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.

Каждый объект на дереве иерархии имеет свое место и наименование. Не допускается для одного головного объекта одинаковые наименования в составе подчиненных 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 nullptr);
  - динамический массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.
- функционал:
  - параметризированный конструктор с параметрами: указатель на объект базового класса, содержащий адрес головного объекта в дереве иерархии; строкового типа, содержащий наименование создаваемого объекта (имеет значение по умолчанию);
  - метод редактирования имени объекта. Один параметр строкового типа, содержит новое наименование объекта. Если нет дубляжа имени подчиненных объектов у головного, то редактирует имя и возвращает «истину», иначе возвращает «ложь»;
  - метод получения имени объекта;

- метод получения указателя на головной объект текущего объекта;
- метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- метод получения указателя на непосредственно подчиненный объект по его имени. Если объект не найден, то возвращает nullptr. Один параметр строкового типа, содержит наименование искомого подчиненного 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 моделируемой системы);
- 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. Исключить создание объекта если его наименование совпадает с именем уже имеющегося подчиненного объекта у предполагаемого головного. Исключить добавление нового объекта, не последнему подчиненному предыдущего уровня.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

```
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
}
```

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

Все версии курсовой работы имеют такую основную функцию.

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.

Пример ввода:

```
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 решения задачи используется:

- объекты стандартного потока ввода/вывода данных cin/cout;
- условный оператор if...else;
- оператор цикла с преусловием while;
- оператор цикла со счетчиком for;
- контейнер vector из STL;
- объект-приложение clApp из класса ClApplication;
- объекты класса cl_1, имена и количество которых задаются полльзователем.

Класс cl_base:

- свойства/поля:
  - поле указатель на головной объект:
    - наименование — p_head_object;
    - тип — указатель на объект класса cl_base;
    - модификатор доступа — private;
  - поле название объекта:
    - наименование — s_name;
    - тип — строка;
    - модификатор доступа — private;
  - поле массив подчиненных объектов:
    - наименование — p_sub_objects;
    - тип — вектор указателей на объекты класса cl_base;
    - модификатор доступа — private;
- функционал:
  - метод cl_base — параметризированный конструктор с 2

параметрами: указателем на головной объект и строкой - наименованием создаваемого объекта. Добавляет объект в массив подчиненных объектов головного, если сам таковым не является;

