Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ.....6
1.1 Описание входных данных.....8
1.2 Описание выходных данных.....9
2 МЕТОД РЕШЕНИЯ.....11
3 ОПИСАНИЕ АЛГОРИТМОВ.....13
3.1 Алгоритм конструктора класса cl_2.....13
3.2 Алгоритм конструктора класса cl_3.....13
3.3 Алгоритм конструктора класса cl_4.....14
3.4 Алгоритм конструктора класса cl_5.....14
3.5 Алгоритм метода Out класса cl_base.....14
3.6 Алгоритм метода Out_fc класса cl_base.....15
3.7 Алгоритм метода can_off класса cl_base.....15
3.8 Алгоритм метода Find_current класса cl_base.....16
3.9 Алгоритм метода Find_global класса cl_base.....17
3.10 Алгоритм метода Tree класса cl_application.....17
3.11 Алгоритм метода Start класса cl_application.....18
3.12 Алгоритм функции main.....18
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....20
5 КОД ПРОГРАММЫ.....29
5.1 Файл cl_1.cpp.....29
5.2 Файл cl_1.h.....29
5.3 Файл cl_2.cpp.....29
5.4 Файл cl_2.h.....30
5.5 Файл cl_3.cpp.....30
5.6 Файл cl_3.h.....30
5.7 Файл cl_4.cpp.....31

5.8 Файл cl_4.h.....31
5.9 Файл cl_5.cpp.....31
5.10 Файл cl_5.h.....31
5.11 Файл cl_application.cpp.....32
5.12 Файл cl_application.h.....33
5.13 Файл cl_base.cpp.....34
5.14 Файл cl_base.h.....37
5.15 Файл main.cpp.....37
6 ТЕСТИРОВАНИЕ.....39
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....40

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:

- 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;
- 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;
- метод вывода иерархии объектов (дерева или ветки) от текущего объекта;
- метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта;
- 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:

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

2.1. 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).

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

1.2. 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 задачи используется:

- switch - множественный условный оператор;
- case, default - решение условного оператора;
- break - метод выхода из цикла.

Класс cl_2:

- функционал:
    - метод cl_2 — Конструктор.

Класс cl_3:

- функционал:
    - метод cl_3 — Конструктор.

Класс cl_4:

- функционал:
    - метод cl_4 — Конструктор.

Класс cl_5:

- функционал:
    - метод cl_5 — Конструктор.

Класс cl_base:

- свойства/поля:
    - поле Статус объекта:
        - наименование — status;
        - тип — int;
        - модификатор доступа — private;
- функционал:
    - метод Out — Вывод готовности объектов;
    - метод Out_fc — Вывод дерева иерархии;

- метод can_off — Установка готовности;
- метод Find_current — Поиск объекта по имени от текущего;
- метод Find_global — Поиск объекта по имени от корня.

Класс cl_application:

- функционал:
  - метод Tree — Построение дерева иерархии;
  - метод Start — Запуск программы.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_2			Производный от cl_base	
2	cl_3			Производный от cl_base	
3	cl_4			Производный от cl_base	
4	cl_5			Производный от cl_base	
5	cl_base			Базовый класс	
		cl_2	public		1
		cl_3	public		2
		cl_4	public		3
		cl_5	public		4
		cl_application	public		6
6	cl_application			Класс программы	

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 конструктора класса cl_2

Функционал: Конструктор.

Параметры: cl_base* p_head - указатель на головный объект; string s_object_name - имя объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1		Создание объекта	∅

3.2 Алгоритм конструктора класса cl_3

Функционал: Конструктор.

Параметры: cl_base* p_head - указатель на головный объект; string s_object_name - имя объекта.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1		создание объекта	∅

3.3 Алгоритм конструктора класса cl_4

Функционал: Конструктор.

Параметры: cl_base* p_head - указатель на головный объект; string s_object_name - имя объекта.

Алгоритм конструктора представлен в таблице 4.

Таблица 4 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1		создание объекта	∅

3.4 Алгоритм конструктора класса cl_5

Функционал: Конструктор.

Параметры: cl_base* p_head - указатель на головный объект; string s_object_name - имя объекта.

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1		создание объекта	∅

3.5 Алгоритм метода Out класса cl_base

Функционал: Вывод готовности объектов.

Параметры: int probel - количество пробелов-разделителей.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода Out класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод имени объекта с новой строки и его готовность	2
2	Есть подчиненные объекты	вывод с новой строки имен и готовности подчиненных объектов посредством вызова метода Out_fc для каждого из них с параметром probel+4	∅
			∅

3.6 Алгоритм метода Out_fc класса cl_base

Функционал: Вывод дерева иерархии.

Параметры: int probel - количество пробелов-разделителей.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода Out_fc класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод имени объекта с новой строки	2
2	Есть подчиненные объекты	вывод с новой строки подчиненных объектов посредством вызова метода Out_fc для каждого из них с параметром probel+4	∅
			∅

3.7 Алгоритм метода can_off класса cl_base

Функционал: Установка готовности.

Параметры: int status1 - переменная статус.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода can_off класса cl_base

№	Предикат	Действия	№ перехода
1	существует родитель и он отключен	присвоение status=0	2
	status1==0	присвоение status=0	2
		присвоение status=1	2
2	status==0	отключение всех производных классов (status=false)	∅
			∅

3.8 Алгоритм метода Find_current класса cl_base

Функционал: Поиск объекта по имени от текущего.

