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:

- 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

Класс cl_base:

- свойства/поля:
    - поле Состояние объекта:
        - наименование — state;
        - тип — int;
        - модификатор доступа — protected;
- функционал:
    - метод find_on_branch — Поиск объекта по имени на ветке;
    - метод find_on_tree — Поиск объекта по имени на всём дереве;
    - метод print — Вывод дерева объектов;
    - метод print_state — Вывод дерева объектов и их состояний;
    - метод set_state — Установка состояния объекта.

Класс cl_application:

- функционал:
    - метод build_tree_objects — Метод построения дерева объектов;
    - метод exec_app — Метод запуска выполнения задачи.

Класс cl_obj2:

- функционал:
    - метод cl_obj2 — Параметризированный конструктор.

Класс cl_obj3:

- функционал:
    - метод cl_obj3 — Параметризированный конструктор.

Класс cl_obj4:

- функционал:
    - метод cl_obj4 — Параметризированный конструктор.

Класс cl_obj5:

- функционал:
    - метод cl_obj5 — Параметризированный конструктор.

Класс cl_obj6:

- функционал:
    - метод cl_obj6 — Параметризированный конструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Базовый класс для остальных компонентов дерева	
		cl_application	public		2
		cl_obj2	public		3
		cl_obj3	public		4
		cl_obj4	public		5
		cl_obj5	public		6
		cl_obj6	public		7
2	cl_application			Главный объект выполняющий построение и обработку дерева объектов	
3	cl_obj2			Класс для построения дерева	
4	cl_obj3			Класс для построения дерева	
5	cl_obj4			Класс для построения дерева	
6	cl_obj5			Класс для построения дерева	
7	cl_obj6			Класс для построения дерева	

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 find_on_branch класса cl_base

Функционал: Поиск объекта по имени на ветке.

Параметры: string name - наименование искомого объекта.

Возвращаемое значение: cl_base* - указатель на найденный объект.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода find_on_branch класса cl_base

№	Предикат	Действия	№ перехода
1	this->name==name	Возврат указателя на себя	∅
		Инициализация итератора child типа cl_base* по children_ptr	2
2	child в children_ptr	Инициализация укзазателя fchild типа cl_base* равному значению вызова метода find_on_branch(name) у объекта по указателю child	3
		Возврат nullptr	∅
3	fchild	Возврат fchild	∅
			4
4		Ход итерации child	2

3.2 Алгоритм метода find_on_tree класса cl_base

Функционал: Поиск объекта по имени на всём дереве.

Параметры: string name - наименование искомого объекта.

Возвращаемое значение: cl_base* - указатель на найденный объект.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода find_on_tree класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация укзазателя fheader типа cl_base* равному указателю на текущий объект this	2
2	Значение вызова метода get_header у объекта fheader	Присвоение fheader значения вызова метода get_header у объекта по указателю fheader	2
		Возврат значения вызова find_on_branch с параметром name у объекта по указателю fheader	∅

3.3 Алгоритм метода print класса cl_base

Функционал: Вывод дерева объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода print класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод значения поля name	2
2		Инициализация tab = 0 типа int	3
3		Инициализация укзазателя par типа cl_base* равному указателю на текущий объект this	4
4	Значение вызова метода get_header у объекта par	Присвоение par значения вызова метода get_header у объекта по указателю par	5
			6
5		tab += 1	4
6	children_ptr не пустой	Инициализация итератора child типа cl_base* по	7

№	Предикат	Действия	№ перехода
		children_ptr	
			∅
7	child в children_ptr		8
			∅
8		Вывод конца строки	9
9		Инициализация i = 0 типа int	10
10	i <= tab	Вывод 4 пробела	11
		Вызов метода print у объекта по указателю child	12
11		i++	10
12		Ход итерации child	7

3.4 Алгоритм метода set_state класса cl_base

Функционал: Установка состояния объекта.

Параметры: int state - новое состояние объекта.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода set_state класса cl_base

№	Предикат	Действия	№ перехода
1	header_ptr && Значения поля state у header_ptr == 0	Установка значения поля state = 0	2
		Установка значения поля state = state	2
2	state == 0	Инициализация итератора child типа cl_base* по children_ptr	3
			∅
3	child в children_ptr	Вызов метода set_state у объекта по указателю child с передачей параметра 0	4
			∅

№	Предикат	Действия	№ перехода
4		Ход итерации child	3

3.5 Алгоритм метода build_tree_objects класса cl_application

Функционал: Метод построения дерева объектов.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода build_tree_objects класса cl_application

№	Предикат	Действия	№ перехода
1		Объявление переменных parentName, childName типа string	2
2		Объявление переменной childClass типа int	3
3		Ввод значения parentName	4
4		Вызов метода edit_name с параметром parentName	5
5		Ввод значения parentName	6
6		Вызов метода edit_name с параметром parentName	7
7	parentName == "endtree"		11
		Инициализация указателя parent типа cl_base* значением вызова метода find_on_tree с параметром parentName	8
8		Ввод значения childName и childClass	9
9	parent && !(значение вызова get_child с параметром childName у объекта по указателю parent)	switch(childClass)	10
			11
10	childClass == 2	Выделение памяти объекту класса cl_obj2 с	11

№	Предикат	Действия	№ перехода
		параметрами parent и childName	
	childClass == 3	Выделение памяти объекту класса cl_obj3 с параметрами parent и childName	11
	childClass == 4	Выделение памяти объекту класса cl_obj4 с параметрами parent и childName	11
	childClass == 5	Выделение памяти объекту класса cl_obj5 с параметрами parent и childName	11
	childClass == 6	Выделение памяти объекту класса cl_obj6 с параметрами parent и childName	11
11		Инициализация s = 0 типа int	12
12	cin >> parentName >> s	Инициализация указателя obj типа cl_base* значением вызова метода find_on_tree с параметром s	13
			∅
13	obj	Вызов метода set_state с параметром s у объекта по указателю obj	12
			12

3.6 Алгоритм метода exec_app класса cl_application

Функционал: Метод запуска выполнения задачи.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода
1		Вывод "Object tree"	2
2		Вызов метода print	3

№	Предикат	Действия	№ перехода
3		Вывод "The tree of objects and their readiness	4
4		Вызов метода print_state	5
5		Возврат значения 0	∅

3.7 Алгоритм конструктора класса cl_obj2

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - название.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса cl_obj2

№	Предикат	Действия	№ перехода
1		Передача параметров header и name параметризированному конструктору родительского класса cl_base	∅

3.8 Алгоритм конструктора класса cl_obj3

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - название.

Алгоритм конструктора представлен в таблице 9.

Таблица 9 – Алгоритм конструктора класса cl_obj3

№	Предикат	Действия	№ перехода
1		Передача параметров header и name параметризированному конструктору родительского класса cl_base	∅

3.9 Алгоритм конструктора класса cl_obj4

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - название.

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса cl_obj4

№	Предикат	Действия	№ перехода
1		Передача параметров header и name параметризированному конструктору родительского класса cl_base	∅

3.10 Алгоритм конструктора класса cl_obj5

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - название.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса cl_obj5

№	Предикат	Действия	№ перехода
1		Передача параметров header и name параметризированному конструктору родительского класса cl_base	∅

3.11 Алгоритм конструктора класса cl_obj6

Функционал: Параметризированный конструктор.

Параметры: cl_base* header - указатель на головной объект, string name - название.

Алгоритм конструктора представлен в таблице 12.