- метод ~cl_base — деструктор, удаляет подчиненных объектов, освобождает память;
- метод print_tree — вывод дерева объектов слева направо сверху вниз;
- метод get_head — получение указателя на головной объект;
- метод get_name — получение названия объекта;
- метод set_name — задание объекту нового имени;
- метод get_sub_object — получение указателя на дочерний объект по его названию.

Класс ClApplication:

- функционал:
  - метод ClApllication — параметризированный конструктор с 1 параметром: указателем на головной объект;
  - метод build_tree_objects — построение дерева объектов;
  - метод exec_app — запуск приложения, выводит в консоль дерево иерархии посредством метода print_tree.

Класс cl_1:

- функционал:
  - метод cl_1 — параметризированный конструктор с 2 параметрами: указателем на головной объект и строкой - наименованием создаваемого объекта.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			базовый класс, содержащий поля и методы для производных классов	
		ClApplication	public		2
2	ClApplication			Класс объекта-приложения	
3	cl_1			Класс объекта дерева иерархии	

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_base

Функционал: параметризированный конструктор с 2 параметрами: указателем на головной объект и строкой - наименованием создаваемого объекта. Добавляет объект в массив подчиненных объектов головного, если сам таковым не является.

Параметры: указатель на p_head_object класса cl_base, строка s_name - название объекта.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Присвоение указателю p_head_object создаваемого объекта значения параметра p_head_object	2
2		Присвоение полю s_name создаваемого объекта значения параметра s_name	3
3	Указатель на головной объект не равен nullptr?	Добавление указателя на объект в массив указателей на дочерние объекты родительского	∅
			∅

3.2 Алгоритм деструктора класса cl_base

Функционал: деструктор, удаляет подчиненных объектов, освобождает

память.

Параметры: указатель на p_head_object класса cl_base, строка s_name - название объекта.

Алгоритм деструктора представлен в таблице 3.

Таблица 3 – Алгоритм деструктора класса cl_base

№	Предикат	Действия	№ перехода
1		Инициализация счетчика i=0	2
2	i меньше размера p_sub_objects?	Удаление i-того элемента вектора	3
			∅
3		Увеличение i на 1	∅

3.3 Алгоритм метода print_tree класса cl_base

Функционал: вывод дерева объектов слева направо сверху вниз.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода
1	Объект имеет дочерние объекты	Объявление и инициализация счетчика i = 0	2
			∅
2		Вывод имени объекта с новой строки	3
3	i меньше размера p_sub_objects?	Вывод имени объекта p_sub_objects[i]	4
			∅
4		Вызов метода print_tree для i-того дочернего	5

№	Предикат	Действия	№ перехода
		объекта	
5		Увеличение i на 1	∅

3.4 Алгоритм метода get_head класса cl_base