Параметры: string name - имя искомого объекта.

Возвращаемое значение: cl_base* - указатель на искомый объект.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода Find_current класса cl_base

№	Предикат	Действия	№ перехода
1	искомое имя= имя текущего объекта	возвращение указателя на текущий объект	∅
			2
2		инициализация cl_base* p_sub = метод Find_current с параметром name для всех производных объектов	3
3	p_sub	возвращение указателя на текущий объект	∅
		возвращение nullptr	∅

3.9 Алгоритм метода Find_global класса cl_base

Функционал: Поиск объекта по имени от корня.

Параметры: string name - имя искомого объекта.

Возвращаемое значение: cl_base* - указатель на искомый объект.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода Find_global класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация указателя p_found на класс cl_base = this	2
2	существует родитель	присвоение p_found = указатель на родителя	2
			3
3		возвращение метода Find_current для p_found с параметром name	∅

3.10 Алгоритм метода Tree класса cl_application

Функционал: Построение дерева иерархии.

Параметры: нет.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода Tree класса cl_application

№	Предикат	Действия	№ перехода
1		инициализация int i_class, i_state	2
2	head!=endtree	ввод с клавиатуры head, sub, i_class	3
			4
3		с помощью switch case создание объекта p_sub класса с имененем "cl_i_class"	2

№	Предикат	Действия	№ перехода
4	cin>>head>>i_state	присвоение p_head=метод Find_global с параметром head	5
			∅
5	p_head!=nullptr	вызов метода can_off для p_head с параметром i_state	4
			4

3.11 Алгоритм метода Start класса cl_application

Функционал: Запуск программы.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности завершения алгоритма.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода Start класса cl_application

№	Предикат	Действия	№ перехода
1		вывод с первой строки "Object tree"	2
2		вызов метода Out_fc()	3
3		вывод с новой строки "The tree of objects and their readiness"	4
4		вызов метода Out()	∅

3.12 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности завершения алгоритма.

Алгоритм функции представлен в таблице 13.

Таблица 13 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		вызов методов создания и вывода иерархии дерева	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-9.

