Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Для организации иерархического построения объектов необходимо разработать базовый класс, который содержит функционал и свойства для построения иерархии объектов.

В последующем, в приложениях использовать этот класс как базовый для всех создаваемых классов. Это позволит включать любой объект в состав дерева иерархии объектов.

Создать базовый класс со следующими элементами:

Свойства:

- наименование объекта (строкового типа);
- указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно 0);
- массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.

Функционал:

- параметризированный конструктор с параметрами: указатель на головной объект в дереве иерархии и наименование объекта (имеет значение по умолчанию);
- метод определения имени объекта;
- метод получения имени объекта;
- метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- метод переопределения головного объекта для текущего в дереве иерархии (не допускается задание головного объекта для корневого и появление второго корневого объекта);

- метод получения указателя на головной объект текущего объекта.

Для построения дерева иерархии объектов в качестве корневого объекта используется объект приложение. Класс объекта приложения наследуется от базового класса. Объект приложение реализует следующий функционал:

− метод построения исходного дерева иерархии объектов (конструирования программы-системы, изделия);

− метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Написать программу, которая последовательно строит дерево иерархии объектов, слева направо и сверху вниз.

Переход на новый уровень происходит только от правого (последнего) объекта предыдущего уровня.

Для построения дерева использовать объекты двух производных классов, наследуемых от базового. Каждый объект имеет уникальное имя.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

```
int main()
{
   cl_application  ob_cl_application ( nullptr );
   ob_cl_application.bild_tree_objects ( );  // построение дерева объектов
   return ob_cl_application.exec_app ( );    // запуск системы
}
```

Наименование класса cl_application и идентификатора корневого объекта ob_cl_application могут быть изменены разработчиком.

1.1 Описание входных данных

Первая строка:

«имя корневого объекта»

Вторая строка и последующие строки:

«имя головного объекта» «имя подчиненного объекта»

Создается подчиненный объект и добавляется в иерархическое дерево.

Если «имя головного объекта» равняется «имени подчиненного объекта», то новый объект не создается и построение дерева объектов завершается.

Примерввода

Object_root

Object_root Object_1

Object_root Object_2

Object_root Object_3

Object_3 Object_4

Object_3 Object_5

Object_6 Object_6

Дерево объектов, которое будет построено по данному примеру:

Object_root

    Object_1

    Object_2

Object_3

Object_4

Object_5

1.2 Описание выходных данных

Первая строка:

«имя корневого объекта»

Вторая строка и последующие строки имена головного и подчиненных объектов очередного уровня разделенных двумя пробелами.

«имя головного объекта» «имя подчиненного объекта»[[«имя подчиненного объекта»] …….]

Пример вывода

Object_root

Object_root Object_1 Object_2 Object_3

Object_3 Object_4 Object_5

2 МЕТОД РЕШЕНИЯ

Для построение дерева объекта используем:

Объект стандартного потока ввода с клавиатуры cin

Объект стандартного потока вывода на экран cout

Объекты object класса object(имена и количество которых вводится пользоваетелем)

Объект ob_cl_application класса cl_application

Класс cl_application:

- Свойства/поля:
- Методы:
  - Метод bild_tree_objects

1. Функционал - метод построения исходного дерева иерахии объектов (коструирования программы - системы, изделия);

  - Метод exec_app

1. Функционал - метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Класс cl_base:

Свойства/поля:

Поля отвечающий за наименование объекта:

Наименование - s_object_name;

Тип - строковой;

Модификатор доступа - закрытый;

Поля отвечающий за указатель на головной объект для текущего объекта;

Наименование - p_head_object;

Тип - указательный на базовый класс;

Модификатор доступа - закрытый;

Поля отвечающий за массив казателей на объкты, подчиненные к текущему объекту в дереве иерархии:

Наименование - subordinate_objects;

Тип - контейнер типа vector с указателем на базовый класс;

Модификатор доступа - закрытый;

Методы:

Конструктор

Функционал - параметризированный конструктор с параметрами: указатель на головной объект в дереве иерархии и наименование объекта (имеет значение по умолчанию);

Деструктор

Функционал - удаление дерево целиком;

Меотод set_object_name:

Функционал - метод определения имени объекта;

Метод get_object_name:

Функционал - метод получения имени объекта;

Метод change_head_object:

Функционал - метод переопределения головного объекта для текущего в дереве иерархии;

Метод get_head_object:

Функционал - получения указателя на головной объект текущего объекта;

Метод print:

Функционал - метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз

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 функции main

Функционал: Точка входа в программу, реализация вызова основных методов и инициализация объектов.

Параметры: нет.

Возвращаемое значение: Целое число - индикатор успешного заверщения программы.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создаем объект ob_cl_application класса cl_application посредством параметризированного конструктора с параметром nullptr	2
2		Вызов метода beel_tree_objects объекта ob_cl_application	3
3		Вызов метода exec_app объекта ob_cl_application	4
4		Возвращается результата работы метода exec_app	∅

3.2 Алгоритм деструктора класса cl_base

Функционал: деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 2.

Таблица 2 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1	i меньше размера списка	Удаляем объект из списка по индексу	2
			∅
2		Увеличиваем i на 1	1

3.3 Алгоритм метода change_head_object класса cl_base

Функционал: Переопределения головного объекта для текущего объекта в дереве иерархии.

Параметры: base* p_head_object - указатель на головной объект.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода change_head_object класса cl_base

№	Предикат	Действия	№ перехода
1	Головной объекта не нулевой и у текущего объекта значения указателя на родителя не нулевое	Объявлеяем указатель на базовый класс и присваиваем к нему указатель родительского объекта	2
			∅
2		У головного объекта берем список и добавляем текущий объект в конец	3
3		присваиваем указатель на родительский объект у текущего объекта	4
4	i меньше размера списка головного объекта		5
			∅
5	Текущий объект равен объекту из списка головного	Удаляем объект из контейнера подчиненных старого головного	6

№	Предикат	Действия	№ перехода
	объекта по индексу i		
		Увеличиваем i на 1	4
6			∅

3.4 Алгоритм метода set_object_name класса cl_base

Функционал: Определение имени объекта.

Параметры: string s_object_name - наименование объекта.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_object_name класса cl_base

№	Предикат	Действия	№ перехода
1		Присваеиваем к имени у текущего объекта имя переданного параметра	∅

3.5 Алгоритм метода get_head_object класса cl_base

Функционал: Получения указателя на головной объект текущего объекта.

Параметры: нет.

Возвращаемое значение: base* , головной объект.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_head_object класса cl_base

№	Предикат	Действия	№ перехода
1		Возвращаем указатель на головной объект	∅

3.6 Алгоритм метода get_object_name класса cl_base

Функционал: получение имени объекта.

Параметры: Нет.

Возвращаемое значение: string - имя объекта.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_object_name класса cl_base

№	Предикат	Действия	№ перехода
1		Возвращаем имя объекта	∅

3.7 Алгоритм метода print класса cl_base

Функционал: Вывод в консоль дерево объекта.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода print класса cl_base

№	Предикат	Действия	№ перехода
1	Размер списка головного объекта неравен нулю	Вывод имя текущего объекта	2
			5
2	i меньше размера списка головного объекта	Выводим из списка [i] - объекь	3
			4
3		Увеличиваем i на 1	2
4		Выводим на экран сивмол переноса строки	5
5	i меньше размера списка головного объекта	Вызываем метод print для объекта из списка подчиненных	6
			∅
6		Увеличиваем i на 1	5

3.8 Алгоритм метода bild_tree_objects класса cl_application

Функционал: Метод построения исходного дерева иерархии объектов(конструирования программы-системы, изделия).