Функционал: получение указателя на головной объект.

Параметры: нет.

Возвращаемое значение: указатель p_head_object.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_head класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля p_head_object	∅

3.5 Алгоритм метода get_name класса cl_base

Функционал: получение названия объекта.

Параметры: нет.

Возвращаемое значение: строка s_name.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода
1		Возврат значения поля p_head_object	∅

3.6 Алгоритм метода set_name класса cl_base

Функционал: задание объекту нового имени.

Параметры: строка s_name.

Возвращаемое значение: true, если имя было изменено; false, если нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода set_name класса cl_base

№	Предикат	Действия	№ перехода
1	Указатель на головной объект не равен nullptr?	Объявление и инициализация счетчика i = 0	2
			∅
2	i меньше размера p_sub_objects		3
			4
3	s_name совпадает с именем p_sub_objects[i]	Возврат false	∅
		Увеличение i на 1	2
4		Присвоение полю s_name значения параметра s_name	5
5		Возврат true	∅

3.7 Алгоритм метода get_sub_object класса cl_base

Функционал: получение указателя на дочерний объект по его названию.

Параметры: строка s_name.

Возвращаемое значение: указатель на дочерний объект.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода get_sub_object класса cl_base

№	Предикат	Действия	№ перехода
1		Объявление и инициализация счетчика i = 0	2