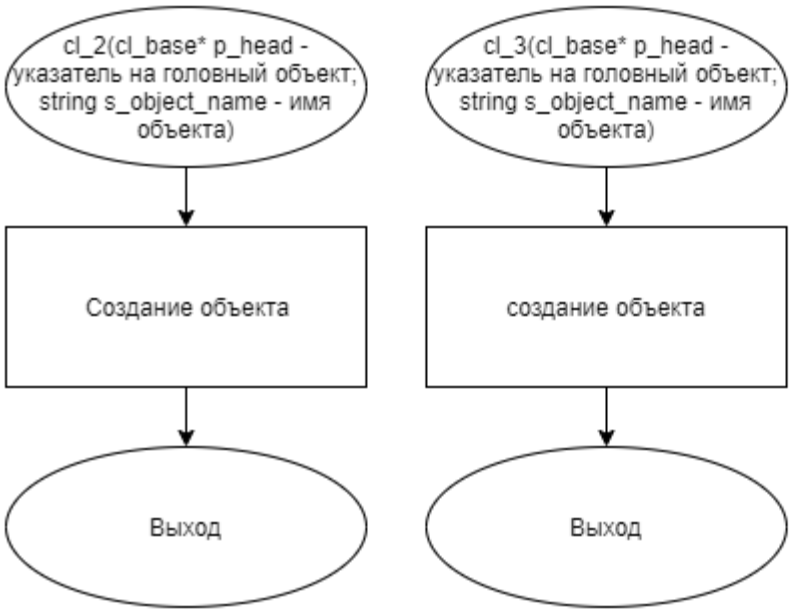

Рисунок 1 – Блок-схема алгоритма

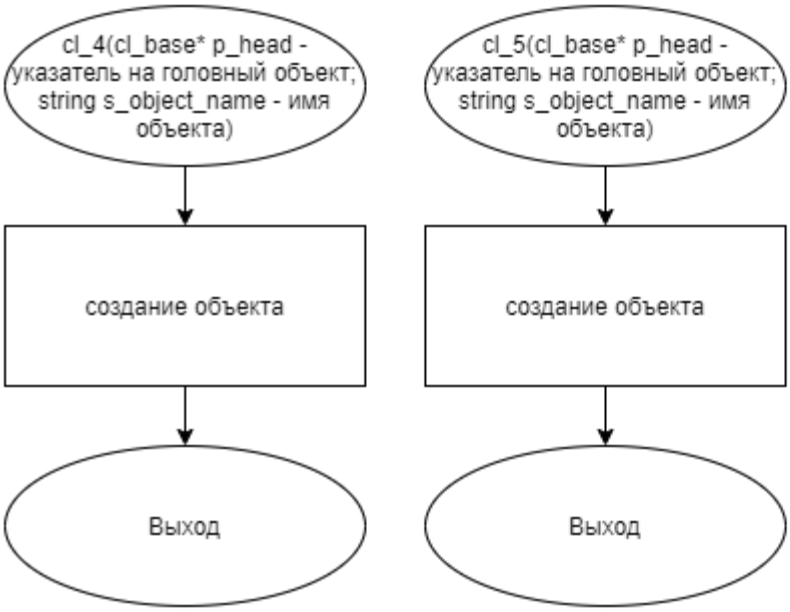

Рисунок 2 – Блок-схема алгоритма

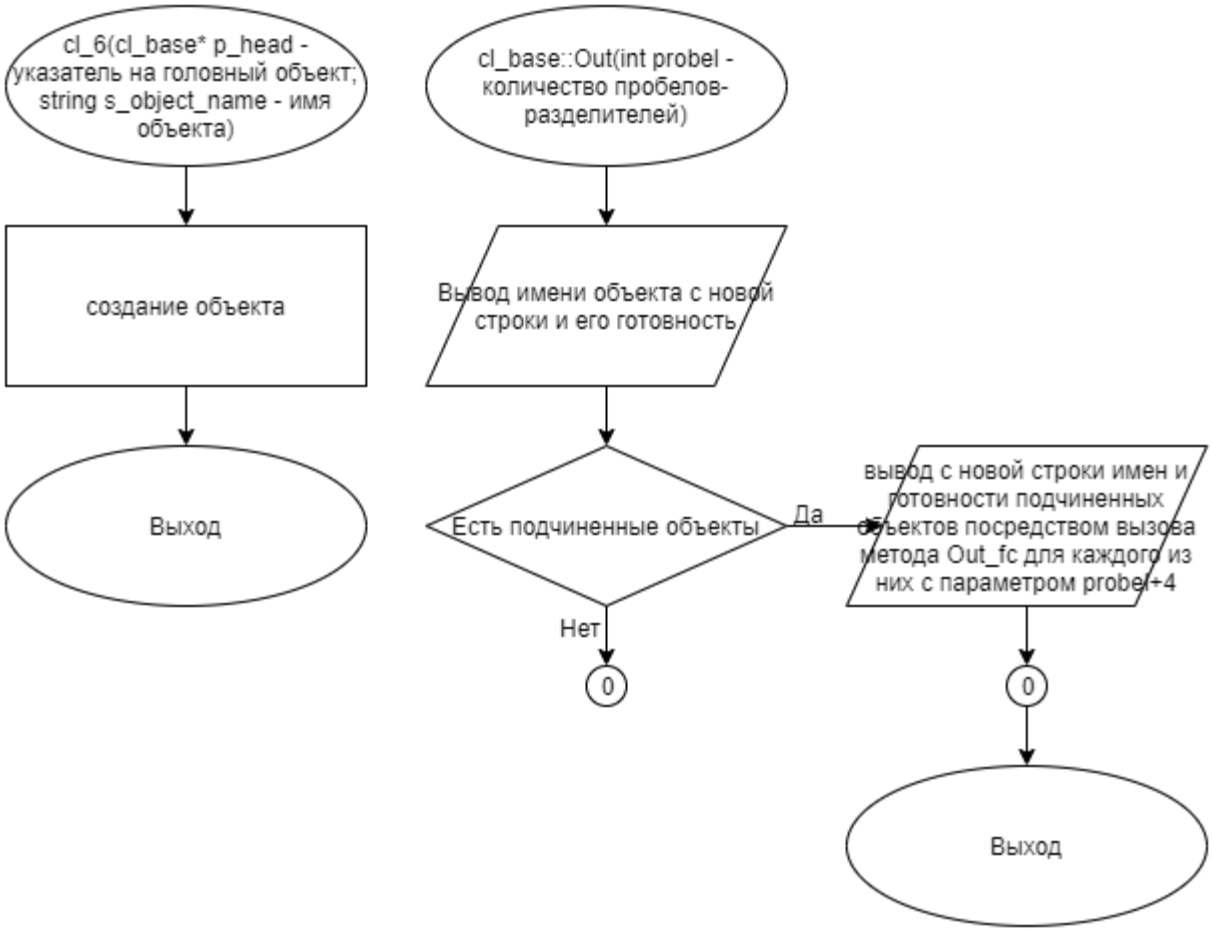

Рисунок 3 – Блок-схема алгоритма

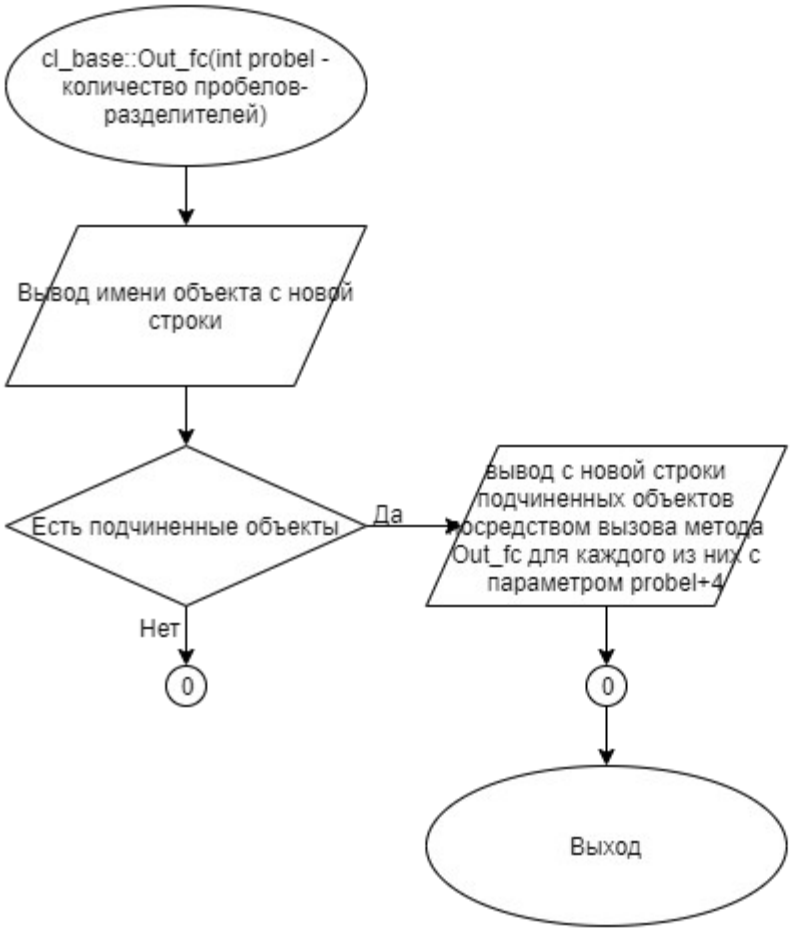

Рисунок 4 – Блок-схема алгоритма

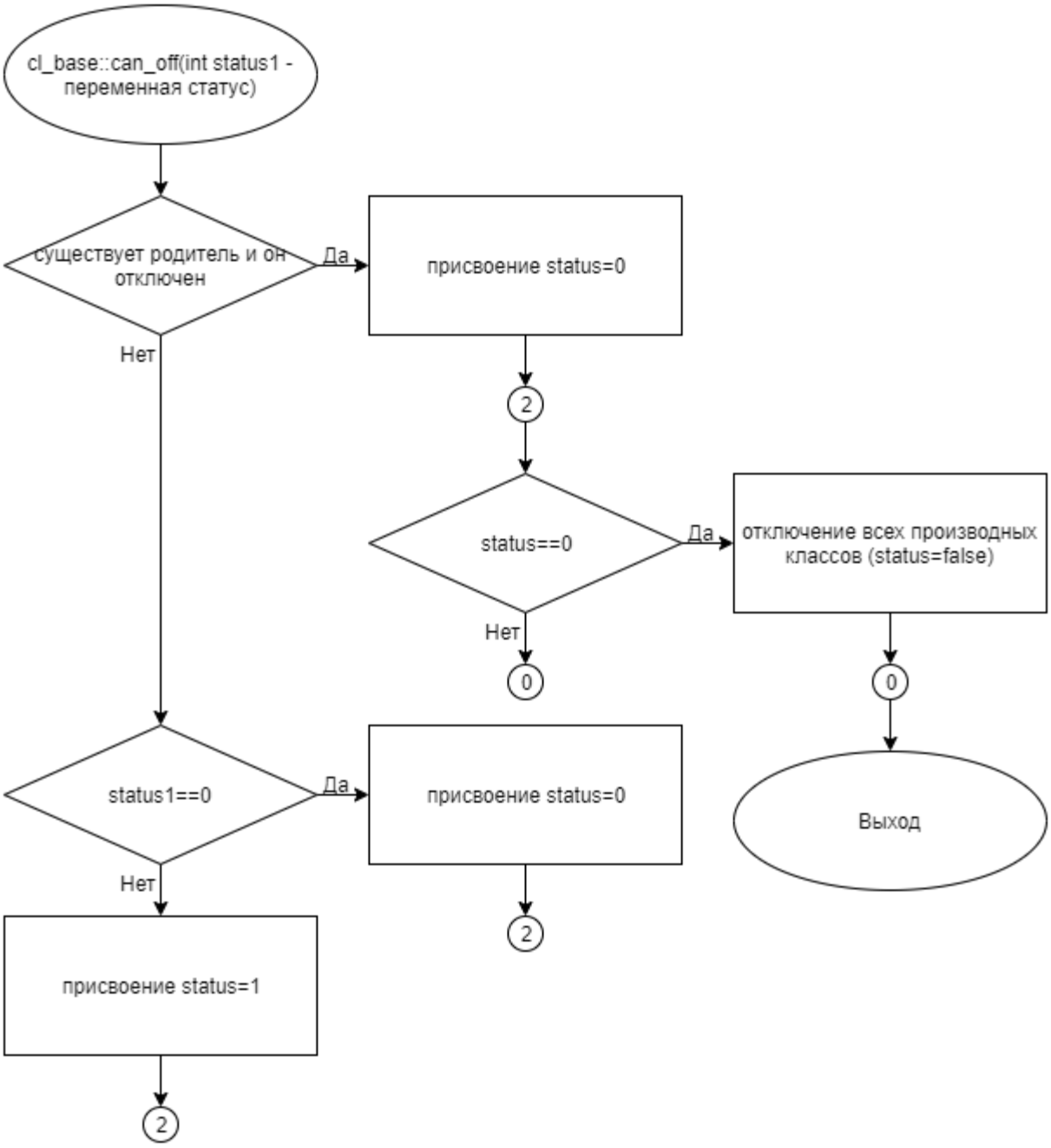

Рисунок 5 – Блок-схема алгоритма

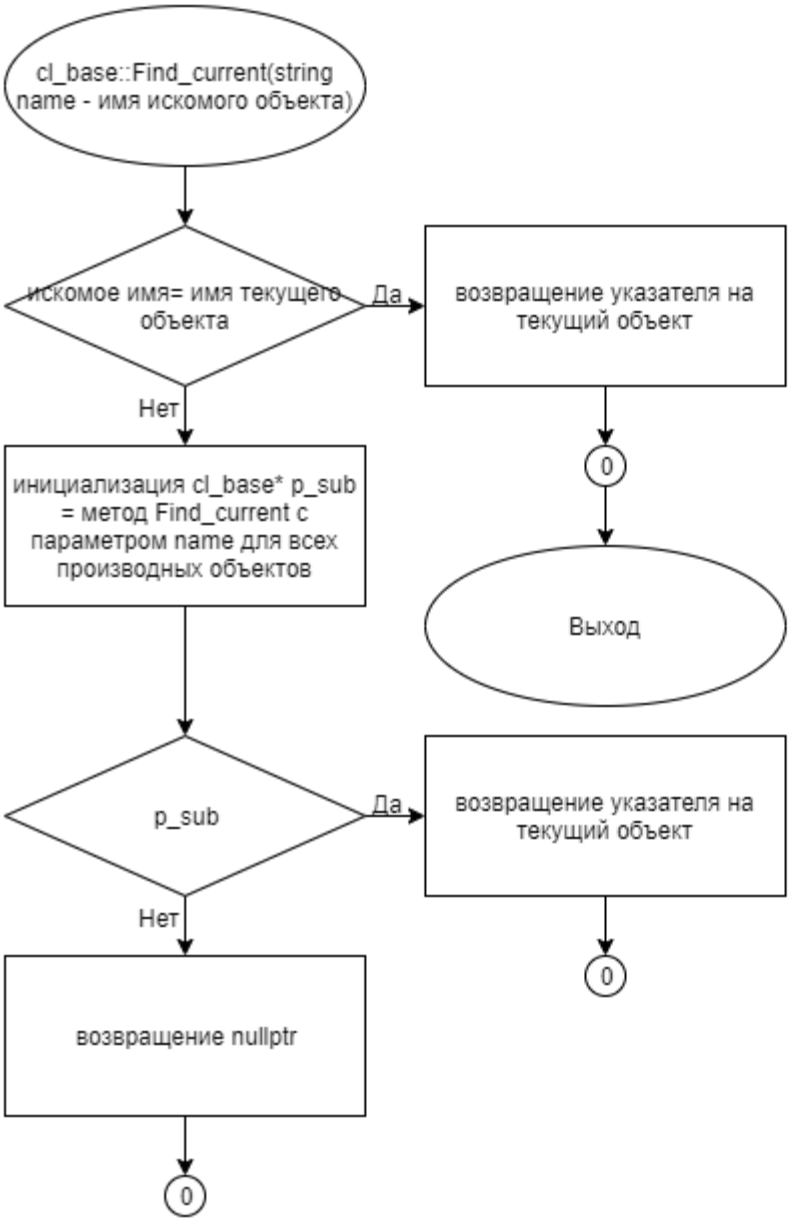

Рисунок 6 – Блок-схема алгоритма

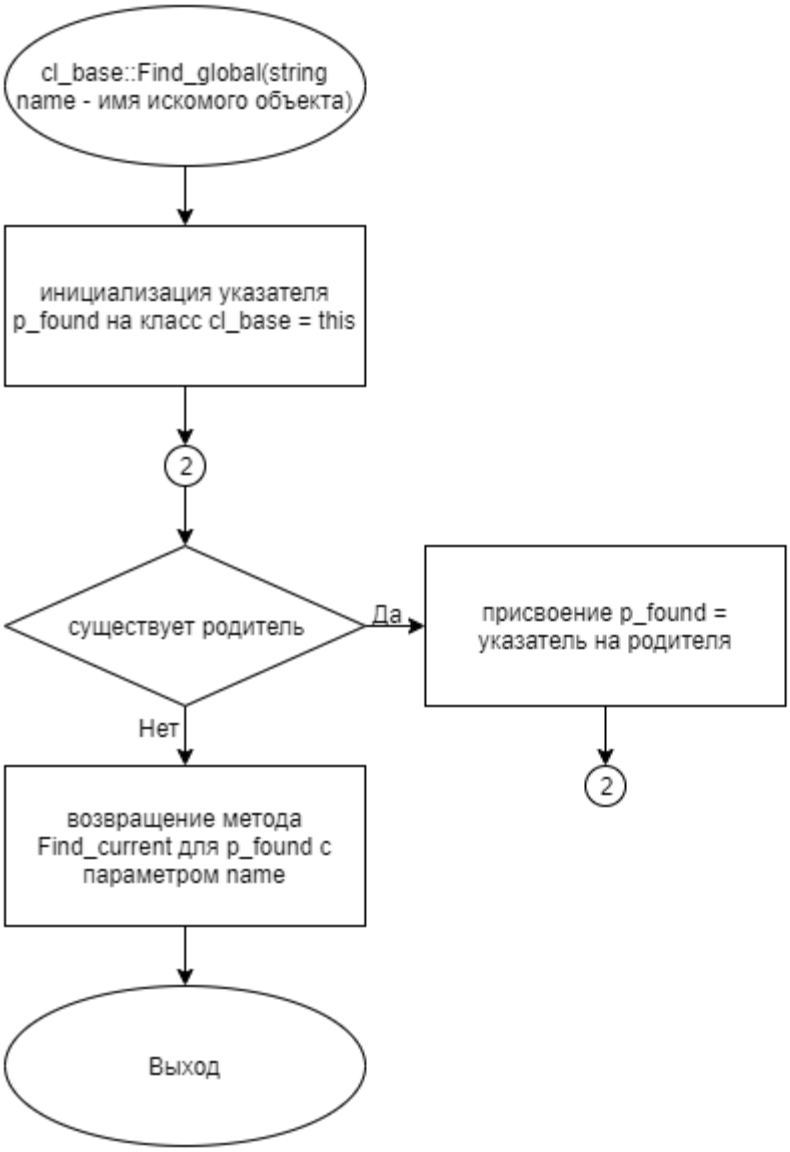

Рисунок 7 – Блок-схема алгоритма

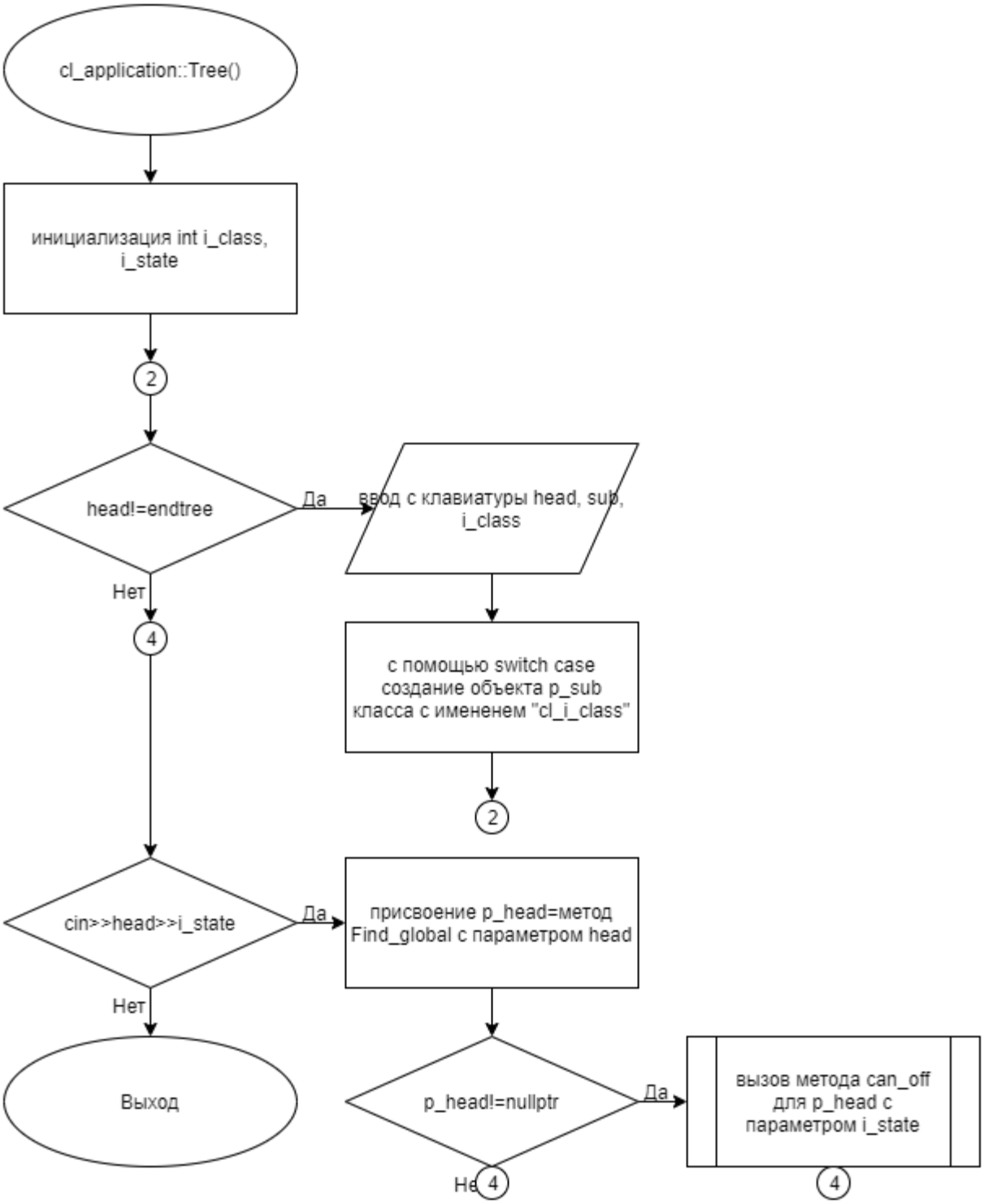

Рисунок 8 – Блок-схема алгоритма

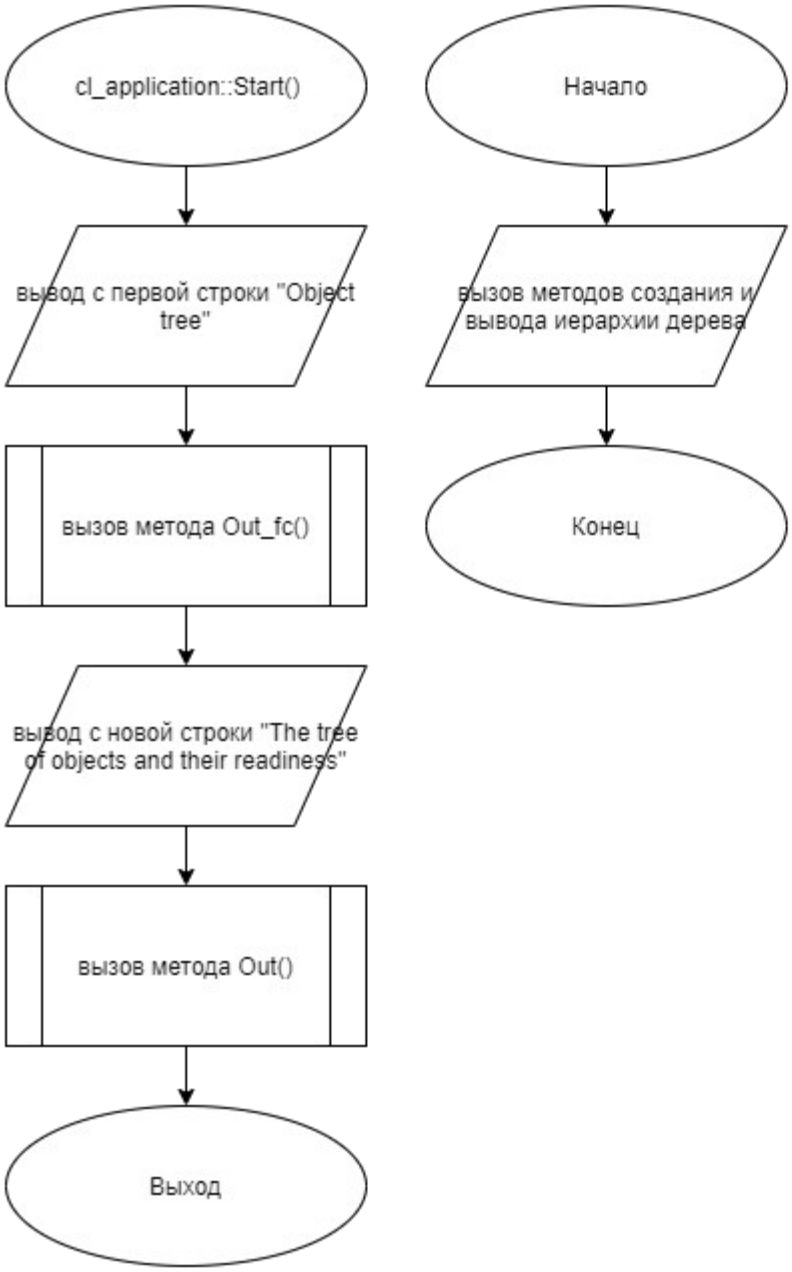

Рисунок 9 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"
cl_1::cl_1(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef __CL_1__H
#define __CL_1__H
#include "cl_base.h"
class cl_1:public cl_base{
   public:
   cl_1(cl_base* p_head, string s_object_name);
};
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"
cl_2::cl_2(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.4 Файл cl_2.h

Листинг 4 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"
class cl_2:public cl_base{
   public:
   cl_2(cl_base* p_head, string s_object_name);
};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"
cl_3::cl_3(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.6 Файл cl_3.h

Листинг 6 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"
class cl_3:public cl_base{
   public:
   cl_3(cl_base* p_head, string s_object_name);
};


#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.8 Файл cl_4.h

Листинг 8 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include "cl_base.h"
class cl_4:public cl_base{
   public:
   cl_4(cl_base* p_head, string s_object_name);
};
#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"
cl_5::cl_5(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.10 Файл cl_5.h

Листинг 10 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include "cl_base.h"
```

```
class cl_5:public cl_base{
   public:
   cl_5(cl_base* p_head, string s_object_name);
};
#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
cl_application::cl_application(cl_base*
p_head_object):cl_base(p_head_object){}
void cl_application::Tree(){
   string head, sub;
   cl_base* p_head=this;
   cl_base* p_sub=nullptr;
   int i_class, i_state;
   cin>>head;
   Change_name(head);
   //Ввод иерархии
   while(true){
      cin>>head;
      if (head=="endtree"){
         break;
      }
      cin>> sub >> i_class;
      p_head=Find_global(head);
      /*
      if(p_sub!=nullptr && head==p_sub->Get_name()){
         p_head=p_sub;
