Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Поиск головного объекта выполняется от последнего созданного объекта. Первоначально последним созданным объектом считается корневой объект. Если для головного объекта обнаруживается дубляж имени в непосредственно подчиненных объектах, то объект не создается. Если обнаруживается дубляж имени на дереве иерархии объектов, то объект не создается. Если номер класса объекта задан некорректно, то объект не создается.

Вывод иерархического дерева объектов на консоль.

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:

1. метод поиска объекта на ветке дереве иерархии от текущего по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на искомой ветке дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
2. метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
3. метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
4. метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;
5. метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется. При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:

6. Построить дерево объектов системы (в методе корневого объекта построения исходного дерева объектов).
7. В методе корневого объекта запуска моделируемой системы реализовать:

7.1. Вывод на консоль иерархического дерева объектов в следующем виде:

```
root
    ob_1
        ob_2
    ob_3
        ob_4
            ob_5
        ob_6
            ob_7
```

где: root - наименование корневого объекта (приложения).

7.2. Переключение готовности объектов согласно входным данным (командам).

7.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:

```
root  is ready
    ob_1  is ready
        ob_2  is ready
    ob_3  is ready
        ob_4 is not ready
            ob_5 is not ready
        ob_6 is ready
            ob_7 is not ready
```

1.1 Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Последовательность ввода организовано так, что головной объект для очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка:

```
«Наименование корневого объекта»
```

Со второй строки:

```
«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта»
. . . . .
endtree
```

Со следующей строки вводятся команды включения или отключения объектов

```
«Наименование объекта» «Номер состояния объекта»
```

Пример ввода:

```
app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
```

```
object_02 object_05 5
object_01 object_07 2
endtree
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
```

1.2 Описание выходных данных

Вывести иерархию объектов в следующем виде:

```
Object tree
«Наименование корневого объекта»
    «Наименование объекта 1»
        «Наименование объекта 2»
    «Наименование объекта 3»
. . . . .
The tree of objects and their readiness
«Наименование корневого объекта» «Отметка готовности»
    «Наименование объекта 1» «Отметка готовности»
        «Наименование объекта 2» «Отметка готовности»
    «Наименование объекта 3» «Отметка готовности»
. . . . .
«Отметка готовности» - равно «is ready» или «is not ready»
```

Отступ каждого уровня иерархии 4 позиции.

Пример вывода:

```
Object tree
app_root
    object_01
        object_07
    object_02
        object_04
        object_05
The tree of objects and their readiness
app_root is ready
    object_01 is ready
        object_07 is not ready
    object_02 is ready
        object_04 is ready
        object_05 is not ready
```

2 МЕТОД РЕШЕНИЯ

Для решения поставленной задачи используются:

8. Условный оператор if
9. Операторы цикла for и while
10. Объект потока стандартного воода/вывода cin/cout
11. Контейнер std::vector
12. Класс Object_base:
13. Класс Object_2, унаследованный от класса Object_base с модификатором public:
    13.1. Параметризированный конструктор Object_2(Object_base* p_head_object,string s_name="Base_object"). Функция - создает объект класса, в качестве первого аргумента принимает указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии, в качестве второго объекта принимает наименование создаваемого объекта (имеет значение по умолчанию).
14. Класс Object_3, унаследованный от класса Object_base с модификатором public:
    14.1. Параметризированный конструктор Object_3(Object_base* p_head_object,string s_name="Base_object"). Функция - создает объект класса, в качестве первого аргумента принимает указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии, в качестве второго объекта принимает наименование создаваемого объекта (имеет значение по умолчанию).
15. Класс Object_4, унаследованный от класса Object_base с модификатором public:
    15.1. Параметризированный конструктор Object_4(Object_base* p_head_object,string s_name="Base_object"). Функция - создает объект класса, в качестве первого аргумента принимает указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии, в

качестве второго объекта принимает наименование создаваемого объекта (имеет значение по умолчанию).

16. Класс Object_5, унаследованный от класса Object_base с модификатором public:
    16.1. Параметризированный конструктор Object_5(Object_base* p_head_object,string s_name="Base_object"). Функция - создает объект класса, в качестве первого аргумента принимает указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии, в качестве второго объекта принимает наименование создаваемого объекта (имеет значение по умолчанию).
17. Класс Object_6, унаследованный от класса Object_base с модификатором public:
    17.1. Параметризированный конструктор Object_6(Object_base* p_head_object,string s_name="Base_object"). Функция - создает объект класса, в качестве первого аргумента принимает указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии, в качестве второго объекта принимает наименование создаваемого объекта (имеет значение по умолчанию).

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: Основновной метод программы.

Параметры: отсутсвуют.

Возвращаемое значение: целочисленное значение.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Объявление объекта ov класса Object_apl с параметром отсутствия ссылки	2
2		Вызов метода vuild_tree_objects() для объекта ob	3
3		Вызов метода exec_app() для объекта ob	∅

3.2 Алгоритм конструктора класса Object_2

Функционал: Конструктор.

Параметры: указатель на объект базового класса p_head_object, строковая переменная s_name.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса Object_2

№	Предикат	Действия	№ перехода
1			∅

3.3 Алгоритм конструктора класса Object_3

Функционал: Конструктор.