Таблица 12 – Алгоритм конструктора класса cl_obj6

№	Предикат	Действия	№ перехода
1		Передача параметров header и name параметризированному конструктору родительского класса cl_base	∅

3.12 Алгоритм метода print_state класса cl_base

Функционал: Вывод дерева объектов и их состояний.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода print_state класса cl_base

№	Предикат	Действия	№ перехода
1		Вывод значения поля name	2
2	Значение поля state == 0	Вывод " is not ready"	3
		Вывод " is ready"	3
3		Инициализация tab = 0 типа int	4
4		Инициализация укзазателя par типа cl_base* равному указателю на текущий объект this	5
5	Значение вызова метода get_header у объекта par	Присвоение par значения вызова метода get_header у объекта по указателю par	6
			7
6		tab += 1	5
7	children_ptr не пустой	Инициализация итератора child типа cl_base* по children_ptr	9
			∅
8	child в children_ptr		9
			∅
9		Вывод конца строки	10

№	Предикат	Действия	№ перехода
10		Инициализация i = 0 типа int	11
11	i <= tab	Вывод 4 пробела	12
		Вызов метода print у объекта по указателю child	13
12		i++	11
13		Ход итерации child	8

3.13 Алгоритм функции main

Функционал: Главная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 14.

Таблица 14 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Инициализация объекта ob_cl_application класса cl_application с передачей nullptr в параметризированный конструктор	2
2		Вывод метода build_tree_objects у объекта ob_cl_application	3
3		Возврат значения вызова метода exec_app у объекта ob_cl_application	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-13.

