target->*handler)(s_message);
            }
        }
    }

    std::string cl_base:: get_path()
    {
        std::string path;
        cl_base* current =  this;
        if(current->get_head() == nullptr)
        {
            return "/";
        }
        while(current->get_head() != nullptr)
        {
            path = "/" + current->get_name() + path;
            current = current->get_head();
        }
        return path;
    }

    int cl_base::get_class()
    {
        return 0;
    }

    void cl_base::set_state_branch(int new_object_state)
    {
        if(get_head() != nullptr && get_head() -> object_state == 0)
        {
            return;
        }
        state(new_object_state);
        for(int i = 0; i < p_sub_objects.size(); ++i)
        {
            p_sub_objects[i]->set_state_branch(new_object_state);
        }
    }

    void cl_base::turn_on_subtree()
    {
        object_state = 1;
        for(auto p_sub_object : p_sub_objects)
        {
```

```
            p_sub_object->turn_on_subtree();
        }
    }
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
    #ifndef __CL_BASE__H
    #define __CL_BASE__H

    #include <string>
    #include <vector>
    #include <iostream>
    #include <queue>
    #include <stack>
    #include <sstream>
    #include <algorithm>
    #include <iomanip>
    #include <bitset>

    #define SIGNAL_D(signal_f) (TYPE_SIGNAL) &signal_f
    #define HANDLER_D(handler_f) (TYPE_HANDLER) &handler_f

    class cl_base;

    typedef void(cl_base::*TYPE_SIGNAL) (std::string& msg);
    typedef void(cl_base::*TYPE_HANDLER) (std::string msg);

    struct o_sh
    {
       TYPE_SIGNAL p_signal;
       cl_base* p_target;
       TYPE_HANDLER handler;
    };

    class cl_base
    {
    public:

       cl_base(cl_base* p_head, std::string s_name = "Base_object"); // Вызов
    конструктора
       ~cl_base();// Вызов деструктора

       cl_base* get_head(); // метод для получения головного объекта
       cl_base* get_object( std::string s_name);// Получение подчиненных объектов
       cl_base* search_sub(std::string);
       cl_base* search_branch(std::string);
       cl_base*  get_object_by_coordinate(std::string  =  "/");//Метод  получения
    указателя на любой объект в составе дерева иерархии объектов согласно пути
```

```
(координаты).

   void delete_sub_object(std::string);//Метод удаления подчиненного объекта
по наименованию
   bool change_head(cl_base* new_head);//Метод переопределения головного
объекта для текущего в дереве иерархии
   bool set_name(std:: string s_new_name); // получение доступа к
редоктированию названий объектов
   void show_object_tree(int index = 0); // Вывод дерева иерархии
   void tree_traversal(int index = 0); // Вывод дереав иерархии объектов и
меток их готовности
   void state(int);//Метод установки готовности объекта
   std::string get_name(); // Получение названий объектов

   void set_connection(TYPE_SIGNAL p_signal, cl_base* p_target, TYPE_HANDLER
handler);//установления связи между сигналом текущего объекта и обработчиком
целевого объекта
   void delete_connection(TYPE_SIGNAL p_signal, cl_base* p_target,
TYPE_HANDLER handler);//удаления (разрыва) связи между сигналом текущего
объекта и обработчиком целевого объекта
   void emit_signal(TYPE_SIGNAL p_signal, std::string s_message);//Выдача
сигнала от текущего объекта с передачей строковой переменной.
   std::string get_path();//Возвращение абсолютного пути
   virtual int get_class();
   void set_state_branch(int);
   void turn_on_subtree();

   std::string s_expression = "", s_operation = "", s_operand_2 = "";
   int i_result = 0, f = 0;

private:

   std::string s_name;
   cl_base* p_head_object;
   std::vector<cl_base*> p_sub_objects;
   int object_state = 0;
   std::vector<o_sh*> connects;

};
#endif
```

5.5 Файл cl_calc.cpp

Листинг 5 – cl_calc.cpp

```
#include "cl_calc.h"

cl_calc::cl_calc(cl_base*         p_head_object,         std::         string
s_name):cl_base(p_head_object, s_name){}
```