Параметры: указатель на объект базового класса p_head_object, строковая переменная s_name.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса Object_3

№	Предикат	Действия	№ перехода
1			∅

3.4 Алгоритм конструктора класса Object_4

Функционал: Конструктор.

Параметры: указатель на объект базового класса p_head_object, строковая переменная s_name.

Алгоритм конструктора представлен в таблице 4.

Таблица 4 – Алгоритм конструктора класса Object_4

№	Предикат	Действия	№ перехода
1			∅

3.5 Алгоритм конструктора класса Object_5

Функционал: Конструктор.

Параметры: указатель на объект базового класса p_head_object, строковая

переменная s_name.

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм конструктора класса Object_5

№	Предикат	Действия	№ перехода
1			∅

3.6 Алгоритм конструктора класса Object_6

Функционал: Конструктор.

Параметры: указатель на объект базового класса p_head_object, строковая переменная s_name.

Алгоритм конструктора представлен в таблице 6.

Таблица 6 – Алгоритм конструктора класса Object_6

№	Предикат	Действия	№ перехода
1			∅

3.7 Алгоритм метода exec_app класса Object_apl

Функционал: Запуск приложения.

Параметры: отсутсвуют.

Возвращаемое значение: целочисленная переменная.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода exec_app класса Object_apl

№	Предикат	Действия	№ перехода
1		Вызов метода print_tree()	∅

3.8 Алгоритм конструктора класса Object_base

Функционал: конструктор.

Параметры: указатель на объект базового класса p_head_object, строковая переменная s_name.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса Object_base

№	Предикат	Действия	№ перехода
1		p_head_object=p_head_object	2
2	p_head_object не равен значению пустого указателя	Добавление головному объекту указателя на текущий объект	3
			3
3		s_name=s_name	∅

3.9 Алгоритм метода get_object_by_name_otnow класса Object_base

Функционал: Поиск нужного элемента по имени на ветке дерева иерархии.

Параметры: строковый тип s_name.

Возвращаемое значение: Указатель на класс Object_base.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода get_object_by_name_otnow класса Object_base

№	Предикат	Действия	№ перехода
1		Вызов метода count с передачей параметра s_name	2
2	результат вызова не равен 1	Возвращение значения пустого указателя	∅
		Вызов метода search_object с передачей параметра s_name Возвращение результата вызова	∅

3.10 Алгоритм метода get_object_by_name_otroot класса Object_base

Функционал: Поиск нужного элемента по имени на дереве иерархии.

Параметры: строковый тип s_name.

Возвращаемое значение: Указатель на класс Object_base.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода get_object_by_name_otroot класса Object_base

№	Предикат	Действия	№ перехода
1		Объявление указателя на объект базового класса start и присвоение ему указателя на текущий объект	2
2		Вызов метода get_head() для объекта start	3
3	Результат вызова не равен значению пустого указателя	Вызов метода get_head() для объекта start и присвоение ему же результата вызова	3
			4
4		Вызов метода get_object_by_name_otnow с параметром s_name для start Возвращение результата вызова	∅

3.11 Алгоритм метода set_status класса Object_base

Функционал: Задает статус готовности объекта.

Параметры: целочисленный тип status.

Возвращаемое значение: отсутствует.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода set_status класса Object_base

№	Предикат	Действия	№ перехода
1	status=0	для текущего объекта: status=0	2

№	Предикат	Действия	№ перехода
	p_head_object не равен значению пустого указателя или status головного объека не равен 0	для текущего объекта: status=status	∅
			∅
2	для всех подчиненных объектов	вызов метода set_status с параметром status	2
			∅

3.12 Алгоритм метода print_tree_otnow класса Object_base

Функционал: Вывод ветки объектов от текущего.

Параметры: целочисленный тип nomer, определяет уровень объекта на дереве иерархии.

Возвращаемое значение: остутсвует.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода print_tree_otnow класса Object_base

№	Предикат	Действия	№ перехода
1	o<nomer		2
		Вывод 4 пробелов o=o+1	1
2		Вывод s_name	3
3		Вызов метода print_tree_otnow с параметром nomer+1 для всех подчиненных объектов	∅

3.13 Алгоритм метода print_tree класса Object_base

Функционал: Вывод дерева иерархии.

Параметры: остутсвуют.

Возвращаемое значение: отсутсвует.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода print_tree класса Object_base

№	Предикат	Действия	№ перехода
1		Вывод Object tree	2
2		Вызов метода print_tree_otnow() с параметром 0	3
3		Вывод The tree of objects and their readiness	4
4		Вызов метода print_tree_otnow_status() с параметром 0	∅

3.14 Алгоритм метода build_tree_objects класса Object_apl

Функционал: Построение дерева иерархии.

Параметры: отсутствуют.