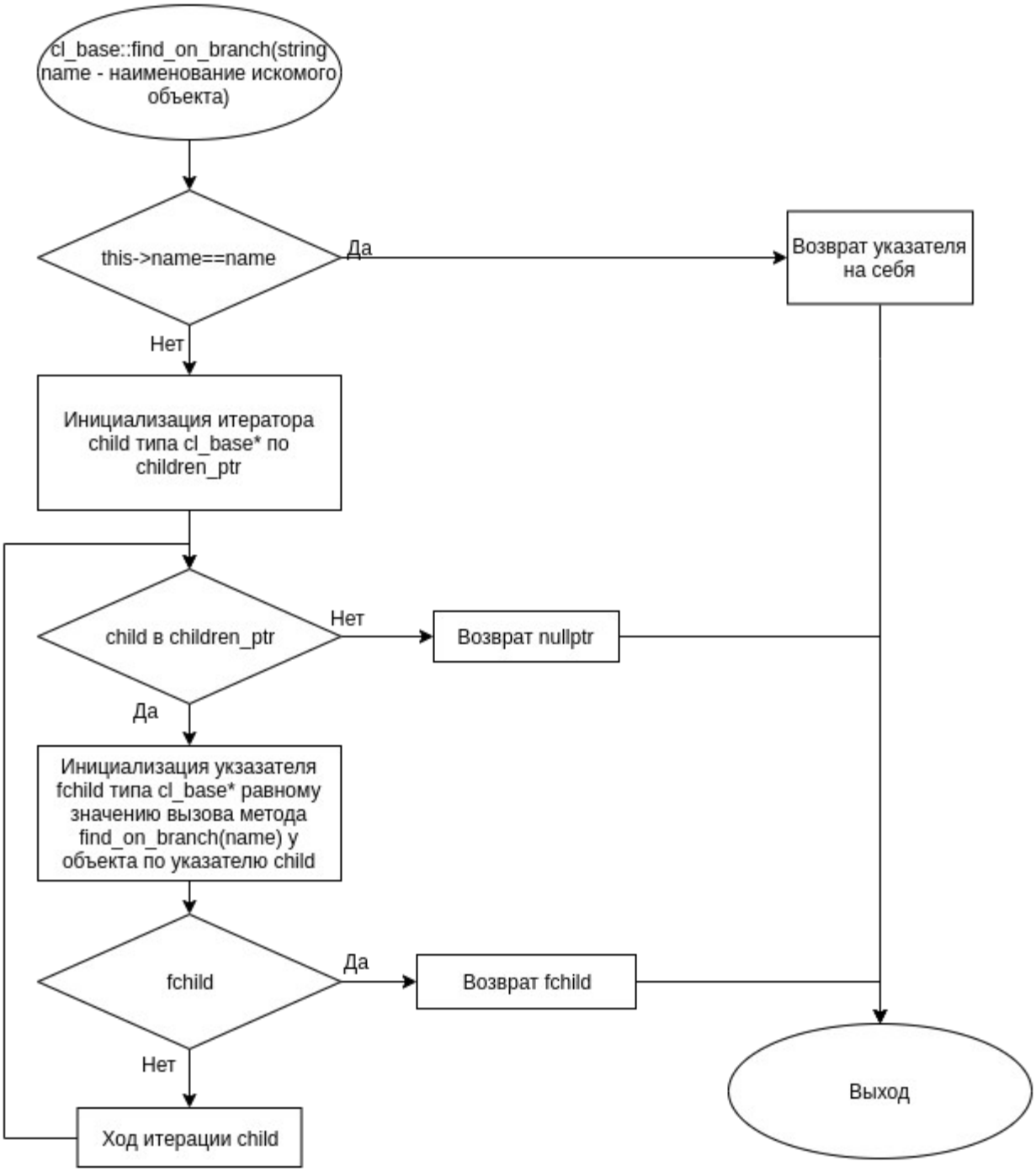

Рисунок 1 – Блок-схема алгоритма

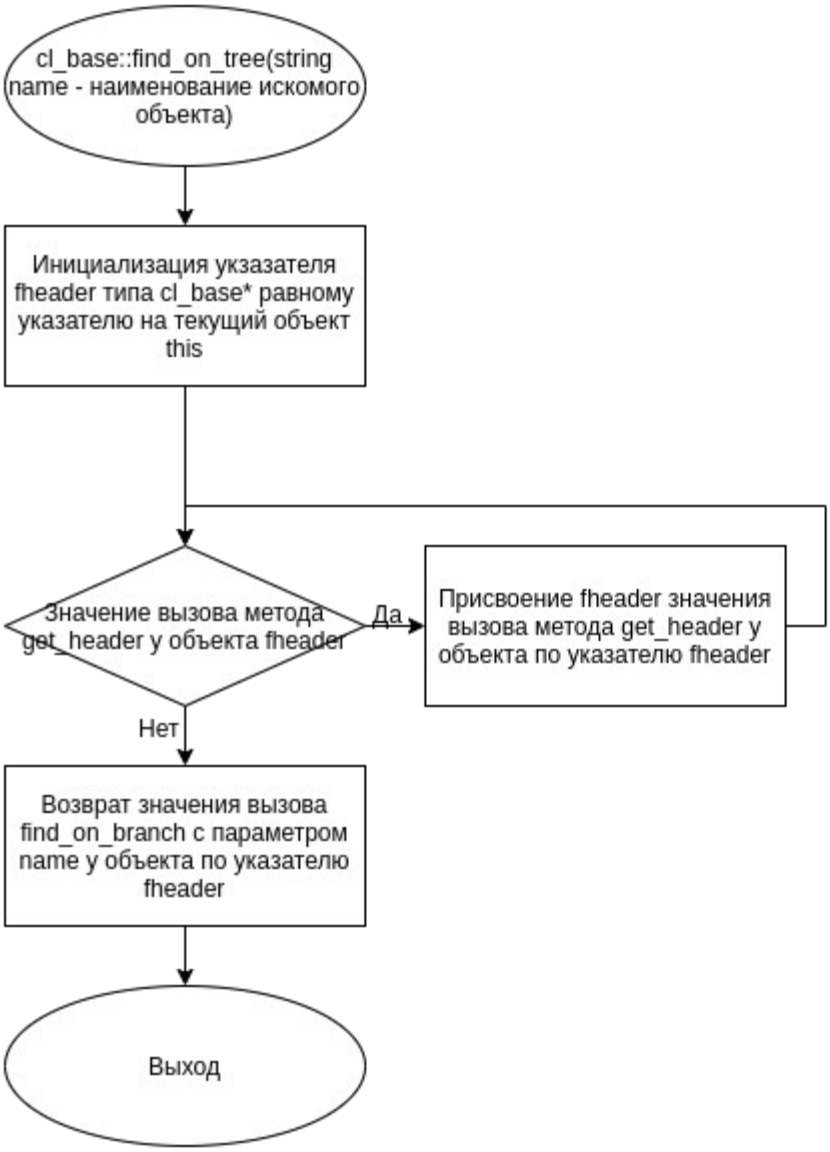

Рисунок 2 – Блок-схема алгоритма

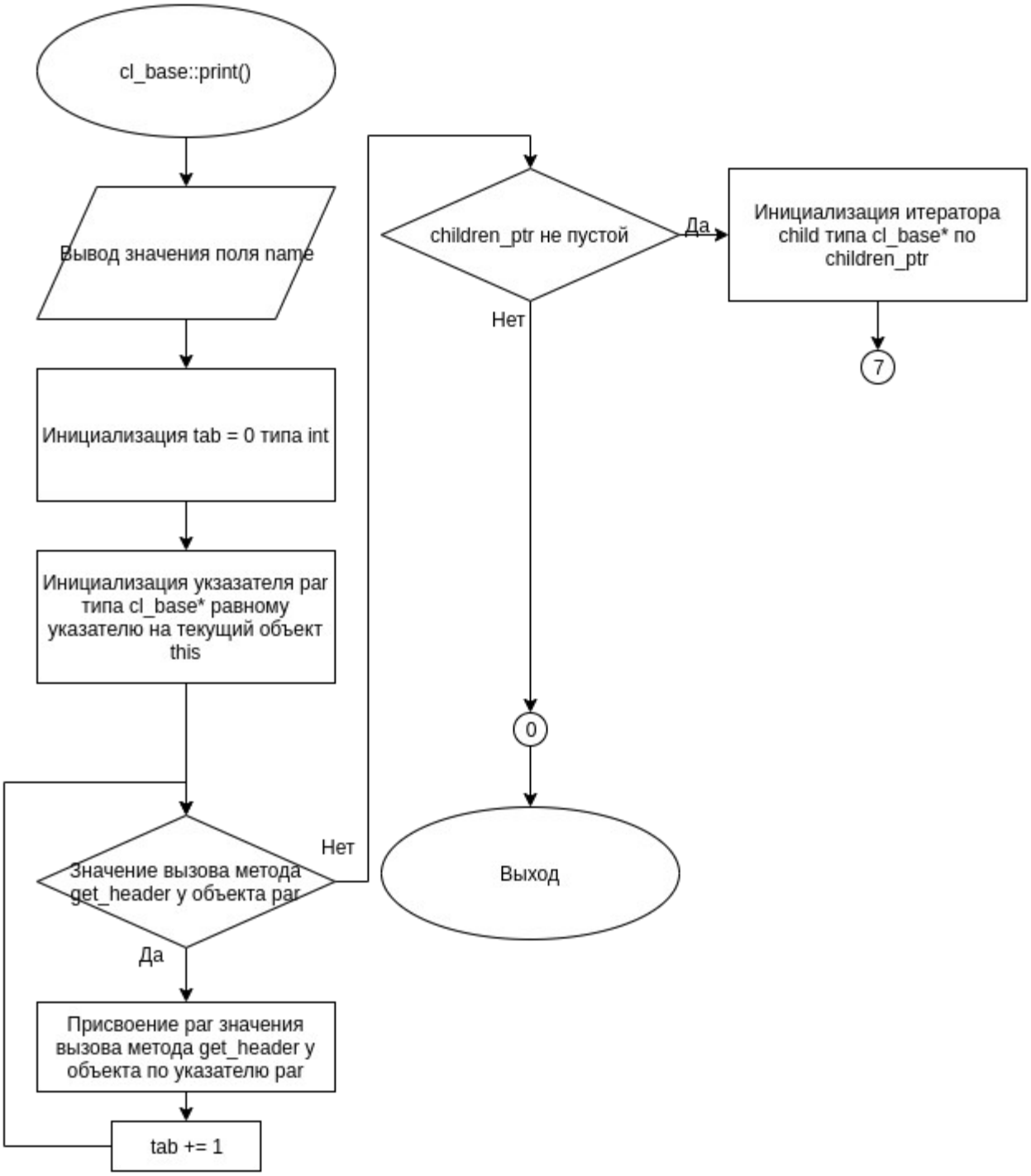

Рисунок 3 – Блок-схема алгоритма