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода bild_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Создаем переменную строкового типа name_root_object	2
2		Вводим с клавиатуры name_root_object	3
3		Вызываем метод set_object_name для текущего объекта, с переданным параметром name_root_object	4
4		Объявляем указатель на базовый класс temp_parent и присваиваем его к текущему объекту	5
5		Объявляем указатель на базовый класс temp_child и присваиваем его к нулевому указателю	6
6		Создаем переменную строкового типа name_parent	7
7		Создаем переменную строкового типа name_child	8
8		Вводим с клавиатуры две переменных name_parent и name_child	9
9	name_parent равно name_child	Остановить цикл	∅
			10
10	Имя главного текущего объекта равно имени временного объекта	Создаем объект посредством вызова параметризированного конструктора и передача переменных (temp_parent, name_child) и присваиваем к указателю (temp_child) указатель на	9

№	Предикат	Действия	№ перехода
		новый объект класса (object)	
		Присваиваем к указателю temp_parent указатель temp_child	11
11		Создаем объект посредством вызова параметризированного конструктора и передача переменных (temp_parent, name_child) и присваиваем к указателю (temp_child) указатель на новый объект класса (object)	9

3.9 Алгоритм метода exec_app класса cl_application

Функционал: Метод запуска приложения (начало функционирования системы, выполнение алгоритма решения задачи).

Параметры: нет.

Возвращаемое значение: ноль или код ошибки.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вызов метода get_object_name	2
2		Вывод результата работы метода get_object_name	3
3		Вызов метода print	4
4		Возвращаем ноль	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-10.