Возвращаемое значение: отсутствуют.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода build_tree_objects класса Object_apl

№	Предикат	Действия	№ перехода
1		Объявление указателей на класс Object_base last, parent и присвоение им значения пустого указателя и указателя на текущий объект соответственно	2
2		Объявление двух строковых переменных name1 и name2	3
3		Ввод name1	4
4		Вызов метода set_name с передачей параметра name1	5
5		Объявление целочисленной переменной g	6
6		Объявление булевой переменной proverka и	7

№	Предикат	Действия	№ перехода
		присваивание ей false	
7		Ввод name1	8
8	name1=endtree		18
			9
9		Ввод name2 и g	10
10		Вызов метода get_object_by_name_otroot с передачей параметра name1 и присвоение результата parent	11
11		Вызов метода search_object с параметром name2	12
12	Результат вызова равен значению пустого указателя		13
			6
13	g=2	Объявление указателя на класс Object_2 obj с вызовом конструктора и передачей ему параметров parent и name2 и присвоение результата last	6
			14
14	g=3	Объявление указателя на класс Object_3 obj с вызовом конструктора и передачей ему параметров parent и name2 и присвоение результата last	6
			15
15	g=4	Объявление указателя на класс Object_4 obj с вызовом конструктора и передачей ему параметров parent и name2 и присвоение результата last	6
			16
16	g=5	Объявление указателя на класс Object_5 obj с вызовом конструктора и передачей ему параметров parent и name2 и присвоение результата last	6
			17

№	Предикат	Действия	№ перехода
17	g=6	Объявление указателя на класс Object_6 obj с вызовом конструктора и передачей ему параметров parent и name2 и присвоение результата last	6
			6
1	Ввод name1	Ввод name1	19
8			∅
19		Ввод g	20
20		Вызов метода get_object_by_name_otroot(name1) и присвоение результата parent	21
21		Вызов метода set_status с передачей параметра g для объекта parent	18

3.15 Алгоритм метода print_tree_otnow класса Object_base

Функционал: Вывод ветки объектов от текущего.

Параметры: целочисленный тип nomer, определяет уровень объекта на дереве иерархии.

Возвращаемое значение: остутсвует.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода print_tree_otnow класса Object_base

№	Предикат	Действия	№ перехода
1	o<nomer		2
		Вывод 4 пробелов o=o+1	1
2		Вывод s_name	3
3	status=0	Вывод is not ready	4
		Вывод is ready	4

№	Предикат	Действия	№ перехода
4		Вызов метода print_tree_otnow_status с параметром nomer+1 для всех подчиненных объектов	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-5.