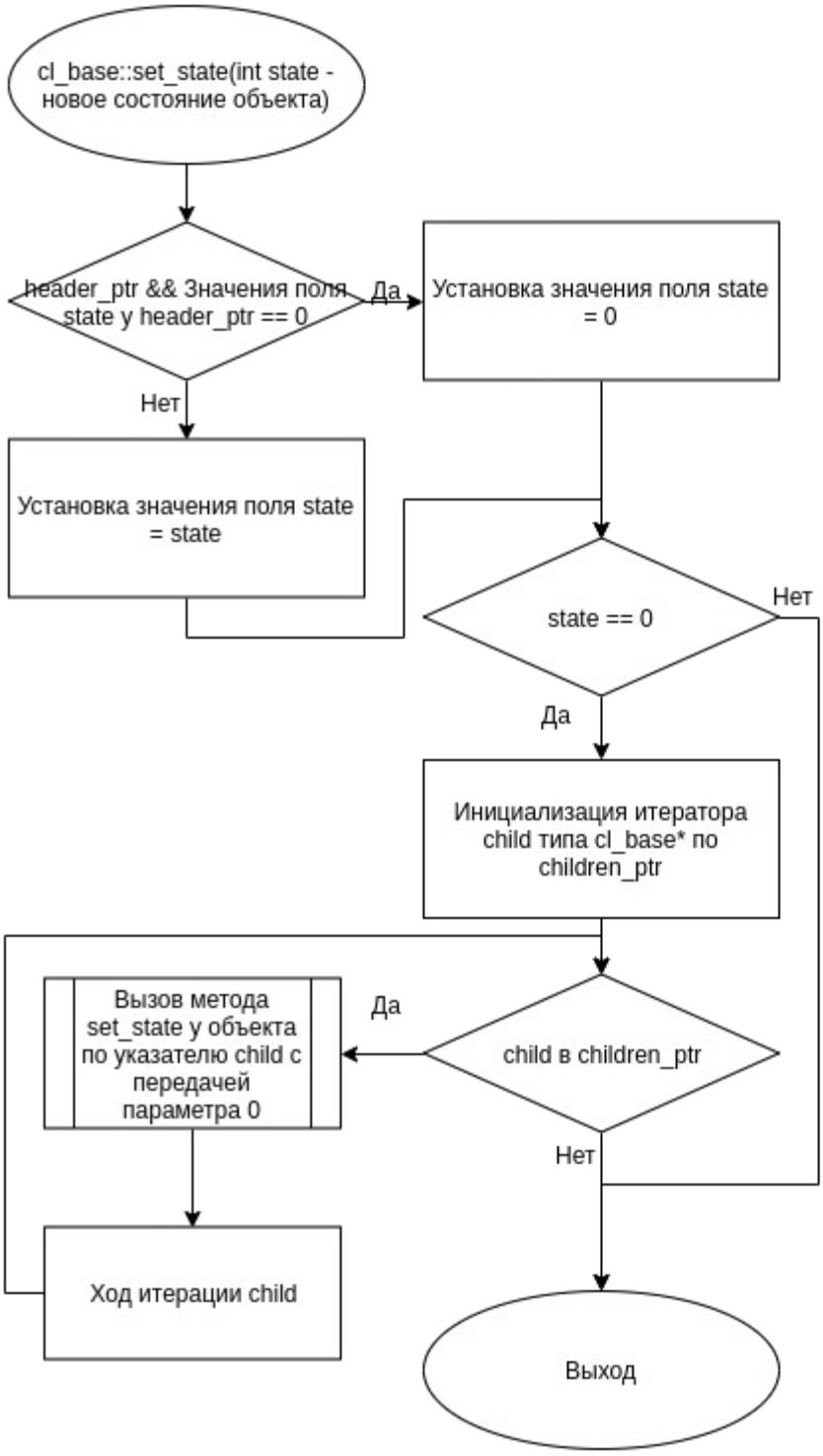

Рисунок 4 – Блок-схема алгоритма

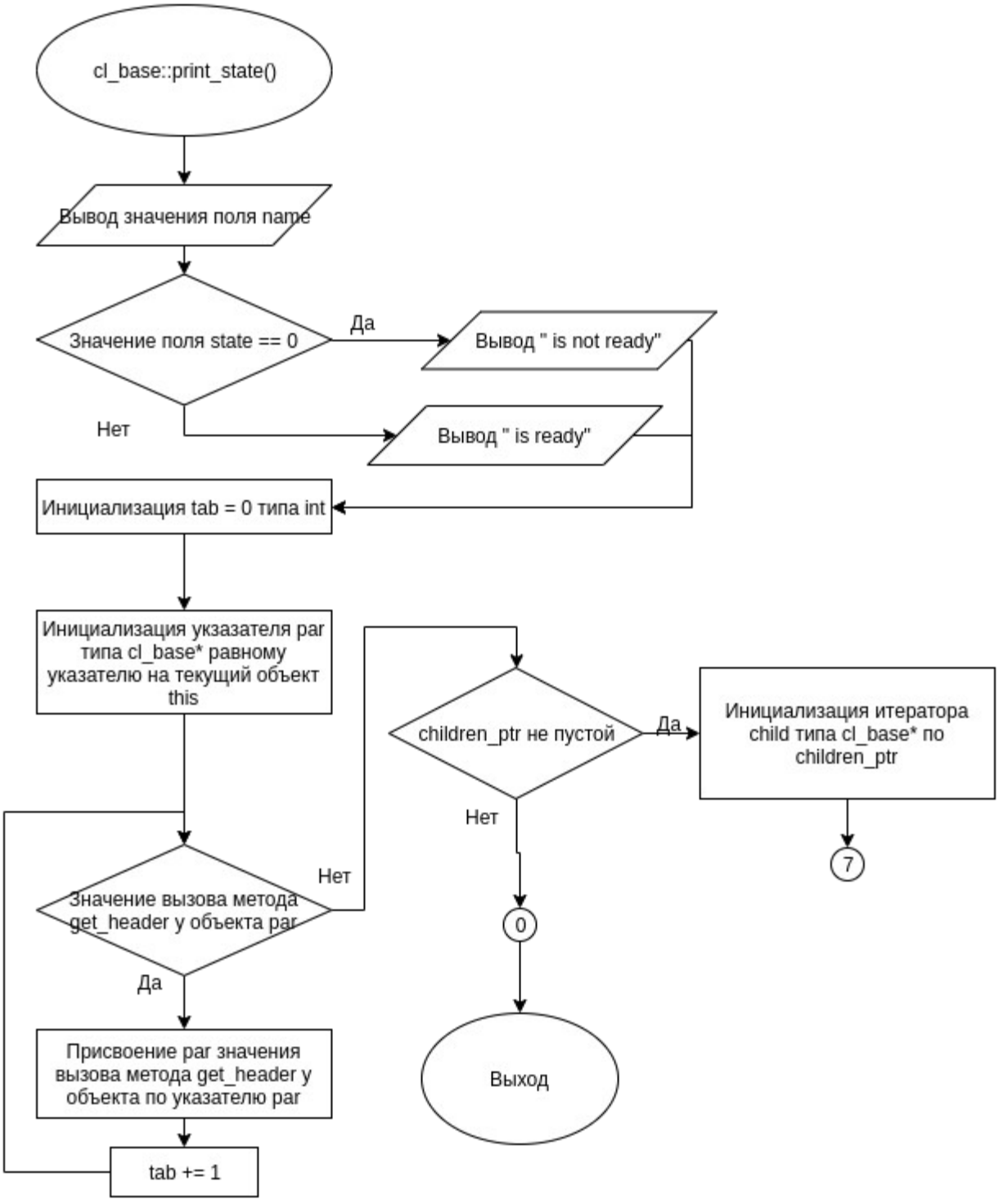

Рисунок 5 – Блок-схема алгоритма

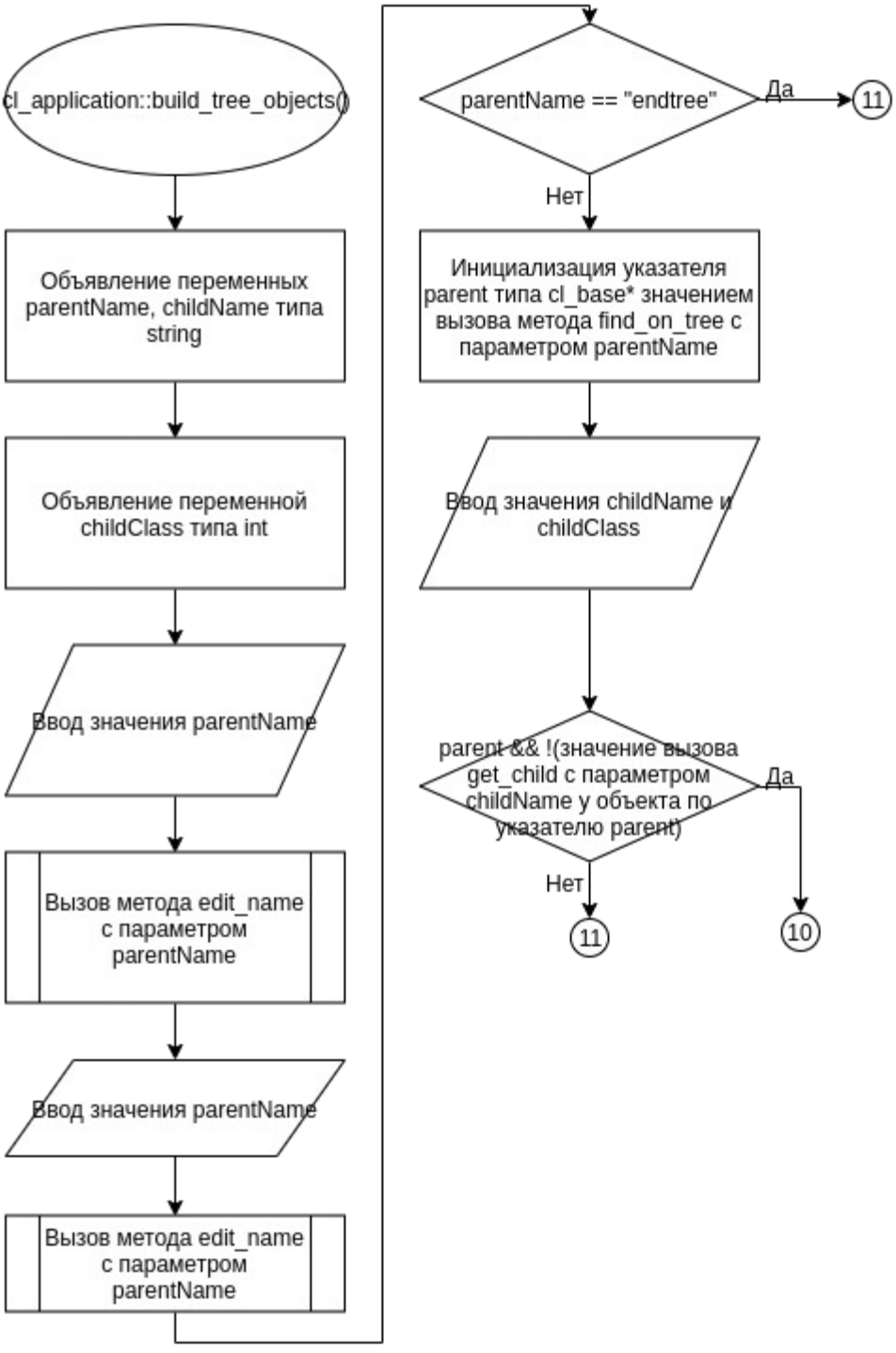

Рисунок 6 – Блок-схема алгоритма