2	i меньше размера p_sub_objects		3

№	Предикат	Действия	№ перехода
		Возврат nullptr	∅
3	Параметр s_name совпадает с именем p_sub_objects[i]	Возврат p_sub_objects[i]	∅
		Увеличение i на 1	2

3.8 Алгоритм метода build_tree_objects класса ClApplication

Функционал: построение дерева объектов.

Параметры: нет.

Возвращаемое значение: нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода build_tree_objects класса ClApplication

№	Предикат	Действия	№ перехода
1		Объявление строковых переменных s_sub_name и s_head_name	2
2		Объявление и инициализация указателей на объекты класса cl_base, p_head, p_sub	3
3		Ввод значения s_head_name с клавиатуры	4
4		Вызов метода set_name для головного объекта с параметром s_head_name	5
5		Ввод значений s_head_name и s_sub_name с клавиатуры	6
6	s_head_name совпадает с s_sub_name		∅
			7
7	s_head_name совпадает с именем p_head и указатель p_sub не нулевой	Присвоение p_head значения указателя p_sub	5

№	Предикат	Действия	№ перехода
			8
8	s_head_name совпадает с именем p_head и у головного объекта нет дочерных с именем s_sub_name	Присвоение p_sub указателя на новый объект класса cl_1, создаваемый конструктором, в качестве параметров в него передаются p_head и s_sub_name	5
			5

3.9 Алгоритм метода exec_app класса ClApplication

Функционал: запуск приложения, выводит в консоль дерево иерархии посредством метода print_tree.

Параметры: нет.