```
void cl_calc::signal_calc_to_screen(std::string& msq){}

void cl_calc::handler_calc_from_reader(std::string msg)
{
   if(msg == "+")
   {
      get_head()->i_result   =   get_head()->i_result   +   atoi((get_head()-
>s_operand_2).c_str());
   }

   else if(msg == "-")
   {
      get_head()->i_result   =   get_head()->i_result   -   atoi((get_head()-
>s_operand_2).c_str());
   }

   else if(msg == "*")
   {
      get_head()->i_result   =   get_head()->i_result   *   atoi((get_head()-
>s_operand_2).c_str());
   }

   else if(msg == "/")
   {
      if(atoi((get_head()->s_operand_2).c_str()) != 0)
      {
         get_head()->i_result  =  get_head()->i_result  /  atoi((get_head()-
>s_operand_2).c_str());
      }
      else
      {
         get_head()->s_operand_2 = "     Division by zero";
         get_head()->i_result = 0;
      }
   }

   else if(msg == "%")
   {
      if(atoi((get_head()->s_operand_2).c_str()) != 0)
      {
         get_head()->i_result  =  get_head()->i_result  %  atoi((get_head()-
>s_operand_2).c_str());
      }

      else
      {
         get_head()->s_operand_2 = "     Division by zero";
         get_head()->i_result = 0;
      }
   }

   emit_signal(SIGNAL_D(cl_calc::signal_calc_to_screen),
std::to_string(get_head()->i_result));
}
```

5.6 Файл cl_calc.h

Листинг 6 – cl_calc.h

```
#ifndef __CL_CALC__H
#define __CL_CALC__H
#include "cl_base.h"

class cl_calc : public cl_base
{
public:
   cl_calc(cl_base* p_head_object, std::string s_name);

   void signal_calc_to_screen(std::string& msg);
   void handler_calc_from_reader(std::string msg);
};

#endif
```

5.7 Файл cl_cancel.cpp

Листинг 7 – cl_cancel.cpp

```
#include "cl_cancel.h"

cl_cancel::cl_cancel(cl_base*        p_head_object,        std::        string
s_name):cl_base(p_head_object, s_name){}

void cl_cancel::handler_cancel_from_reader(std::string msg)
{
   if(msg == "C")
   {
      get_head()->s_expression = "0";
      get_head()->s_operation = "";
      get_head()->s_operand_2 = "C";
      get_head()->i_result = 0;
   }
}
```

5.8 Файл cl_cancel.h

Листинг 8 – cl_cancel.h

```
#ifndef __CL_CANCEL__H
```

```
#define __CL_CANCEL__H
#include "cl_base.h"

class cl_cancel : public cl_base
{
public:
   cl_cancel(cl_base* p_head_object, std::string s_name);

   void handler_cancel_from_reader(std::string msg);
};
#endif
```

5.9 Файл cl_reader.cpp

Листинг 9 – cl_reader.cpp

```
#include "cl_reader.h"

cl_reader::cl_reader(cl_base*       p_head_object,       std::       string
s_name):cl_base(p_head_object, s_name){}

void cl_reader::handler_reader_from_app(std::string msg)
{
   std::string s_cmd;
   std::cin >> s_cmd;

   if(s_cmd == "C" || s_cmd == "Off" || s_cmd == "SHOWTREE")
   {
      emit_signal(SIGNAL_D(cl_reader::signal_reader_to_all), s_cmd);
   }

   else if(s_cmd == "+" || s_cmd == "-" || s_cmd == "*" || s_cmd == "/" ||
s_cmd == "%" ||
         s_cmd == "<<" || s_cmd == ">>")
   {
      get_head()->s_operation = s_cmd;
      get_head()->s_expression += (" " + s_cmd);

      std::cin >> get_head()->s_operand_2;
      get_head()->s_expression += (" " + get_head()->s_operand_2);
      emit_signal(SIGNAL_D(cl_reader::signal_reader_to_all), s_cmd);
   }

   else
   {
      get_head()->s_expression = s_cmd;
      get_head()->i_result = atoi((get_head()->s_expression).c_str());
   }
}
```