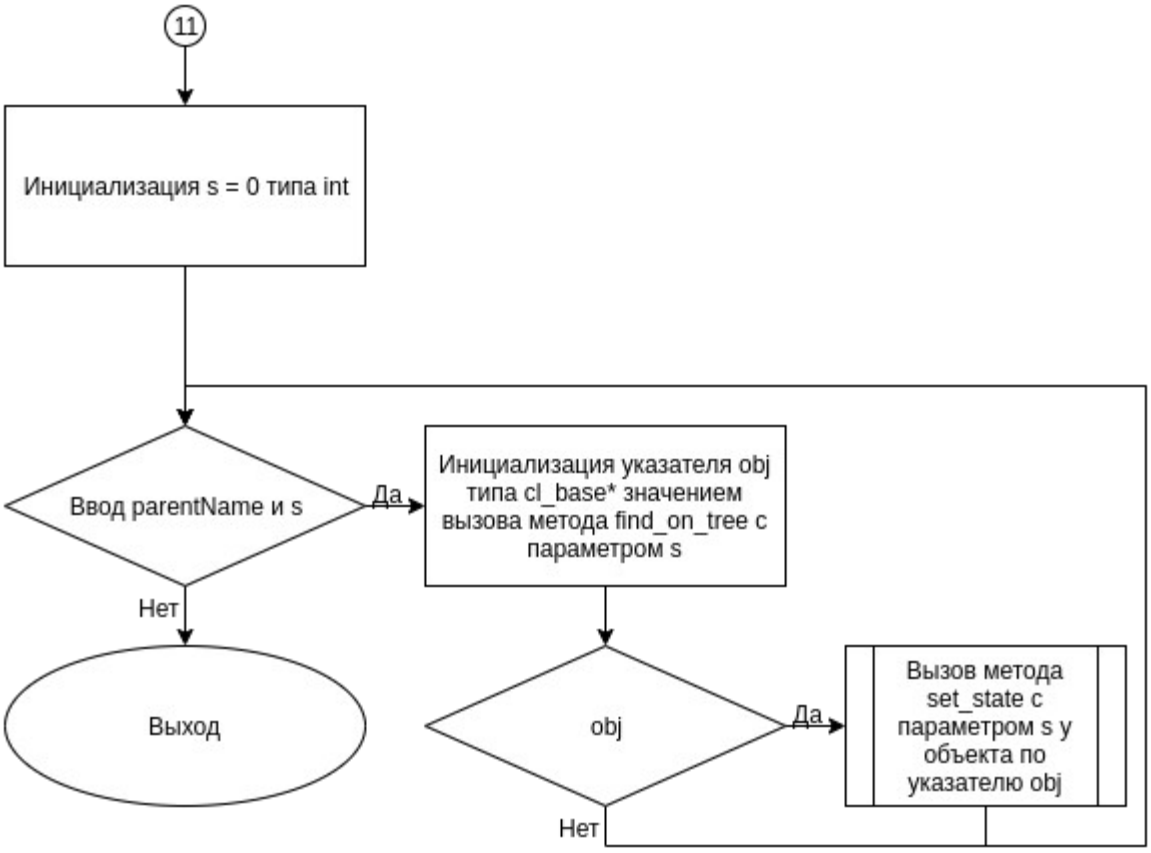

Рисунок 7 – Блок-схема алгоритма

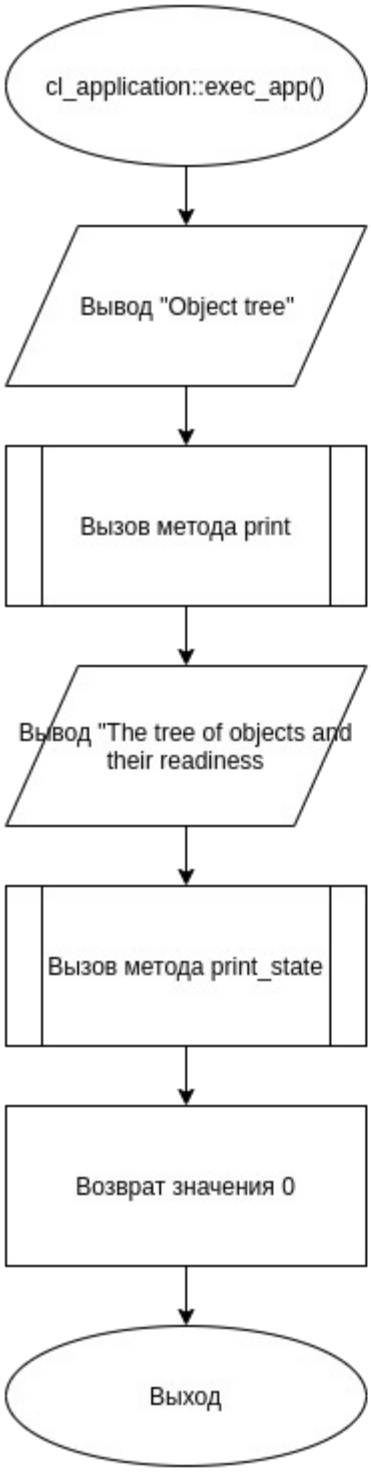

Рисунок 8 – Блок-схема алгоритма

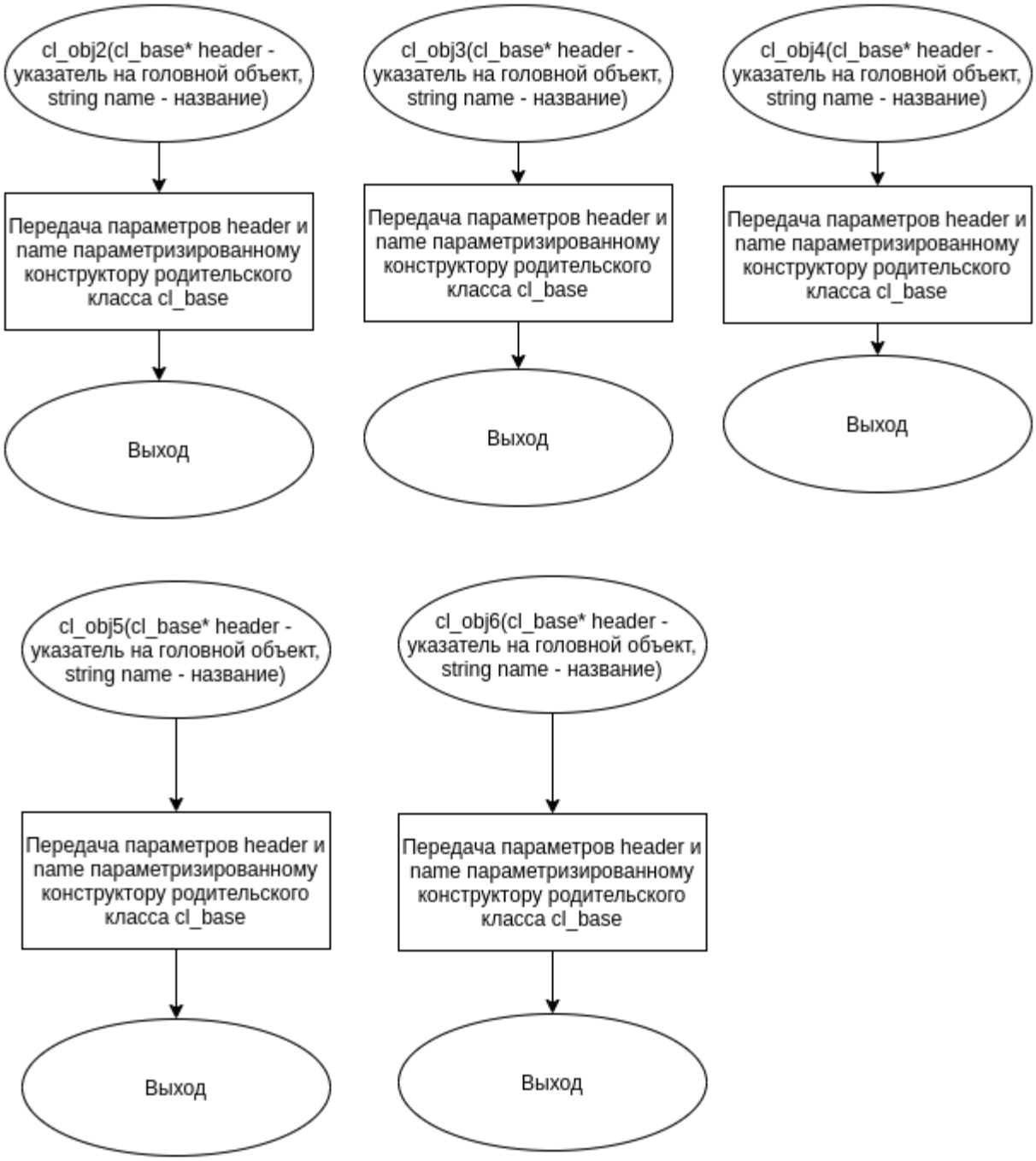

Рисунок 9 – Блок-схема алгоритма