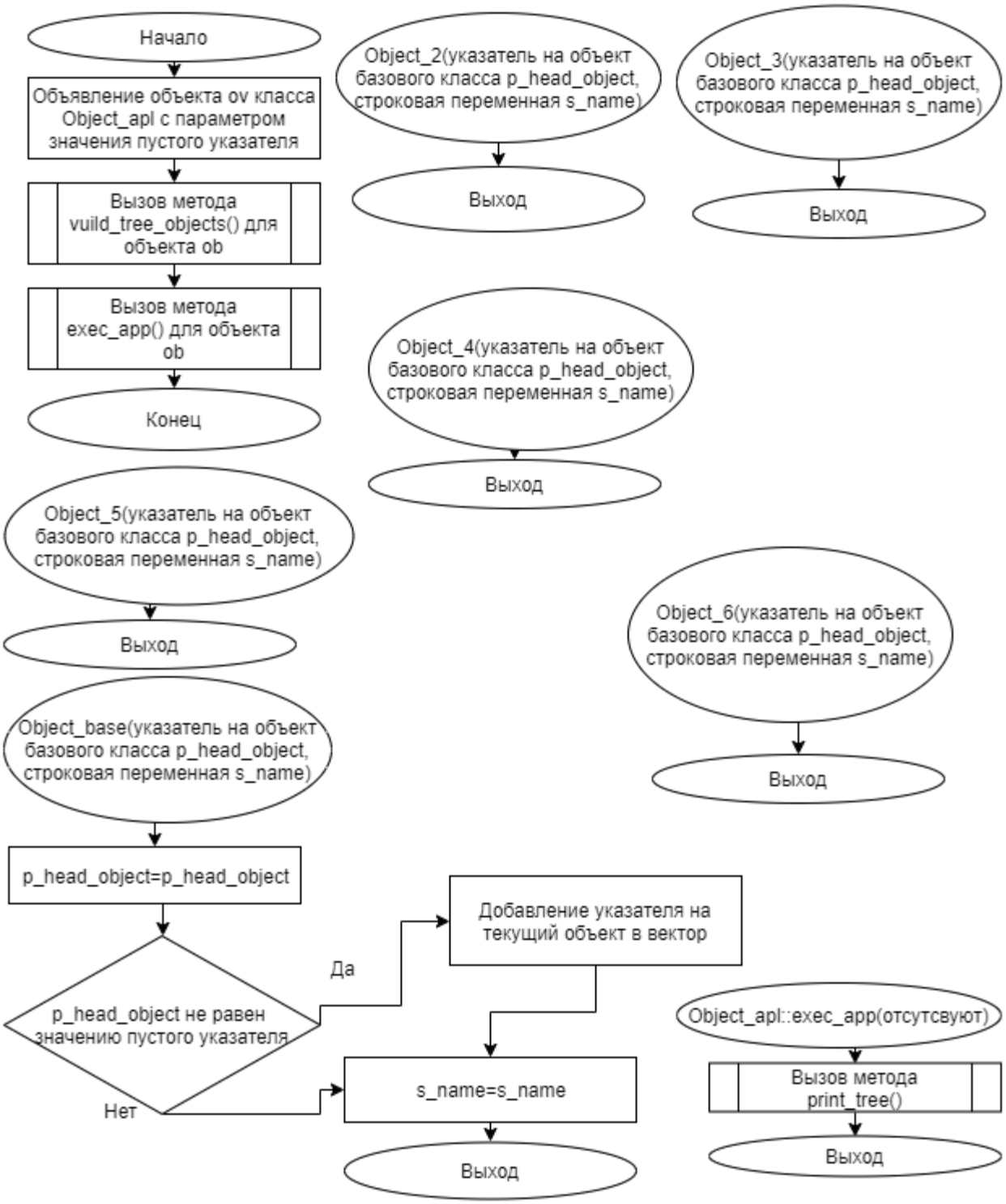

Рисунок 1 – Блок-схема алгоритма

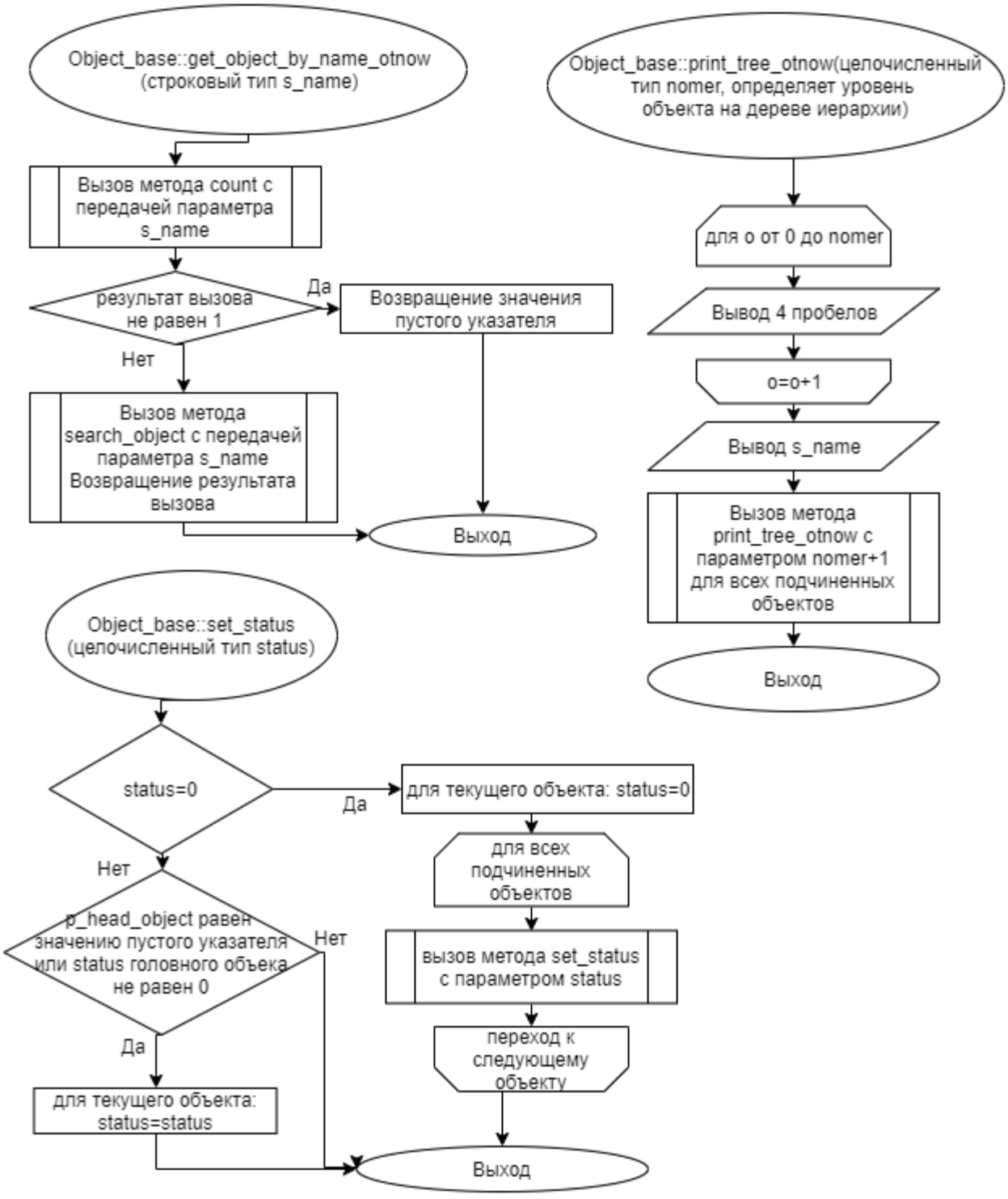

Рисунок 2 – Блок-схема алгоритма

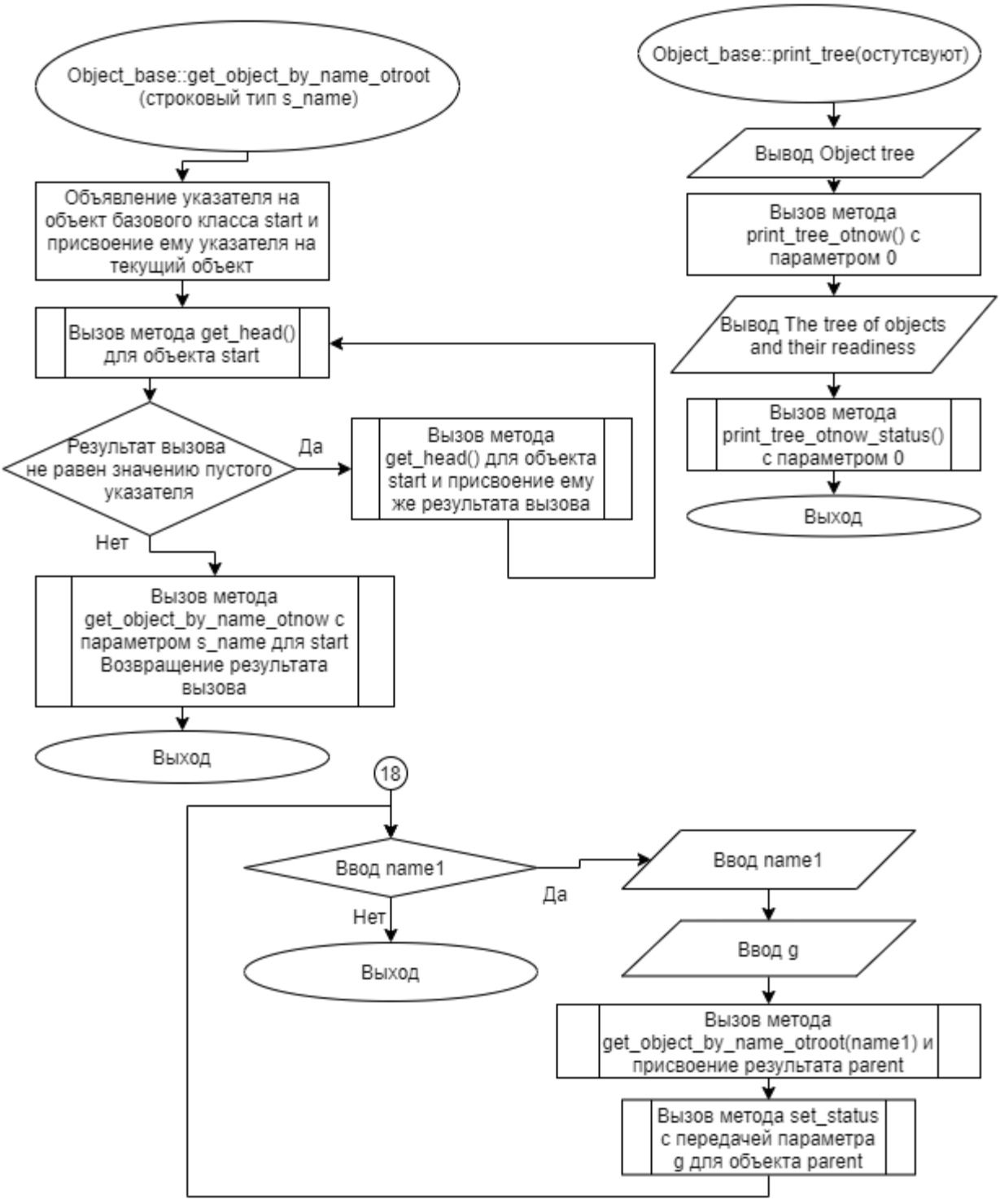