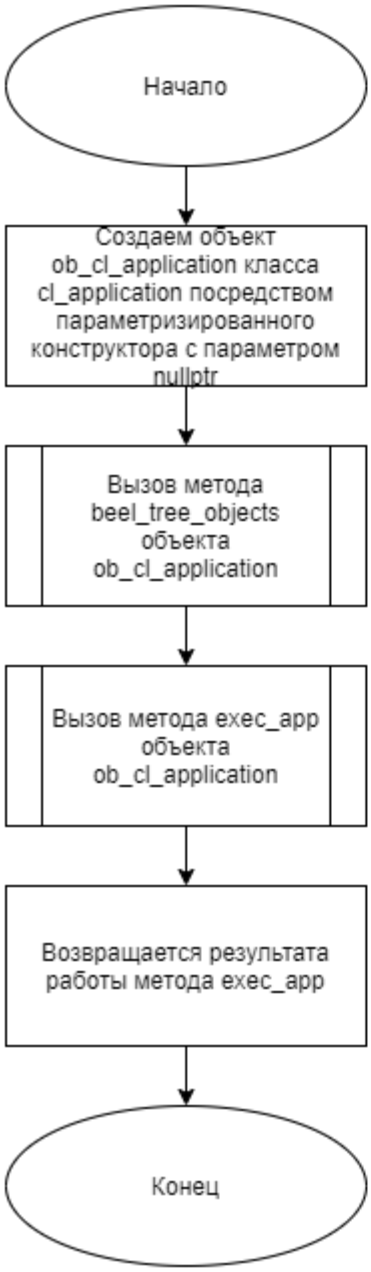

Рисунок 1 – Блок-схема алгоритма

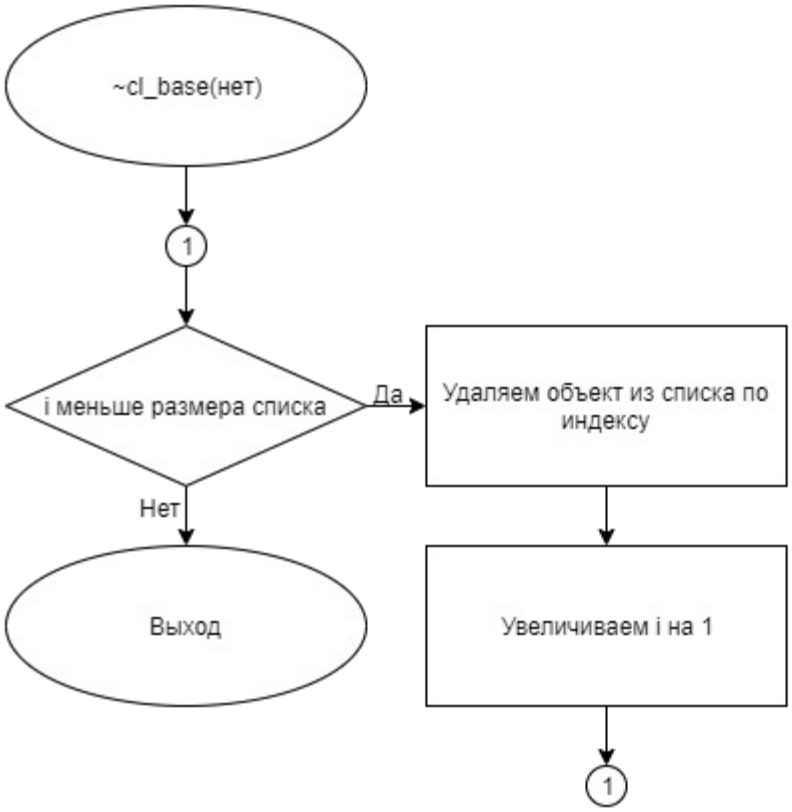

Рисунок 2 – Блок-схема алгоритма

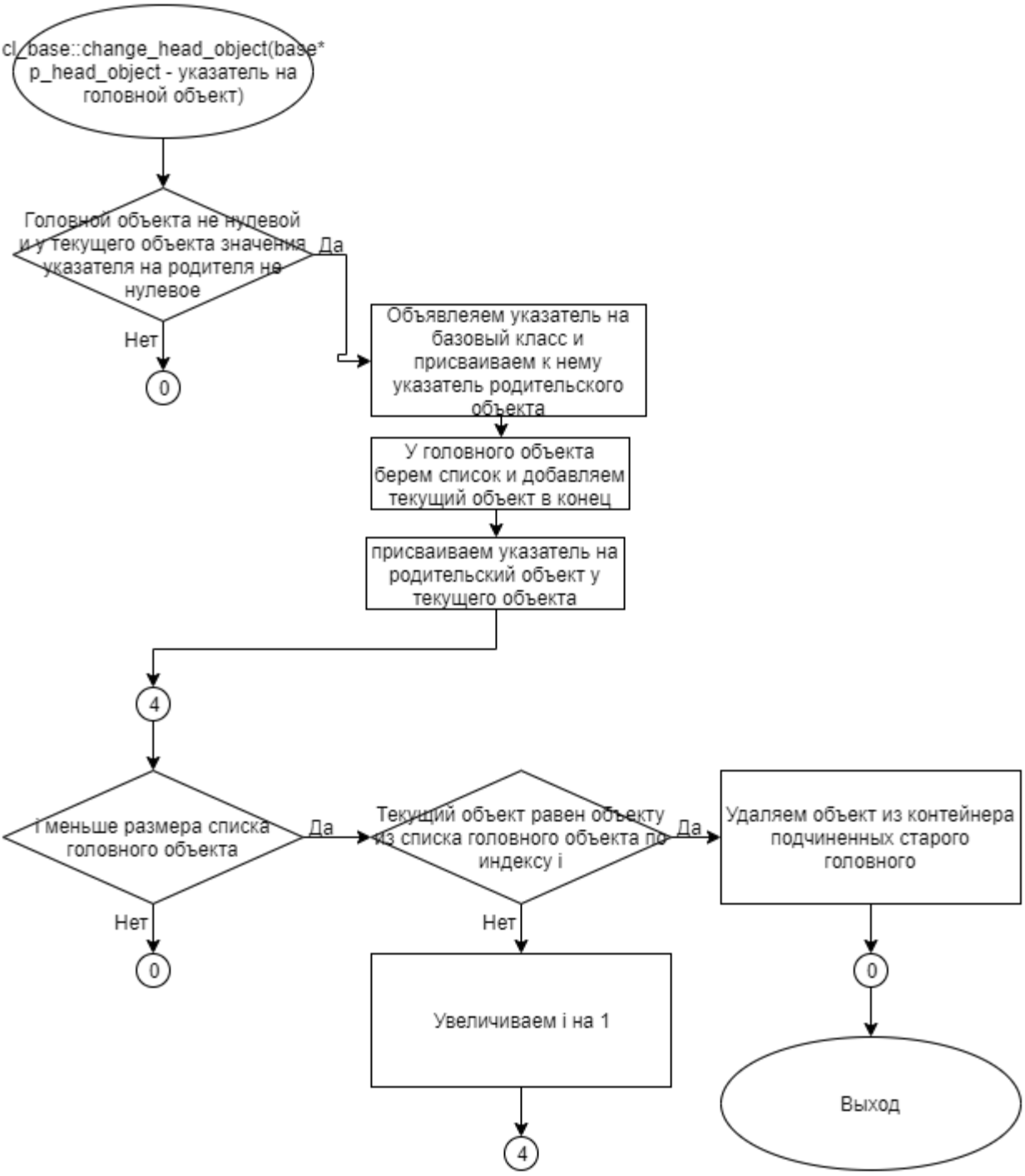

Рисунок 3 – Блок-схема алгоритма

Рисунок 4 – Блок-схема алгоритма

Рисунок 5 – Блок-схема алгоритма

Рисунок 6 – Блок-схема алгоритма

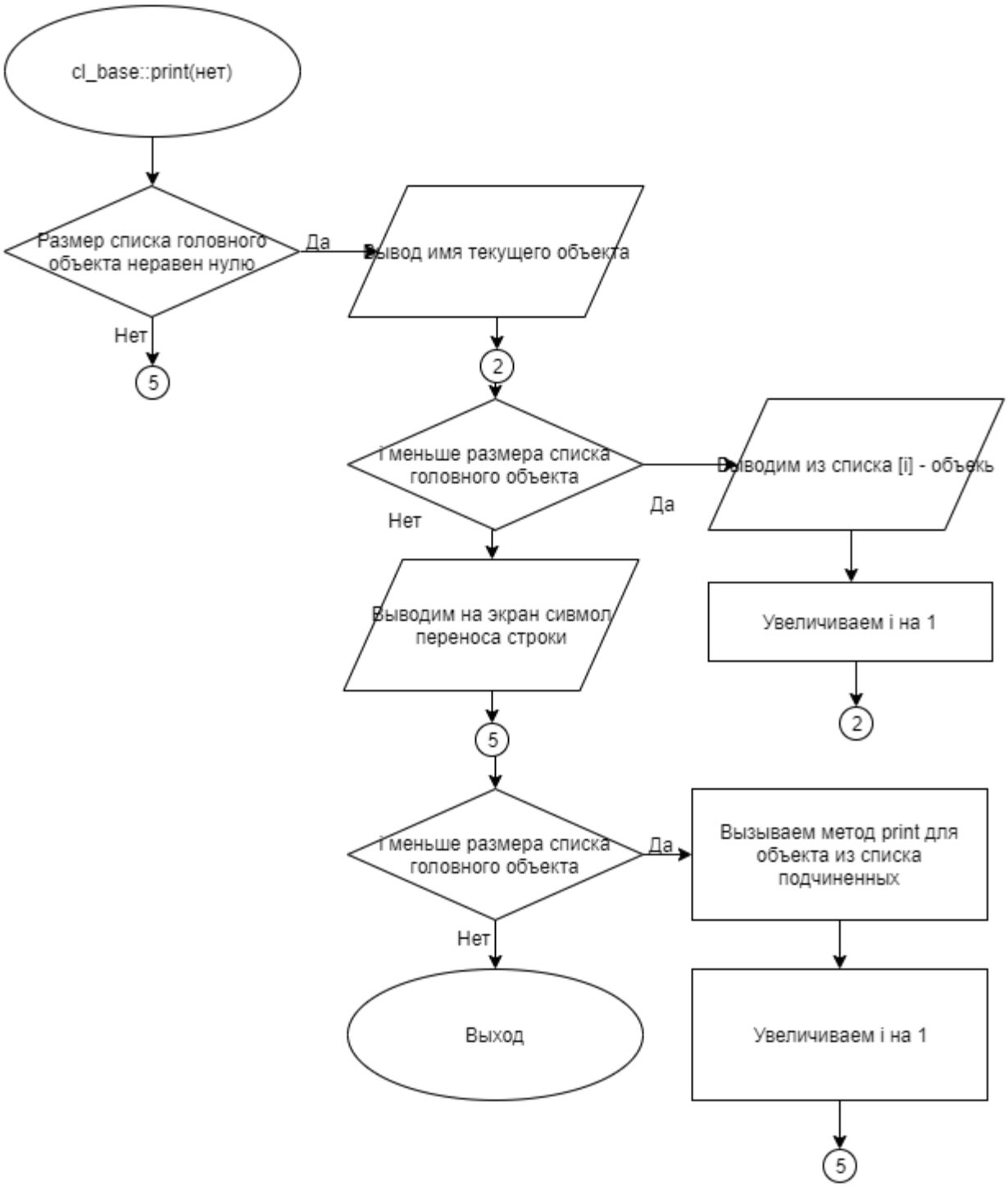

Рисунок 7 – Блок-схема алгоритма

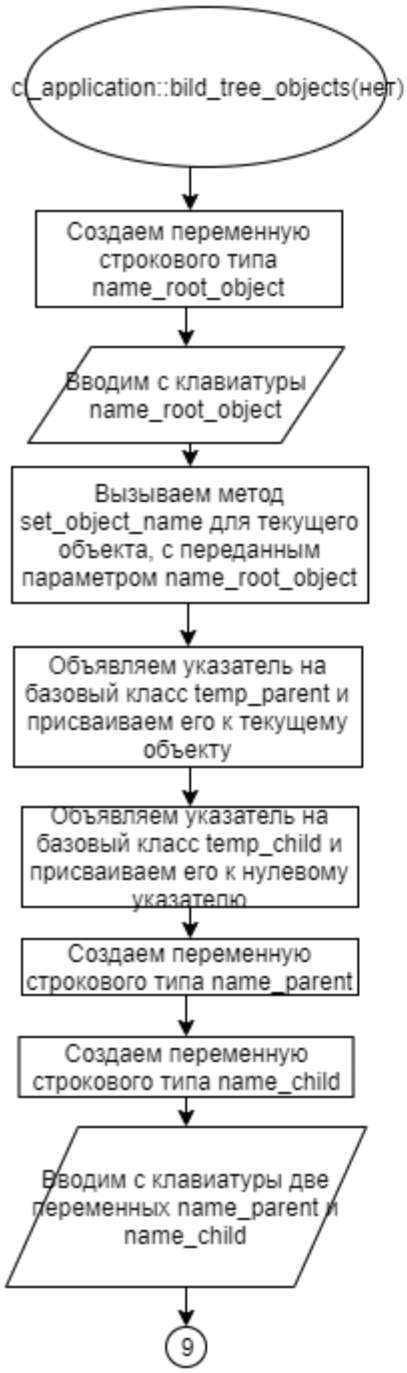

Рисунок 8 – Блок-схема алгоритма

Рисунок 9 – Блок-схема алгоритма

Рисунок 10 – 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"
#include "cl_base.h"

cl_application::cl_application(cl_base* p_head_object, string s_object_name):
cl_base(p_head_object, s_object_name)
{

}

void cl_application::bild_tree_objects()
{
            string name_root_object;
            cin >> name_root_object;
            this->set_object_name(name_root_object);
            cl_base* temp_parent = this;
            cl_base* temp_child = nullptr;
            string name_parent;
            string name_child;
            while (true)
            {
                        cin>> name_parent >> name_child;
                        if(name_parent == name_child)
                        {
                                    break;
                        }