```
void cl_reader::signal_reader_to_all(std::string& msg){}
```

5.10 Файл cl_reader.h

Листинг 10 – cl_reader.h

```
#ifndef __CL_READER__H
#define __CL_READER__H

#include "cl_base.h"

class cl_reader : public cl_base
{
public:
   cl_reader(cl_base* p_head_object, std::string s_name);

   void handler_reader_from_app(std::string msg);
   void signal_reader_to_all(std::string& msg);
};
#endif
```

5.11 Файл cl_screen.cpp

Листинг 11 – cl_screen.cpp

```
#include "cl_screen.h"
cl_screen::cl_screen(cl_base*       p_head_object,       std::       string
s_name):cl_base(p_head_object, s_name){}

void cl_screen::toBinary(unsigned int i)
{
   std::bitset<16> binary(i);
   std::string binaryString = binary.to_string();
   for(int i = 12; i >= 0; i -=4)
   {
      binaryString.insert(i, " ");
   }
   std::cout << binaryString;
}

void cl_screen::handler_screen_from_all(std::string msg)
{
   std::ostringstream s;
   s << std::setfill('0') << std::setw(4) << std::hex << std::uppercase <<
```

```
      (unsigned short)get_head()->i_result;

         if(get_head()->s_operand_2 == "C" || get_head()->s_operand_2 == "Off")
         {
            get_head()->s_operand_2 = "";
         }

         else if(get_head()->i_result > 32767 || get_head()->i_result < -32768)
         {
            std::cout << std::endl << get_head()->s_expression << "     Overflow";
            get_head()->s_expression = "0";
            get_head()->i_result = 0;
         }

         else if(get_head()->s_operand_2 == "     Division by zero")
         {
            std::cout<<std::endl << get_head()->s_expression << "      Division by
      zero";
            get_head()->s_expression = "0";
         }

         else if(get_head()->s_operand_2 != "     Division by zero")
         {
            if(f != 0)
            {
               std::cout << std::endl;
            }

            std::cout << get_head()->s_expression << "     HEX " << s.str();
            std::cout << "  DEC " << get_head()->i_result;
            std::cout << "  BIN";
            toBinary(get_head()->i_result);
            f++;
         }
      }
```

5.12 Файл cl_screen.h

Листинг 12 – cl_screen.h

```
      #ifndef __CL_SCREEN__H
      #define __CL_SCREEN__H
      #include "cl_base.h"

      class cl_screen : public cl_base
      {
      public:
         cl_screen(cl_base* p_head_object, std::string s_name);

         void handler_screen_from_all(std::string msg);
```

```
   void toBinary(unsigned int i);
};
#endif
```

5.13 Файл cl_shift.cpp

Листинг 13 – cl_shift.cpp

```
#include "cl_shift.h"

cl_shift::cl_shift(cl_base*        p_head_object,        std::        string
s_name):cl_base(p_head_object, s_name){}

void cl_shift::signal_shift_to_screen(std::string& msg){}

void cl_shift::handler_shift_from_reader(std::string msg)
{
   if(msg == "<<")
   {
      get_head()->i_result   =   get_head()->i_result   <<   atoi((get_head()-
>s_operand_2).c_str());
   }

   else if(msg == ">>")
   {
      get_head()->i_result   =   get_head()->i_result   >>   atoi((get_head()-
>s_operand_2).c_str());
   }

   emit_signal(SIGNAL_D(cl_shift::signal_shift_to_screen),
std::to_string(get_head()->i_result));
}
```

5.14 Файл cl_shift.h

Листинг 14 – cl_shift.h

```
#ifndef __CL_SHIFT__H
#define __CL_SHIFT__H
#include "cl_base.h"

class cl_shift : public cl_base
{
public:
   cl_shift(cl_base* p_head_object, std::string s_name);
```

```
   void handler_shift_from_reader(std::string msg);
   void signal_shift_to_screen(std::string& msg);
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>

#include "cl_application.h"

int main()
{
   cl_application  ob_cl_application(nullptr); // создание корневого объекта
   ob_cl_application.build_tree_objects();   //   конструирование   системы,
построение дерева объектов
   return ob_cl_application.exec_app(); // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 17.