Рисунок 3 – Блок-схема алгоритма

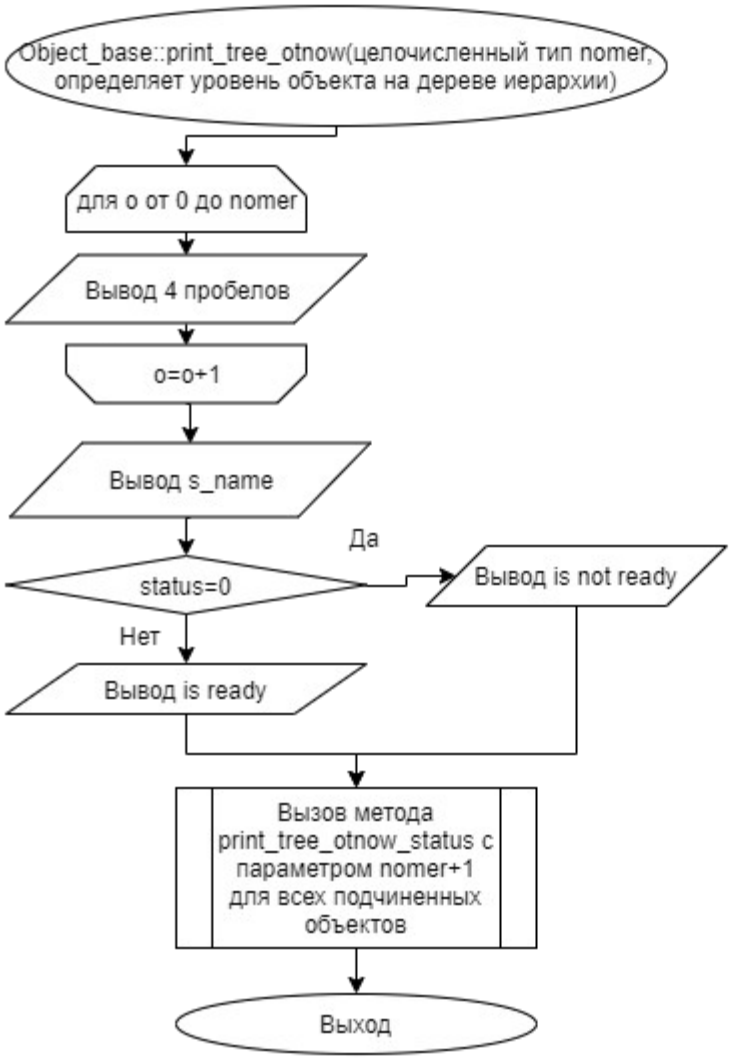

Рисунок 4 – Блок-схема алгоритма

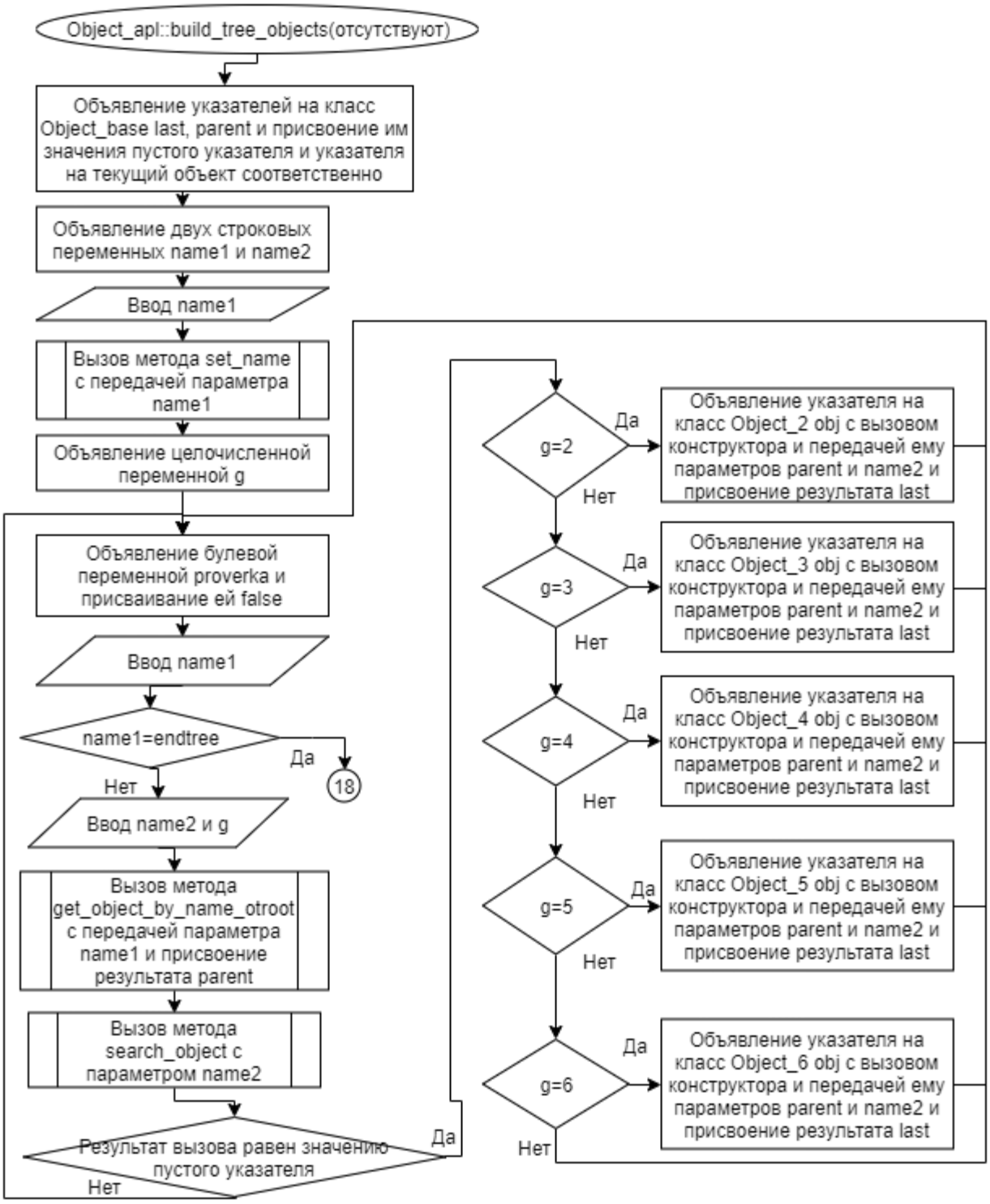

Рисунок 5 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл main.cpp

Листинг 1 – main.cpp

```
#include <iostream>
#include "Object_apl.h"

using namespace std;


int main()
{
	Object_apl ob(nullptr);
	ob.build_tree_objects();
	return ob.exec_app();
}
```