                        if(name_parent == temp_parent ->get_object_name())
                        {
                                    temp_child    =    new    object(temp_parent,
name_child);
                        }
                        else
                        {
                                    temp_parent = temp_child;
                                    temp_child    =    new    object(temp_parent,
name_child);
                        }
            }
}

int cl_application::exec_app()
```

```
{
            cout << get_object_name();//<< endl;
            print();
            return 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 "object.h"
class cl_application:public cl_base
{
private:

public:
            cl_application(cl_base*   p_head_obkect,   string   s_object_name   =
"Base_objaect");
            void bild_tree_objects();
            int exec_app();
};
#endif
```

5.3 Файл cl_base.cpp

Листинг 3 – cl_base.cpp

```
#include "cl_base.h"
cl_base::cl_base(cl_base * p_head_object, string s_object_name)
{
            this -> s_object_name = s_object_name;
            this -> p_head_object = p_head_object;
            if (p_head_object != nullptr)
            {
                        p_head_object -> subordinate_objects.push_back(this);
            }

}

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

```
void cl_base::change_head_object(cl_base * p_head_object)
{
            if(p_head_object != nullptr && this ->p_head_object != nullptr)
            {
                        cl_base* temp = this->p_head_object;
                        p_head_object->subordinate_objects.push_back(this);
                        this->p_head_object = p_head_object;

                        for(int i=0; i< temp->subordinate_objects.size();i++)
                              {
                                    if(this == temp->subordinate_objects[i])
                                          {
                                                temp-
>subordinate_objects.erase(temp->subordinate_objects.begin()+i);


      break;
                                    }
                        }
            }
}

void cl_base::set_object_name(std::string s_object_name)
{
            this->s_object_name = s_object_name;
}

// cl_base * -это указатель на базовый класс
cl_base * cl_base::get_head_object()
{
            return p_head_object;
}

string cl_base::get_object_name()
{
      return s_object_name;
}

void cl_base::print()
{
            if(this->subordinate_objects.size()!=0)
            {

                        cout<<endl<<this->s_object_name;
                        for(int i = 0; i < this->subordinate_objects.size();i++)
                        {
                                    cout<<"     "<<   this->subordinate_objects[i]-
>s_object_name;
                        }
                                    //cout<<endl;
            }
            for(int i = 0; i < this-> subordinate_objects.size();i++)
            {
                        this->subordinate_objects[i]->print();
            }
}
```

5.4 Файл cl_base.h

Листинг 4 – cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H
#include <iostream>
#include <vector>
#include <string>
using namespace std;
class cl_base
{
private:
            string s_object_name;//наименование объекта
            cl_base* p_head_object;//указатель на головной объект
            vector < cl_base * > subordinate_objects;//указатели на подчиненные
объекты
public:
            cl_base(cl_base    *    p_head_object,    string    s_object_name    =
"Base_object");
            ~cl_base();
            void set_object_name(string s_object_name);//определение наименования
объекта
            string get_object_name();//наименование объекта
            void change_head_object( cl_base*);//переопределение головного объекта
            cl_base * get_head_object();//указатель на головной объект
            void print();
};
#endif
```

5.5 Файл main.cpp

Листинг 5 – main.cpp

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

int main()
{
      cl_application ob_cl_application(nullptr);
      ob_cl_application.bild_tree_objects(); // построение дерева объектов
      return ob_cl_application.exec_app(); // запуск системы
}
```

5.6 Файл object.cpp

Листинг 6 – object.cpp

```
#include "object.h"
object::object(cl_base* p_head_object, string
s_object_name):cl_base(p_head_object,s_object_name)
{

}
```

5.7 Файл object.h

Листинг 7 – object.h

```
#ifndef OBJECT_H
#define OBJECT_H
#include "cl_base.h"
class object:public cl_base
{
private:

public:
object(cl_base* p_head_object, string s_object_name);
};
#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 10.

Таблица 10 – Результат тестирования программы

Входные данные	**Ожидаемые выходные данные**	**Фактические выходные данные**
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_3 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5

СПИСОК ИСПОЛЬЗОВАННЫХ 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://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. 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).

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).