      }
      if(head==p_head->Get_name() && p_head->Get_ptr(sub)==nullptr){
         p_sub=new cl_1 (p_head,sub);
      }
      */
      if (p_head&&!p_head->Get_ptr(sub)){
         switch (i_class)
         {
            case 1:
               p_sub=new cl_1(p_head,sub);
               break;
            case 2:
               p_sub=new cl_2(p_head,sub);
               break;
            case 3:
               p_sub=new cl_3(p_head,sub);
               break;
            case 4:
               p_sub=new cl_4(p_head,sub);
```

```
                  break;
               case 5:
                  p_sub=new cl_5(p_head,sub);
                  break;
               default:
                  break;
            }
         }
      }
      //Установка состояний объекта
      while(cin>>head>>i_state){
         p_head = Find_global(head);
         if(p_head!=nullptr) {p_head->can_off(i_state);}
      }
      //cout<<"Статусы заданы\n";
   }
   /*
   void cl_application::Tree_1(){
      string head, sub;
      cl_base* p_head=this;
      cl_base* p_sub=nullptr;
      cin>>head;
      Change_name(head);
      while(true){
         cin>>head>>sub;
         if (head==sub){
            break;
         }
         if(p_sub!=nullptr && head==p_sub->Get_name()){
            p_head=p_sub;
         }
         if(head==p_head->Get_name() && p_head->Get_ptr(sub)==nullptr){
            p_sub=new cl_1 (p_head,sub);
         }
      }
   }
   */
   int cl_application::Start(){
      cout<<"Object tree"<<endl;
      Out_fc();
      cout<<endl<<"The tree of objects and their readiness"<<endl;
      Out();
      return 0;
   }
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
   #ifndef __CL_APPLICATION__H
```