Таблица 17 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
5 + 5 << 1 / 0 + 5 C 7 8 / -3 C 9 % -4 + 7 * 11 Off	5 + 5 HEX 000A DEC 10 BIN 0000 0000 0000 1010 5 + 5 << 1 HEX 0014 DEC 20 BIN 0000 0000 0001 0100 5 + 5 << 1 / 0 Division by zero 0 + 5 HEX 0005 DEC 5 BIN 0000 0000 0000 0101 8 / -3 HEX FFFE DEC -2 BIN 1111 1111 1111 1110 9 % -4 HEX 0001 DEC 1 BIN 0000 0000 0000 0001 9 % -4 + 7 HEX 0008 DEC 8 BIN 0000 0000 0000 1000 9 % -4 + 7 * 11 HEX 0058 DEC 88 BIN 0000 0000 0101 1000	5 + 5 HEX 000A DEC 10 BIN 0000 0000 0000 1010 5 + 5 << 1 HEX 0014 DEC 20 BIN 0000 0000 0001 0100 5 + 5 << 1 / 0 Division by zero 0 + 5 HEX 0005 DEC 5 BIN 0000 0000 0000 0101 8 / -3 HEX FFFE DEC -2 BIN 1111 1111 1111 1110 9 % -4 HEX 0001 DEC 1 BIN 0000 0000 0000 0001 9 % -4 + 7 HEX 0008 DEC 8 BIN 0000 0000 0000 1000 9 % -4 + 7 * 11 HEX 0058 DEC 88 BIN 0000 0000 0101 1000
5 + 5 << 1 / 0 + 5 C 7 8 / -3 C 9 % -4 + 7 * 11 SHOWTREE Off	5 + 5 HEX 000A DEC 10 BIN 0000 0000 0000 1010 5 + 5 << 1 HEX 0014 DEC 20 BIN 0000 0000 0001 0100 5 + 5 << 1 / 0 Division by zero 0 + 5 HEX 0005 DEC 5 BIN 0000 0000 0000 0101 8 / -3 HEX FFFE DEC -2 BIN 1111 1111 1111 1110 9 % -4 HEX 0001 DEC 1 BIN 0000 0000 0000 0001 9 % -4 + 7 HEX 0008 DEC 8 BIN 0000 0000 0000 1000	5 + 5 HEX 000A DEC 10 BIN 0000 0000 0000 1010 5 + 5 << 1 HEX 0014 DEC 20 BIN 0000 0000 0001 0100 5 + 5 << 1 / 0 Division by zero 0 + 5 HEX 0005 DEC 5 BIN 0000 0000 0000 0101 8 / -3 HEX FFFE DEC -2 BIN 1111 1111 1111 1110 9 % -4 HEX 0001 DEC 1 BIN 0000 0000 0000 0001 9 % -4 + 7 HEX 0008 DEC 8 BIN

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	9 % -4 + 7 * 11 HEX 0058 DEC 88 BIN 0000 0000 0101 1000 System is ready Reader is ready Calc is ready Shift is ready Cancel is ready Screen is ready	0000 0000 0000 1000 9 % -4 + 7 * 11 HEX 0058 DEC 88 BIN 0000 0000 0101 1000 System is ready Reader is ready Calc is ready Shift is ready Cancel is ready Screen is ready

ЗАКЛЮЧЕНИЕ

Изучение курса "Объектно-ориентированное программирование" позволило мне глубже понять методологию объектно-ориентированного подхода. Я освоил концепцию классов и объектов и их применение для моделирования систем. Я также приобрел понимание таких ключевых компонентов парадигмы ООП, как наследование, полиморфизм и инкапсуляция. Обучение проходило на языке программирования C++, что позволило мне изучить такие понятия, как указатели, ссылки, встроенные и дружественные функции, дружественные классы, виртуальные методы, статические методы, абстрактные классы, перегрузка и переопределение функций, контейнеры и структуры, а также шаблоны функций и классов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

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).