5.2 Файл Object_2.cpp

Листинг 2 – Object_2.cpp

```
#include "Object_2.h"

Object_2::Object_2(Object_base * p_head_object, string
s_name):Object_base(p_head_object,s_name){}
```

5.3 Файл Object_2.h

Листинг 3 – Object_2.h

```
#ifndef __OBJECT_2__H
#define __OBJECT_2__H

#include <iostream>
#include "Object_base.h"

using namespace std;
```

```
class Object_2:public Object_base{
      public:
      Object_2(Object_base* p_head_object,string s_name="Base_object");
};

#endif
```

5.4 Файл Object_3.cpp

Листинг 4 – Object_3.cpp

```
#include "Object_3.h"

Object_3::Object_3(Object_base           *           p_head_object,           string
s_name):Object_base(p_head_object,s_name){}
```

5.5 Файл Object_3.h

Листинг 5 – Object_3.h

```
#ifndef __OBJECT_3__H
#define __OBJECT_3__H

#include <iostream>
#include "Object_base.h"

using namespace std;

class Object_3:public Object_base{
      public:
      Object_3(Object_base* p_head_object,string s_name="Base_object");
};

#endif
```

5.6 Файл Object_4.cpp

Листинг 6 – Object_4.cpp

```
#include "Object_4.h"

Object_4::Object_4(Object_base           *           p_head_object,           string
s_name):Object_base(p_head_object,s_name){}
```

5.7 Файл Object_4.h

Листинг 7 – Object_4.h

```
#ifndef __OBJECT_4__H
#define __OBJECT_4__H

#include <iostream>
#include "Object_base.h"

using namespace std;

class Object_4:public Object_base{
      public:
      Object_4(Object_base* p_head_object,string s_name="Base_object");
};

#endif
```

5.8 Файл Object_5.cpp

Листинг 8 – Object_5.cpp

```
#include "Object_5.h"

Object_5::Object_5(Object_base           *           p_head_object,           string
s_name):Object_base(p_head_object,s_name){}
```

5.9 Файл Object_5.h

Листинг 9 – Object_5.h

```
#ifndef __OBJECT_5__H
#define __OBJECT_5__H

#include <iostream>
#include "Object_base.h"

using namespace std;

class Object_5:public Object_base{
      public:
      Object_5(Object_base* p_head_object,string s_name="Base_object");
};

#endif
```

5.10 Файл Object_6.cpp

Листинг 10 – Object_6.cpp

```
#include "Object_6.h"

Object_6::Object_6(Object_base           *           p_head_object,           string
s_name):Object_base(p_head_object,s_name){}
```

5.11 Файл Object_6.h

Листинг 11 – Object_6.h

```
#ifndef __OBJECT_6__H
#define __OBJECT_6__H

#include <iostream>
#include "Object_base.h"

using namespace std;

class Object_6:public Object_base{
      public:
      Object_6(Object_base* p_head_object,string s_name="Base_object");
};

#endif
```

5.12 Файл Object_apl.cpp

Листинг 12 – Object_apl.cpp

```
#include "Object_apl.h"
#include "Object_2.h"
#include "Object_3.h"
#include "Object_4.h"
#include "Object_5.h"
#include "Object_6.h"
#include <iostream>
#include <vector>
#include <string>

using namespace std;

Object_apl::Object_apl(Object_base * parent):Object_base(parent){
}
void Object_apl::build_tree_objects(){
```

```
	Object_base * last = nullptr;
	Object_base * parent = this;
	string name1,name2;
	cin >> name1;
	this->set_name(name1);
	int g;
	while(true){
		bool proverka=false;
		cin >> name1;
		if(name1=="endtree"){
			break;
		}
		cin >> name2 >> g;
		parent=get_object_by_name_otroot(name1);
		if(search_object(name2)==nullptr){
			proverka=true;
		}
		if(proverka){
			if(g==2){
				last=new Object_2(parent,name2);
			}else if(g==3){
				last=new Object_3(parent,name2);
			}else if(g==4){
				last=new Object_4(parent,name2);
			}else if(g==5){
				last=new Object_5(parent,name2);
			}else if(g==6){
				last=new Object_6(parent,name2);
			}
		}
	}
	while(cin >> name1){
		cin >> g;
		parent=get_object_by_name_otroot(name1);
		parent->set_status(g);
	}
}

int Object_apl::exec_app(){
	print_tree();
	return 0;
}
```

5.13 Файл Object_apl.h

Листинг 13 – Object_apl.h

```
#ifndef __OBJECT_APL__H
#define __OBJECT_APL__H

#include "Object_base.h"
```

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Object_apl:public Object_base{
      public:
      Object_apl(Object_base* parent);
      void build_tree_objects();
      int exec_app();
};

#endif
```

5.14 Файл Object_base.cpp

Листинг 14 – Object_base.cpp

```
#include "Object_base.h"
#include <iostream>
#include <string>
#include <vector>
using namespace std;