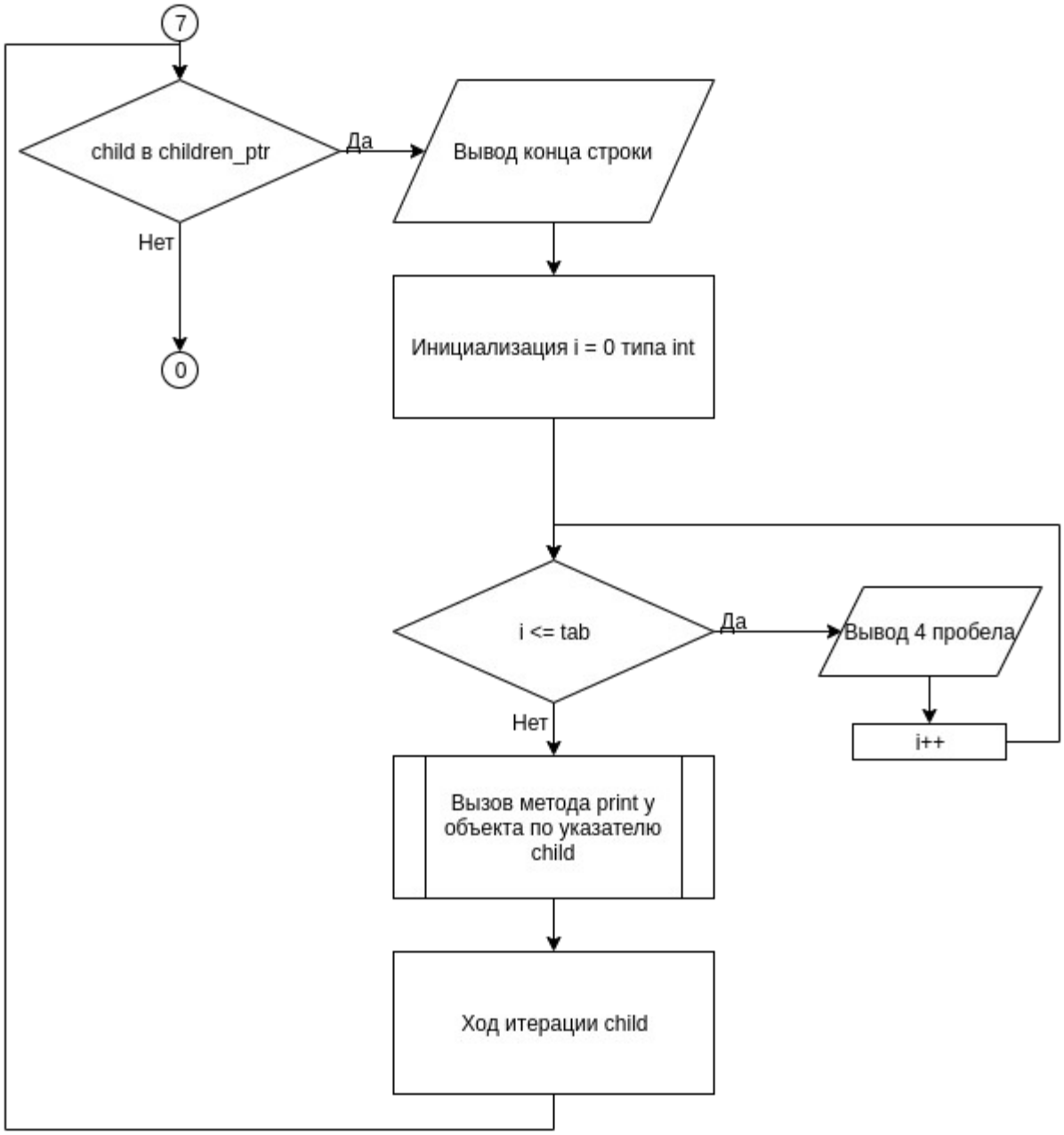

Рисунок 10 – Блок-схема алгоритма

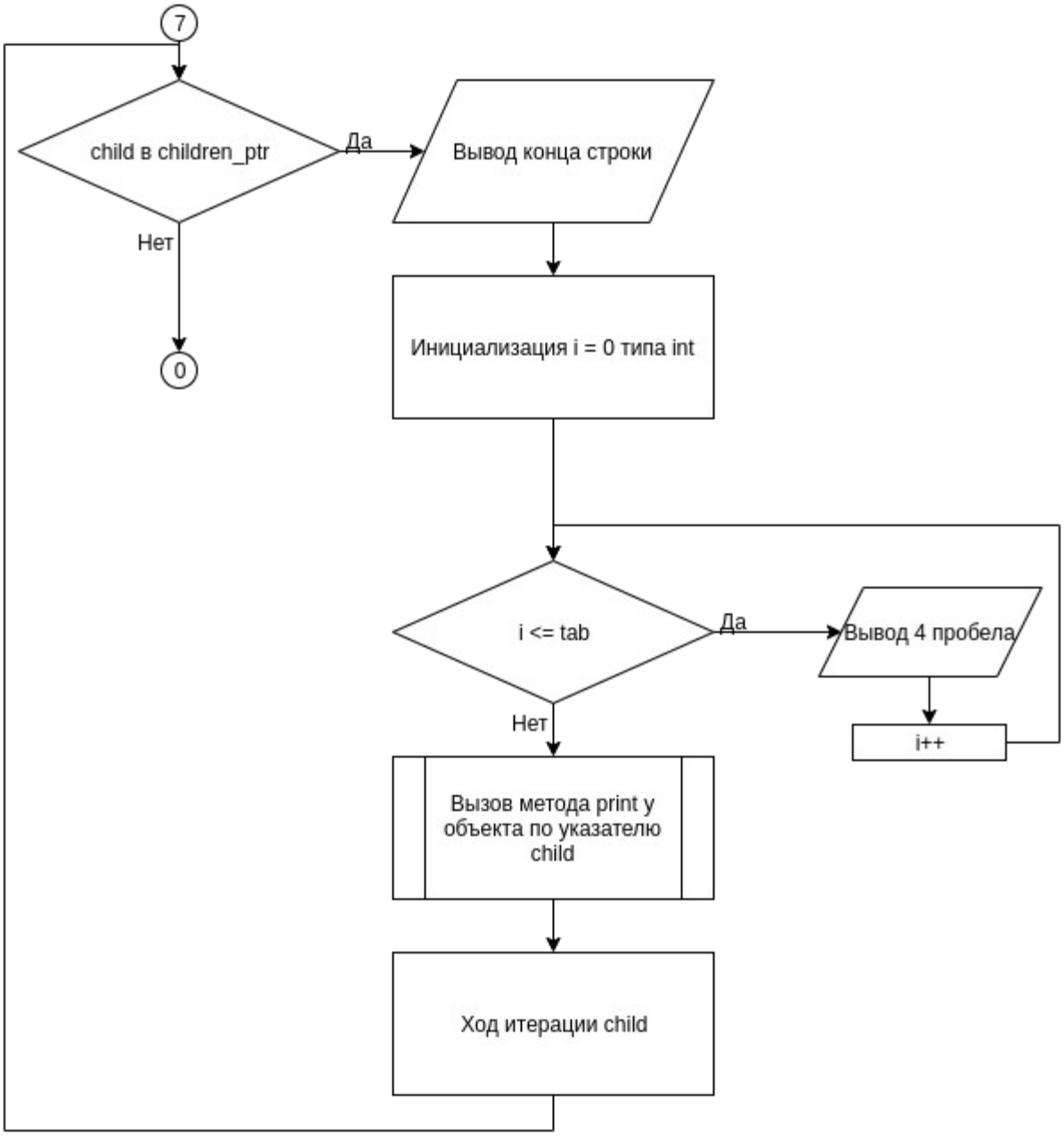

Рисунок 11 – Блок-схема алгоритма

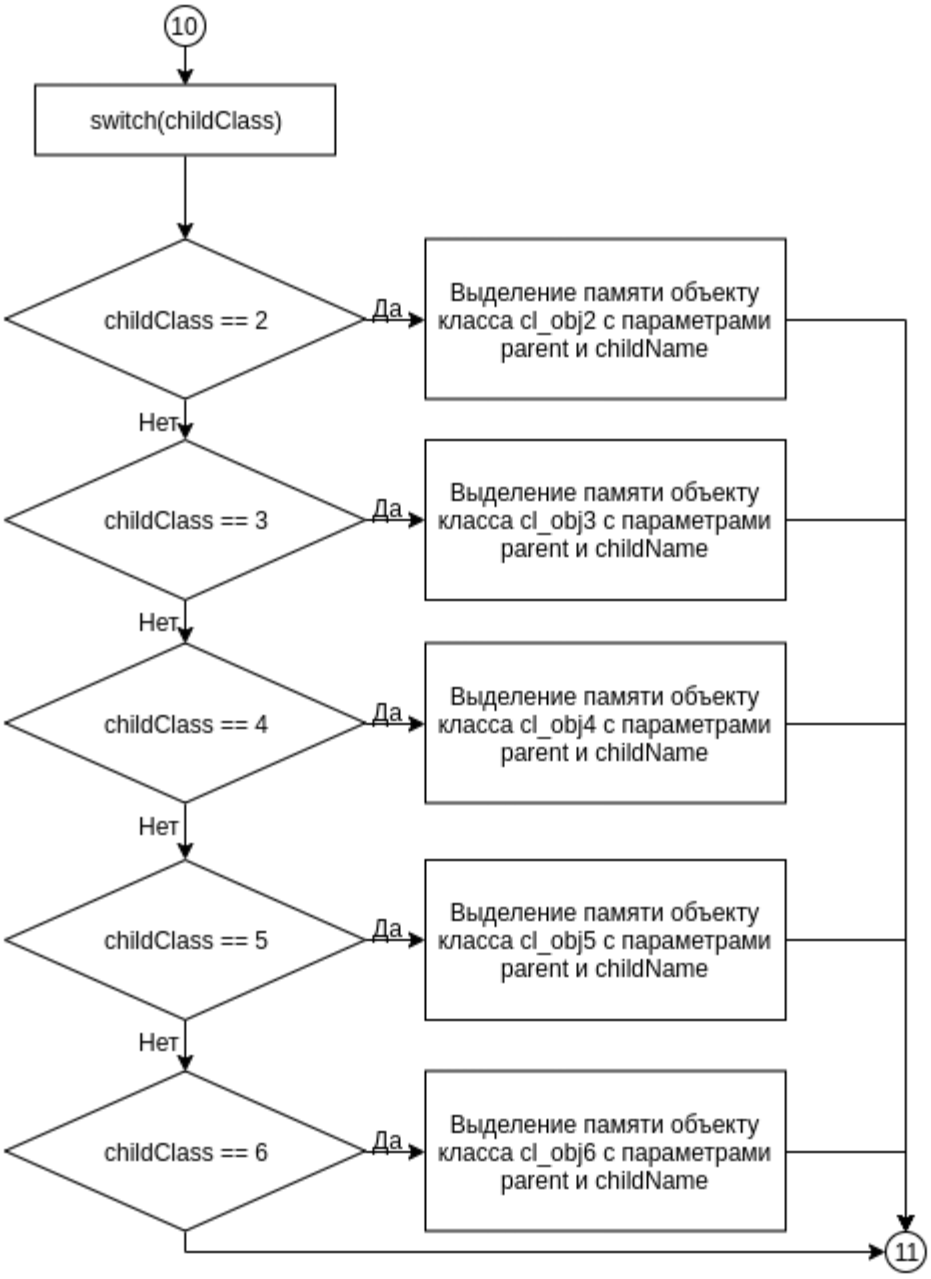

Рисунок 12 – Блок-схема алгоритма

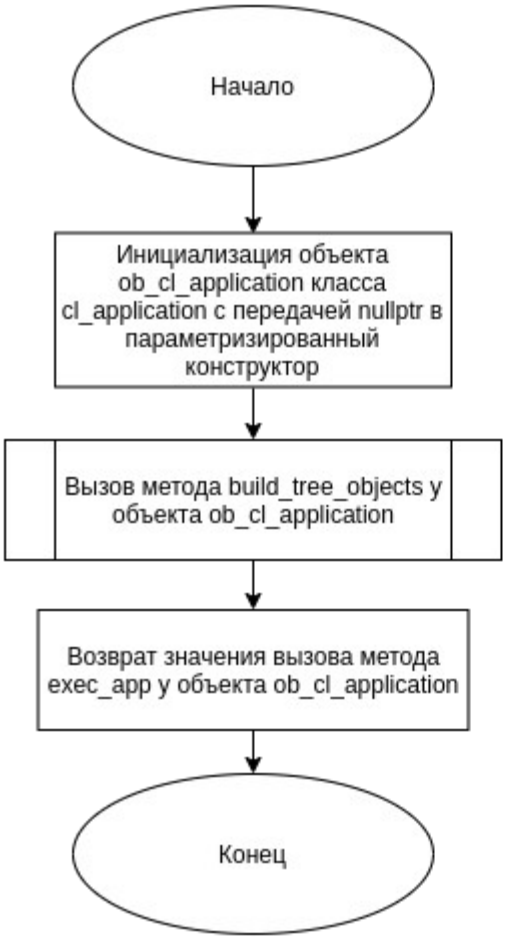

Рисунок 13 – Блок-схема алгоритма

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* header) : cl_base(header) {};
void cl_application::build_tree_objects() {
   string parentName, childName;
   int childClass;