```
#define __CL_APPLICATION__H
#include "cl_base.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
class cl_application:public cl_base{
   public:
   cl_application(cl_base* p_head_object);
   void Tree();
   //void Tree_1();
   int Start();
};
#endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
#include "cl_base.h"
using namespace std;
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);
   }
}
string cl_base::Get_name(){
   return s_object_name;
}//Получение имени объекта
bool cl_base::Change_name(string name){
   if(Get_baseptr()!=nullptr){
      for (int i=0; i<p_head_object->subordinate_objects.size();i++){
         if (p_head_object->subordinate_objects[i]->Get_name()==name){
            return false;
         }
      }
   }
   this->s_object_name=name;
   return true;
}//Поменять имя объекта
cl_base* cl_base::Get_ptr(string s_object_name){
   for (int i=0; i<subordinate_objects.size();i++){
      if (subordinate_objects[i]->Get_name()==s_object_name){
         return subordinate_objects[i];
      }
   }
```

```
   return nullptr;
}//Получение указателя на объект
cl_base* cl_base::Get_baseptr(){
   return p_head_object;
}//Получение головного указателя
//Статус
/*
int cl_base::Get_status(){
   return status;
}
*/
/*Сет статус
void cl_base::Set_status(int i){
   this->status=i;
}
*/
//Статус
void cl_base::can_off(int status1){
   if (status1==0){
      status=0;
      for (auto sub:subordinate_objects){
         sub->can_off(0);
      }
      //cout<<"Поставил фальш\n";
   }
   if (Get_baseptr()&&(Get_baseptr()->status==0)){
      return;
   }
   else if (status1!=0){
      status=1;
   }
}

void cl_base::Out_fc(int probel){
   cout<<Get_name();
   if(subordinate_objects.size()!=0){
      for (auto sub:subordinate_objects){
         cout<<endl;
         for(int i =0; i<probel;i++) cout<<" ";
         sub->Out_fc(probel+4);
      }
   }
}//Вывод дерева
/*
void cl_base::Out(int probel){
   string ready="is ready";
   string nready="is not ready";
   cout<<"    "<<this->Get_name();
   if(this->Get_status()!=0){
      cout<<ready<<endl;
   }
   else{cout<<nready<<endl;}
}*/
//Вывод дерева
void cl_base::Out(int probel){
```

```
   cout<< Get_name();
   if (status==0)cout<<" is not ready";
   else if (status!=0){cout<<" is ready";}
   if (subordinate_objects.size()!=0){
      for(int i=0; i<subordinate_objects.size();i++){
         cout<<endl;
         for(int i =0; i<probel;i++) cout<<" ";
         subordinate_objects[i]->Out(probel+4);
      }
   }
}//Вывод дерева

//Поиск не на основе очереди, был на ней, но не судьба
cl_base* cl_base::Find_current(string name){
   if (Get_name() == name)
   {
      //cout<<"Нашел\n";
      return this;
   }
   for (auto sub : subordinate_objects)
   {
      cl_base* p_sub = sub->Find_current(name);
      if (p_sub)
      {
         return p_sub;
      }
   }
   return nullptr;
}
//Счетчик
/*
int cl_base::Count(string name){
   int count=0;
   if (Get_name()==name) count++;
   for(auto sub:subordinate_objects)count+=sub->Count(name);
   return count;
}
*/
cl_base* cl_base::Find_global(string name){
   cl_base* p_found = this;
   while(p_found->Get_baseptr()) {p_found=p_found->Get_baseptr();}
   //cout<<"Нащел родича \n";
   //cout<<p_found->Find_current(name)->Get_name();
   return p_found->Find_current(name);
}
cl_base::~cl_base(){
   for (int i=0; i<subordinate_objects.size();i++){
      delete subordinate_objects[i];
   }
}
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <string>
#include <vector>
#include <queue>
using namespace std;
class cl_base{
   private:
   int status=0;
   string s_object_name;
   cl_base * p_head_object;
   vector <cl_base*> subordinate_objects;
   public:

   cl_base(cl_base* p_head_object,string s_object_name="Base_object");
   bool Change_name(string name);
   string Get_name();
   cl_base* Get_baseptr();
   void Out(int probel=4);
   void Out_fc(int probel=4);
   void can_off(int status);
   //void Out1();
   //int Get_status();
   //void Set_status(int i);
   //int Count(string name);
   cl_base* Find_current(string name);
   cl_base* Find_global(string name);
   cl_base* Get_ptr(string s_object_name);
   //cl_base* Search(string name);
   ~cl_base();
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "cl_base.h"
#include "cl_application.h"
int main()
{
```

```
        cl_application  ob_cl_application ( nullptr );
        ob_cl_application.Tree();
        //cout<<"Дерево построено\n";
        return ob_cl_application.Start();
    }
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 14.

Таблица 14 – Результат тестирования программы

Входные данные	**Ожидаемые выходные данные**	**Фактические выходные данные**
app_root endtree app_root 1	Object tree app_root The tree of objects and their readiness app_root is ready	Object tree app_root The tree of objects and their readiness app_root is 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).