Возвращаемое значение: код возврата метода.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода exec_app класса ClApplication

№	Предикат	Действия	№ перехода
1		Вывод имени текущего объекта	2
2		Вызов метода print_tree	3
3		Возврат значения 0	∅

3.10 Алгоритм функции main

Функционал: главная функция программы.

Параметры: нет.

Возвращаемое значение: целое, идентификатор работоспособности программы.

Алгоритм функции представлен в таблице 11.

Таблица 11 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта clApp класса cl_application	2
2		Вызов метода build_tree_objects объекта clApp	3
3		Возврат результата метода exec_app объекта clApp	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

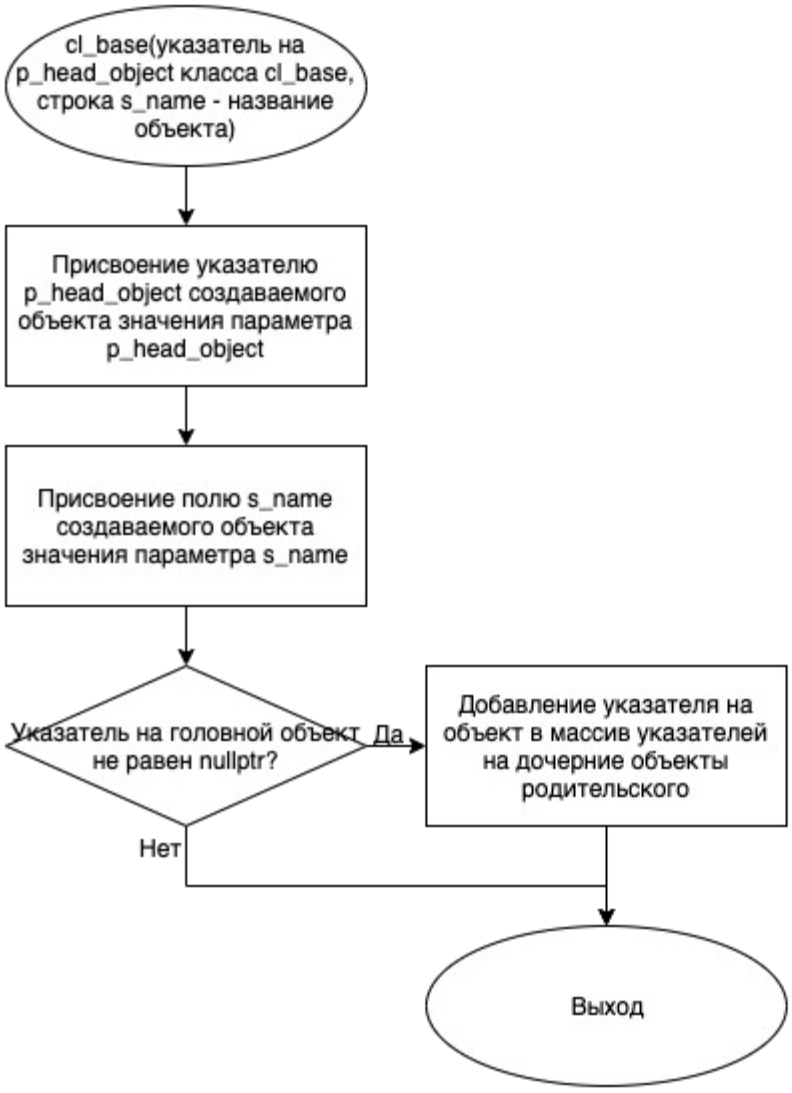


Рисунок 1 – Блок-схема алгоритма

Рисунок 2 – Блок-схема алгоритма

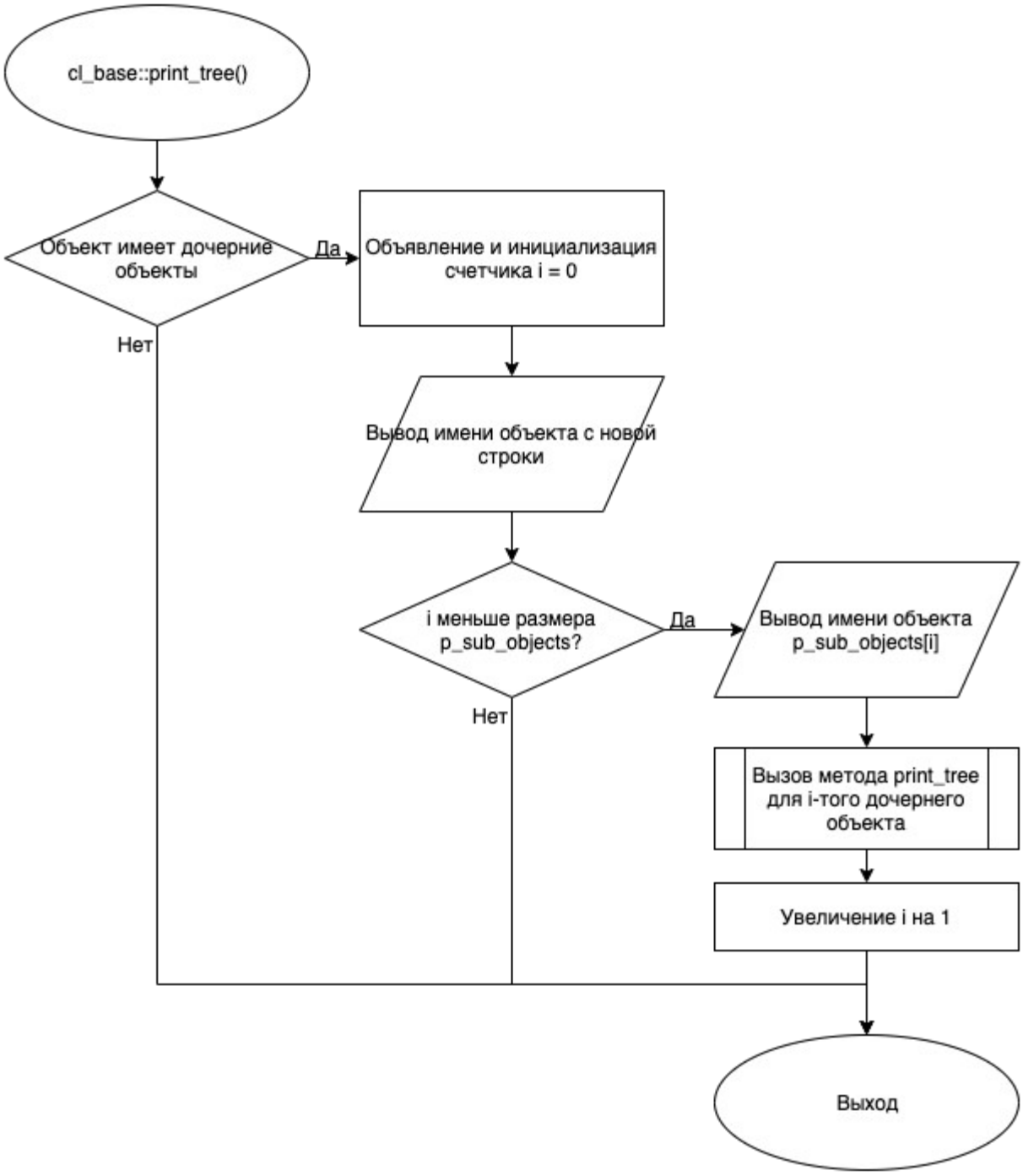


Рисунок 3 – Блок-схема алгоритма

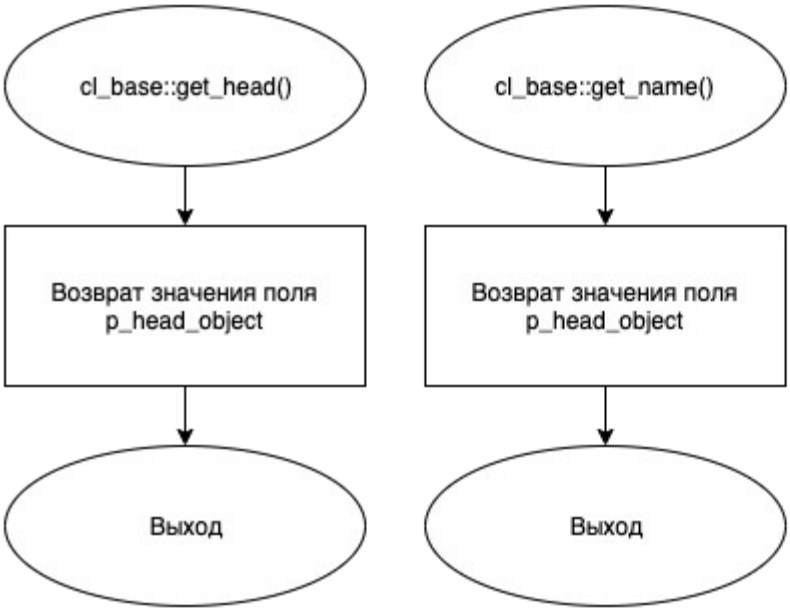


Рисунок 4 – Блок-схема алгоритма

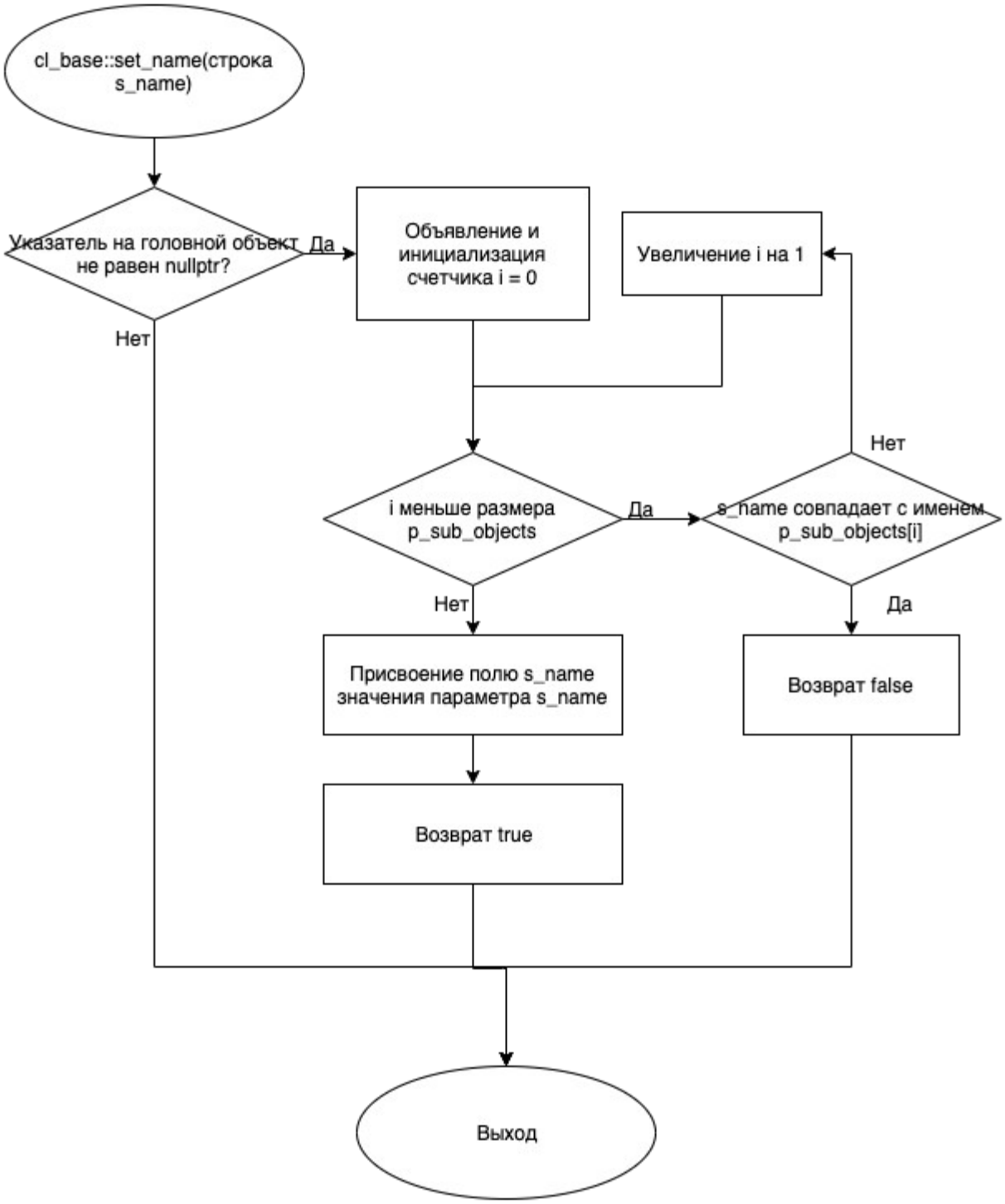


Рисунок 5 – Блок-схема алгоритма

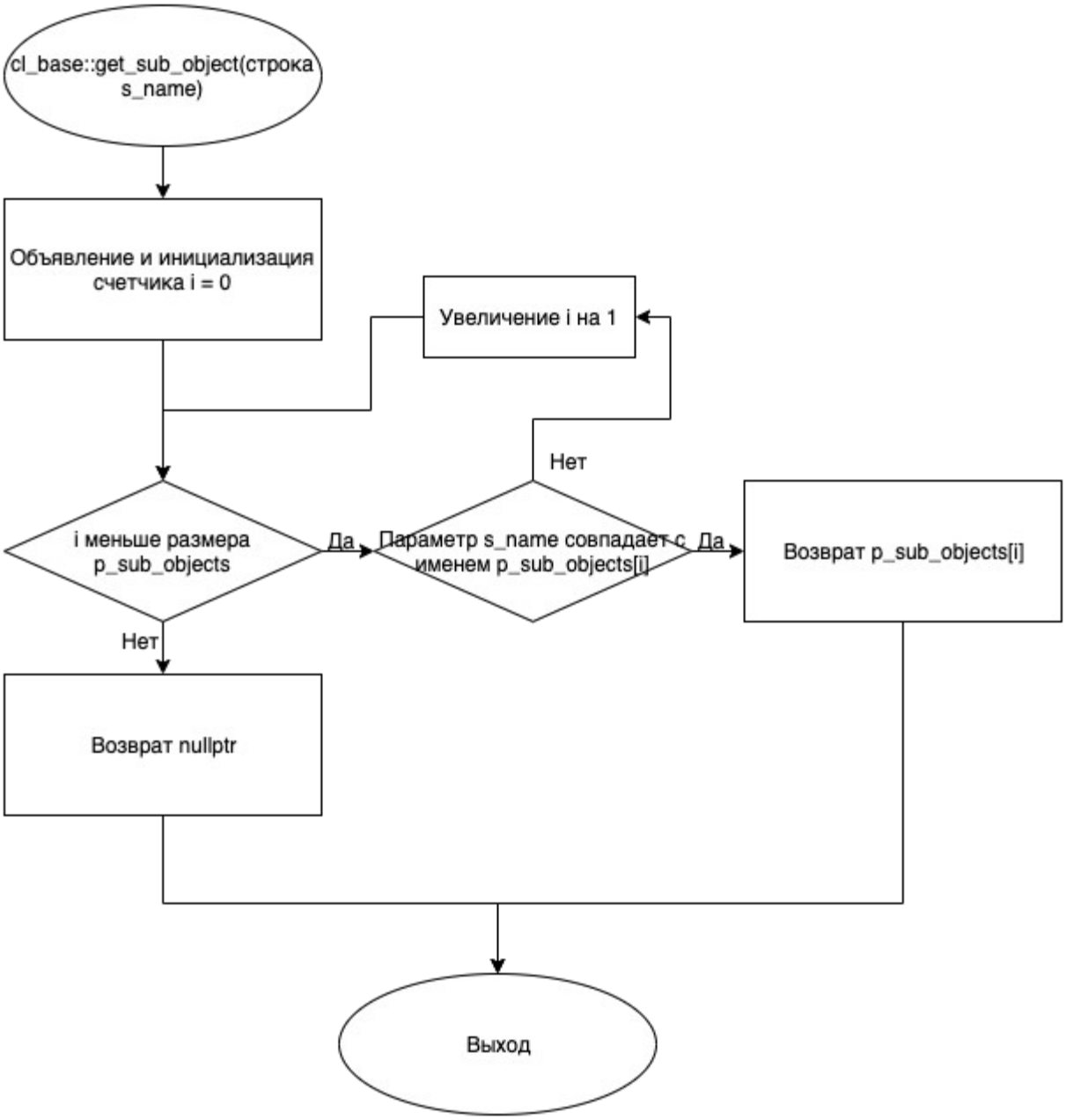


Рисунок 6 – Блок-схема алгоритма

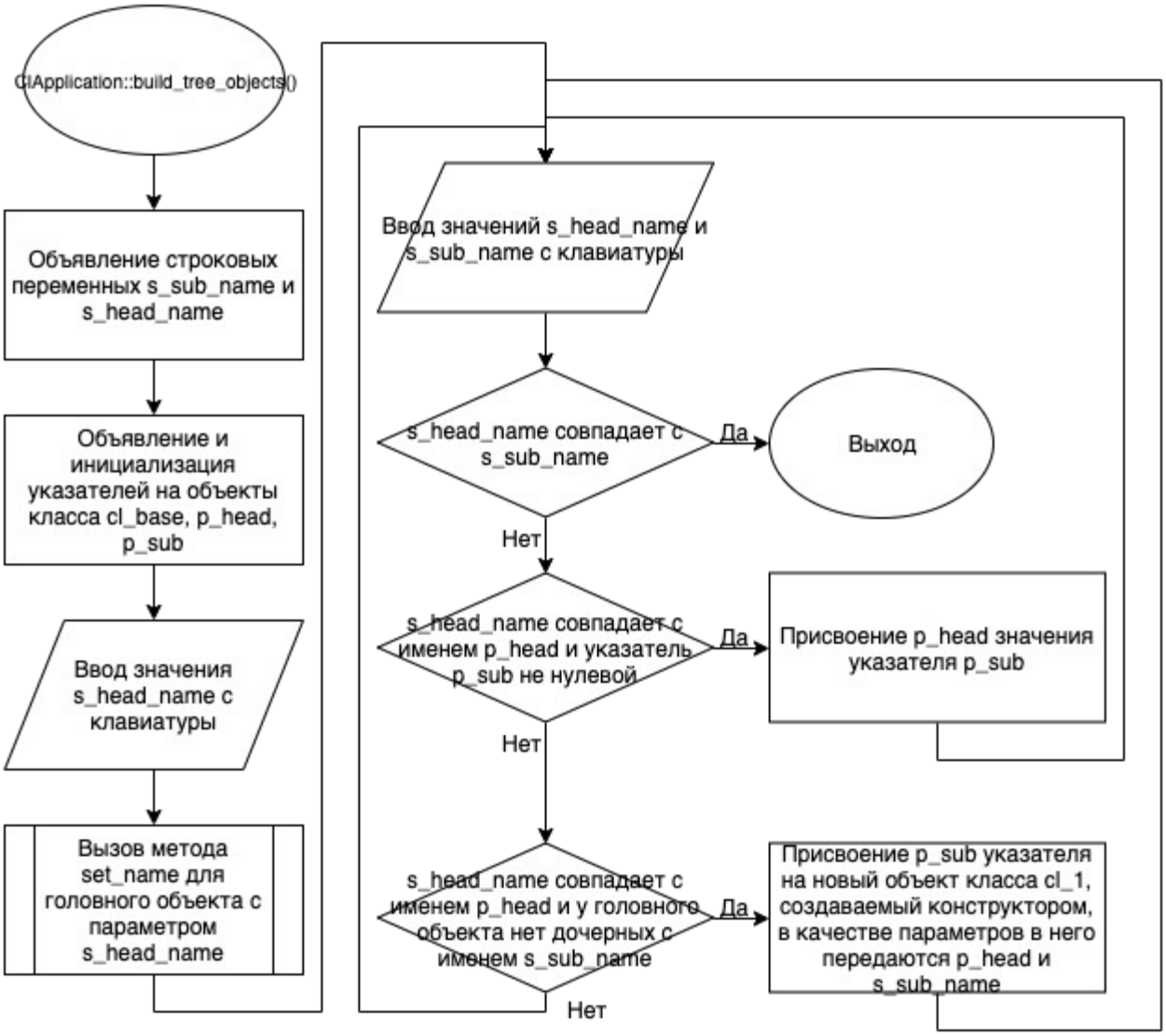
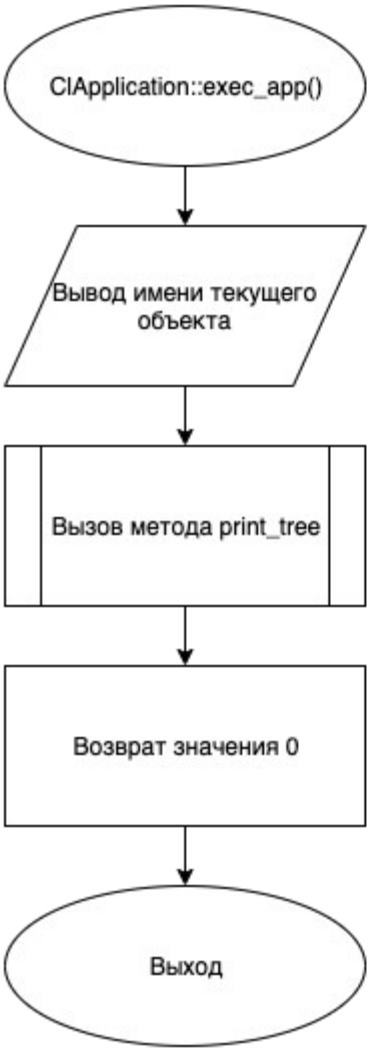


Рисунок 7 – Блок-схема алгоритма



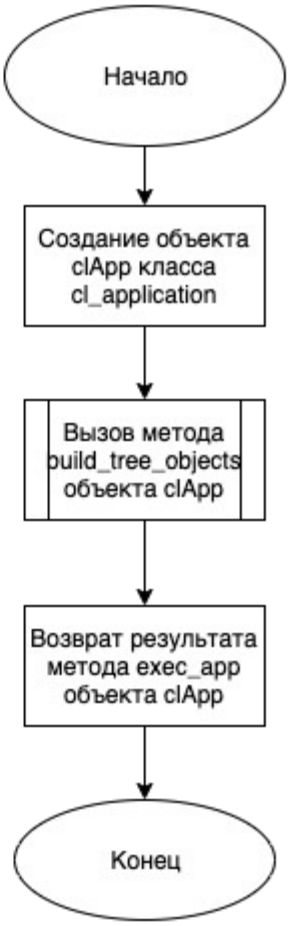


Рисунок 8 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"

cl_1::cl_1(cl_base* p_head_object, string s_name) : cl_base(p_head_object,
s_name){
   // конструктор объекта cl_1
}
```