   cin >> parentName;
   this->edit_name(parentName);
   while(true)
   {
      cin >> parentName;
      if(parentName == "endtree") break;
      cl_base* parent = find_on_tree(parentName);
      cin >> childName >> childClass;

      if(parent && !parent->get_child(childName))
      {
         switch(childClass)
         {
            case 2:
            {
               new cl_obj2(parent, childName);
               break;
            }
            case 3:
            {
               new cl_obj3(parent, childName);
               break;
            }
            case 4:
            {
               new cl_obj4(parent, childName);
               break;
            }
            case 5:
            {
               new cl_obj5(parent, childName);
```

```
                break;
             }
             case 6:
             {
                new cl_obj6(parent, childName);
                break;
             }
          }
       }
    }
    int s = 0;
    while(cin >> parentName >> s)
    {
       cl_base* obj = find_on_tree(parentName);
       if(obj) obj->set_state(s);
    }
};
int cl_application::exec_app()
{
   cout << "Object tree" << endl;
   print();
   cout << endl << "The tree of objects and their readiness" << endl;
   print_state();
   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 "cl_obj2.h"
#include "cl_obj3.h"
#include "cl_obj4.h"
#include "cl_obj5.h"
#include "cl_obj6.h"

class cl_application : public cl_base {
public:
   cl_application(cl_base* header);
   void build_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* header, string name){
   this->name = name;
   this->header_ptr = header;
   if(header_ptr){
      header_ptr-> children_ptr.push_back(this);
   }
}

bool cl_base::edit_name(string new_name){
   this->name = new_name;
   return true;
}

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

cl_base* cl_base::get_header(){
   return this->header_ptr;
}

cl_base* cl_base::get_child(string name){
   for(cl_base* child : children_ptr) {
      if(child->name == name) return child;
   }
   return nullptr;
}



cl_base* cl_base::find_on_branch(string name) {
   if(this->name == name){
      return this;
   }

   for(cl_base* child : children_ptr) {
      cl_base* fchild = child->find_on_branch(name);
      if(fchild) {
         return fchild;
      }
   }
   return nullptr;
}


cl_base* cl_base::find_on_tree(string name) {
   cl_base* fheader = this;
```

```
      while(fheader->get_header()){
         fheader = fheader->get_header();
      }
      return fheader->find_on_branch(name);
   }


   void cl_base::print() {
      cout << this->name;

      int tab = 0;
      cl_base* par = this;
      while(par->get_header()){
         par = par->get_header();
         tab += 1;
      }


      if(!children_ptr.empty())
      {
         for(cl_base* child : children_ptr) {
            cout << endl;
            for(int i = 0; i <= tab; i++) cout << "    ";
            child->print();
         }
      }
   }


   void cl_base::print_state() {
      cout << this->name;
      if(this->state == 0) cout << " is not ready";
      else cout << " is ready";

      int tab = 0;
      cl_base* par = this;
      while(par->get_header()){
         par = par->get_header();
         tab += 1;
      }


      if(!children_ptr.empty())
      {
         for(cl_base* child : children_ptr) {
            cout << endl;
            for(int i = 0; i <= tab; i++) cout << "    ";
            child->print_state();
         }
      }
   }

   void cl_base::set_state(int state) {
      if(header_ptr && header_ptr->state == 0) this->state = 0;
      else this->state = state;
```

```
   if(state == 0) {
      for(cl_base* child : children_ptr){
         child->set_state(0);
      }
   }
}
```

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>


using namespace std;

class cl_base {
protected:
   string name;
   cl_base *header_ptr;
   vector<cl_base*> children_ptr;

   int state;
public:
   cl_base(cl_base* base, string name = "Object");
   bool edit_name(string new_name);
   string get_name();
   cl_base* get_header();
   cl_base* get_child(string name);

   cl_base* find_on_branch(string name); // Поиск на ветке
   cl_base* find_on_tree(string name); // Поиск на всем дереве

   void print();
   void print_state();

   void set_state(int state);

};

#endif
```

5.5 Файл cl_obj2.cpp

Листинг 5 – cl_obj2.cpp

```
#include "cl_obj2.h"

cl_obj2::cl_obj2(cl_base* header, string name) : cl_base(header, name) {}
```

5.6 Файл cl_obj2.h

Листинг 6 – cl_obj2.h

```
#ifndef __CL_OBJ2__H
#define __CL_OBJ2__H
#include "cl_base.h"

class cl_obj2 : public cl_base
{
public:
   cl_obj2(cl_base* header, string name);
};

#endif
```

5.7 Файл cl_obj3.cpp

Листинг 7 – cl_obj3.cpp

```
#include "cl_obj3.h"

cl_obj3::cl_obj3(cl_base* header, string name) : cl_base(header, name) {}
```

5.8 Файл cl_obj3.h

Листинг 8 – cl_obj3.h

```
#ifndef __CL_OBJ3__H
#define __CL_OBJ3__H
```

```
#include "cl_base.h"

class cl_obj3 : public cl_base
{
public:
   cl_obj3(cl_base* header, string name);
};

#endif
```

5.9 Файл cl_obj4.cpp

Листинг 9 – cl_obj4.cpp

```
#include "cl_obj4.h"

cl_obj4::cl_obj4(cl_base* header, string name) : cl_base(header, name) {}
```

5.10 Файл cl_obj4.h

Листинг 10 – cl_obj4.h

```
#ifndef __CL_OBJ4__H
#define __CL_OBJ4__H
#include "cl_base.h"

class cl_obj4 : public cl_base
{
public:
   cl_obj4(cl_base* header, string name);
};

#endif
```

5.11 Файл cl_obj5.cpp

Листинг 11 – cl_obj5.cpp

```
#include "cl_obj5.h"
```

```
cl_obj5::cl_obj5(cl_base* header, string name) : cl_base(header, name) {}
```

5.12 Файл cl_obj5.h

Листинг 12 – cl_obj5.h

```
#ifndef __CL_OBJ5__H
#define __CL_OBJ5__H
#include "cl_base.h"

class cl_obj5 : public cl_base
{
public:
   cl_obj5(cl_base* header, string name);
};


#endif
```

5.13 Файл cl_obj6.cpp

Листинг 13 – cl_obj6.cpp

```
#include "cl_obj6.h"

cl_obj6::cl_obj6(cl_base* header, string name) : cl_base(header, name) {}
```

5.14 Файл cl_obj6.h

Листинг 14 – cl_obj6.h

```
#ifndef __CL_OBJ6__H
#define __CL_OBJ6__H
#include "cl_base.h"

class cl_obj6 : public cl_base
{
```

```
public:
   cl_obj6(cl_base* header, string name);
};



#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 15.

Таблица 15 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
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	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	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

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).