Object_base::Object_base(Object_base* p_head_object,string s_name){
      this->p_head_object=p_head_object;
      this->s_name=s_name;
      if(p_head_object!=nullptr){
            p_head_object->p_sub_objects.push_back(this);
      }
}
int Object_base::count(string s_name){
      int counter = 0;
      if(this->s_name==s_name){
            counter++;
      }
      for(auto child:p_sub_objects){
            counter += child->count(s_name);
      }
      return counter;
}
bool Object_base::set_name(string s_new_name){
      for(auto child : p_sub_objects){
            if(child->get_name() == s_new_name){
                  return false;
            }
      }
      s_name=s_new_name;
      return true;
}

Object_base* Object_base::get_head(){
      return p_head_object;
```

```
}

string Object_base::get_name(){
	return s_name;
}

Object_base* Object_base::get_sub_object(string s_name){
	for(auto child : p_sub_objects){
		if(child->get_name()==s_name){
			return child;
		}
	}
	return nullptr;
}

void Object_base::print_tree(){
	cout << "Object tree" << endl;
	print_tree_otnow(0);
	cout << "The tree of objects and their readiness";
	print_tree_otnow_status(0);
}

Object_base::~Object_base(){
}
Object_base * Object_base::search_object(string s_name){
	if(this->get_name()==s_name)
		return this;
	for(auto child : p_sub_objects){
		Object_base * found = child->search_object(s_name);
		if(found !=nullptr)
			return found;
	}
	return nullptr;
}
Object_base * Object_base::get_object_by_name_otnow(string s_name){
	if(count(s_name)!=1)
		return nullptr;
	return search_object(s_name);
}

Object_base * Object_base::get_object_by_name_otroot(string s_name){
	Object_base* start=this;
	while(start->get_head()!=nullptr){
		start=start->get_head();
	}
	return start->get_object_by_name_otnow(s_name);
}

void Object_base::set_status(int status){
	if(status == 0){
		this->status=0;
		for(auto child:p_sub_objects){
			child->set_status(status);
		}
	}else if(get_head()==nullptr){
		this->status=status;
	}else if(get_head()->status!=0){
```

```
            this->status=status;
      }
}

void Object_base::print_tree_otnow(int nomer){
      for(int o=0;o<nomer;o++){
            cout << "    ";
      }
      cout << s_name << endl;
      for(auto child : p_sub_objects){
            child->print_tree_otnow(nomer+1);
      }
}

void Object_base::print_tree_otnow_status(int nomer){
      cout << endl;
      for(int o=0;o<nomer;o++){
            cout << "    ";
      }
      cout << s_name;
      if(status == 0){
            cout << " is not ready";
      }else{
            cout << " is ready";
      }
      for(auto child : p_sub_objects){
            child->print_tree_otnow_status(nomer+1);
      }

}
```

5.15 Файл Object_base.h

Листинг 15 – Object_base.h

```
#ifndef __OBJECT_BASE__H
#define __OBJECT_BASE__H

#include <iostream>
#include <vector>
#include <string>
using namespace std;

class Object_base{
      string s_name;
      Object_base* p_head_object;
      vector<Object_base*> p_sub_objects;
      int status;
      public:
      int count(string s_name);
      Object_base(Object_base* p_head,string s_name="Base_object");
      ~Object_base();
      bool set_name(string s_new_name);
```

```
      Object_base* get_head();
      string get_name();
      Object_base* get_sub_object(string s_name);
      void print_tree();
      Object_base* get_object_by_name_otnow(string s_name);
      Object_base* get_object_by_name_otroot(string s_name);
      Object_base* search_object(string s_name);
      void print_tree_otnow(int nomer);
      void print_tree_otnow_status(int nomer);
      void set_status(int status);
};

#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 16.

Таблица 16 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

app_root
app_root object_1 3
app_root object_2 2
object_2 object_4 3
object_2 object_5 5
object_1 object_7 2
endtree
app_root 1
object_7 3
object_1 1
object_2 -2
object_4 1
	Object tree
	app_root
	    object_1
	        object_7
	    object_2
	        object_4
	        object_5
	The tree of objects and their readiness
	app_root is ready
	    object_1 is ready
	            object_7 is not ready
	    object_2 is ready
	        object_4 is ready
	            object_5 is not ready
		Object tree
		app_root
		    object_1
		        object_7
		    object_2
		        object_4
		        object_5
		The tree of objects and their readiness
		app_root is ready
		    object_1 is ready
		            object_7 is not ready
		    object_2 is ready
		        object_4 is ready
		            object_5 is not ready

app_root
app_root object_1 5
app_root object_2 2
object_2 object_4 3
object_dqwe object_3 2
object_2 object_5 5
object_1 object_7 4
endtree
app_root 1
object_7 3
object_1 0
object_2 0
object_4 1
object_5 3
	Object tree
	app_root
	    object_1
	        object_7
	    object_2
	        object_4
	        object_5
	The tree of objects and their readiness
	app_root is ready
	    object_1 is not ready
	            object_7 is not ready
	    object_2 is not ready
	            object_4 is not ready
	            object_5 is not ready
		Object tree
		app_root
		    object_1
		        object_7
		    object_2
		        object_4
		        object_5
		The tree of objects and their readiness
		app_root is ready
		    object_1 is not ready
		            object_7 is not ready
		    object_2 is not ready
		            object_4 is not ready
		            object_5 is not ready

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).