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_object, string s_name);
};

#endif
```

5.3 Файл cl_application.cpp

Листинг 3 – cl_application.cpp

```
#include "cl_application.h"

cl_application::cl_application(cl_base*           p_head_object)           :
```

```
cl_base(p_head_object){

};

void cl_application::build_tree_objects(){
   string s_sub_name, s_head_name;
   cl_base* p_head = this;
   cl_base* p_sub = nullptr;

   cin >> s_head_name;

   set_name(s_head_name);

   while(true){
      cin >> s_head_name >> s_sub_name;
      if (s_head_name == s_sub_name)
         break;
      if ((p_sub!=nullptr)&&(s_head_name == p_sub->get_name()))
         p_head = p_sub;
      if ((p_head->get_sub_object(s_sub_name) == nullptr) && (s_head_name ==
p_head->get_name()))
         p_sub = new cl_1(p_head, s_sub_name);
   }
}

int cl_application::exec_app(){
   cout << get_name();
   print_tree();
   return 0;
}
```

5.4 Файл cl_application.h

Листинг 4 – cl_application.h

```
#ifndef __CL_APPLICATION__H
#define __CL_APPLICATION__H

#include "cl_base.h"
#include "cl_1.h"

class cl_application : public cl_base{
public:
   cl_application(cl_base* p_head_object);
   void build_tree_objects();
   int exec_app();
};

#endif
```

5.5 Файл cl_base.cpp

Листинг 5 – cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base* p_head_object, string s_name){
   this->p_head_object = p_head_object;
   this->s_name = s_name;
   if (get_head() != nullptr)
      get_head()->p_sub_objects.push_back(this);
      // если объект не является корневым, то он становится дочерним для
своего родителя
}

cl_base::~cl_base(){
   for (int i = 0; i<p_sub_objects.size(); i++){
      delete p_sub_objects[i];
      // удаление i-того дочернего объекта
   }
}

string cl_base::get_name(){
   return s_name; // возврат имени объекта
}

cl_base* cl_base::get_head(){
   return p_head_object;
   // возвращение указателя на головной объект
}

cl_base* cl_base::get_sub_object(string s_name){
   for (int i = 0; i<p_sub_objects.size(); i++){
      if (p_sub_objects[i]->get_name() == s_name)
         return p_sub_objects[i];
         // возвращение указателя на дочерний объект с именем s_name
   }
   return nullptr;
}

void cl_base::print_tree(){
   if (!p_sub_objects.empty()){
      cout << endl << get_name();
      for (int i = 0; i<p_sub_objects.size(); i++){
         cout << "  " << p_sub_objects[i]->get_name();
         p_sub_objects[i]->print_tree();
         // вывод имен дочерних объектов от i-того объекта массива
      }
   }
}

bool cl_base::set_name(string s_new_name){
   if (get_head() != nullptr)
      for (int i = 0; i<get_head()->p_sub_objects.size(); i++)
```

```
            if(get_head()->p_sub_objects[i]->get_name() == s_new_name)
               return false;
      s_name = s_new_name;
      // присвоение дочернему объекту имени, если не имеется других дочерних
   объектов с аналогичным именем
      return true;
   }
```

5.6 Файл cl_base.h

Листинг 6 – cl_base.h

```
   #ifndef __CL_BASE__H
   #define __CL_BASE__H

   #include <iostream>
   #include <vector>
   #include <string>

   using namespace std;

   class cl_base{
      string s_name;
      cl_base* p_head_object;
      vector <cl_base*> p_sub_objects;
   public:
      cl_base(cl_base* p_head_object, string s_name = "Base object");
      ~cl_base();
      bool set_name(string s_new_name);
      string get_name();
      cl_base* get_head();
      void print_tree();
      cl_base* get_sub_object(string s_name);
   };

   #endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

```
   #include "cl_base.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 12.

Таблица 12 – 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_3	Object_root Object_root Object_1 Object_2 Object_3	Object_root Object_root Object_1 Object_2 Object_3
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_4 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_4 Object_5	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_4 Object_5
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_3 Object_5 Object_5 Object_6 Object_5 Object_7 Object_8 Object_8	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5 Object_5 Object_6 Object_7	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5 Object_5 Object_6 Object_7

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).