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.....10
3 ОПИСАНИЕ АЛГОРИТМОВ.....12
3.1 Алгоритм функции main.....12
3.2 Алгоритм конструктора класса base.....12
3.3 Алгоритм метода set_name класса base.....13
3.4 Алгоритм метода get_name класса base.....13
3.5 Алгоритм метода get_parent класса base.....14
3.6 Алгоритм метода print_branch класса base.....14
3.7 Алгоритм метода find_name класса base.....15
3.8 Алгоритм деструктора класса base.....16
3.9 Алгоритм конструктора класса application.....16
3.10 Алгоритм метода build_tree класса application.....16
3.11 Алгоритм метода exec_app класса application.....17
3.12 Алгоритм конструктора класса cl.....18
4 БЛОК-СХЕМЫ АЛГОРИТМОВ.....19
5 КОД ПРОГРАММЫ.....27
5.1 Файл application.cpp.....27
5.2 Файл application.h.....28
5.3 Файл base.cpp.....28
5.4 Файл base.h.....29
5.5 Файл cl.cpp.....30
5.6 Файл cl.h.....30
5.7 Файл main.cpp.....30

6 ТЕСТИРОВАНИЕ.....32
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....33

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:

- объект tree класса application предназначен для генерации и хранения дерева иерархии;
- cin - объект стандартного потока ввода с клавиатуры;
- cout - объект стандартного потока вывода на экран;
- if .. else - условный оператор;
- for - оператор цикла со счётчиком;
- while - оператор цикла с предусловием;
- new - оператор динамического выделения памяти;
- delete - оператор динамического очищения памяти;
- break - оператор прерывания цикла.

Класс base:

- свойства/поля:
    - поле имя объекта:
        - наименование — name;
        - тип — string;
        - модификатор доступа — private;
    - поле указатель на главный объект для текущего объекта:
        - наименование — parent;
        - тип — base*;
        - модификатор доступа — private;
    - поле список указателей на объекты класса base_class:
        - наименование — children;
        - тип — vector <base*>;
        - модификатор доступа — private;

- функционал:
    - метод base — параметризированный конструктор;
    - метод set_name — даёт имя объекту;
    - метод get_name — получает имя объекта;
    - метод get_parent — получает указатель на главный объект;
    - метод print_branch — вывод наименований в дереве иерархии;
    - метод find_name — получает указатель на объект с именем, указанном в аргументе;
    - метод ~base — деструктор.

Класс application:

- функционал:
    - метод application — параметризированный конструктор;
    - метод build_tree — построение дерева иерархии;
    - метод exec_app — запуск приложения.

Класс cl:

- функционал:
    - метод cl — параметризированный конструктор.

Таблица 1 – Иерархия наследования классов

№	**Имя класса**	**Классы-наследники**	**Модификатор доступа при наследовании**	**Описание**	**Номер**
1	base			базовый класс в иерархии	
2	application			класс корневого объекта	
		base	virtual public		1
3	cl			используется для построения дерева иерархии	
		base	virtual public		1

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: основная функция программы.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта tree класса application с параметром nullptr	2
2		вызов метода build_tree() объекта tree	3
3		возврат результата вызова метода exec_app() объекта tree	∅

3.2 Алгоритм конструктора класса base

Функционал: параметризированный конструктор.

Параметры: base* parent, string name = "base_name".

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса base

№	Предикат	Действия	№ перехода
1		запись в поле parent объекта значения параметра parent	2
2		запись в поле name значения параметра name	3

№	Предикат	Действия	№ перехода
3	parent != nullptr	вызов метода push_back поля children главного объекта для текущего с параметром указателем на текущий объект	∅
			∅

3.3 Алгоритм метода set_name класса base

Функционал: даёт имя объекту.

Параметры: string name.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода set_name класса base

№	Предикат	Действия	№ перехода
1	parent != nullptr		2
			5
2	проход по всем "детям" объекта		3
			5
3	child->get_name() == name		4
			5
4		возврат false	∅
5		запись в поле name значение параметра name	6
6		возврат true	∅

3.4 Алгоритм метода get_name класса base

Функционал: получает имя объекта.

Параметры: нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода get_name класса base

№	Предикат	Действия	№ перехода
1		возврат значения поля name объекта	∅

3.5 Алгоритм метода get_parent класса base

Функционал: получает указатель на главный объект.

Параметры: нет.

Возвращаемое значение: base*.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода get_parent класса base

№	Предикат	Действия	№ перехода
1		возврат указателя на главный объект текущего объекта	∅

3.6 Алгоритм метода print_branch класса base

Функционал: вывод наименований в дереве иерархии.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода print_branch класса base

№	Предикат	Действия	№ перехода
1	children.empty()	возврат	∅
			2
2		вывод на экран перехода на новую строку и имени	3

№	Предикат	Действия	№ перехода
		объекта	
3		инициализация переменной i типа int значением 0	4
4	i < children.size()	вывод на экран двух пробелов и имени подчинённого объекта	5
			6
5		i++	4
6		вызов метода print_branch() последнего "ребёнка" головного объекта	∅

3.7 Алгоритм метода find_name класса base

Функционал: получает указатель на объект с именем, указанном в аргументе.

Параметры: string name.

Возвращаемое значение: base*.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода find_name класса base

№	Предикат	Действия	№ перехода
1		инициализация переменной i типа int значением 0	2
2	i < this->children.size()		3
		возврат nullptr	∅
3	this->children[i]->get_name() == name	вовзрат children[i]	∅
			4
4		i++	2

3.8 Алгоритм деструктора класса base

Функционал: деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 9.

Таблица 9 – Алгоритм деструктора класса base

№	Предикат	Действия	№ перехода
1		инициализация переменной i типа int значением 0	2
2	i < children.size()	удаление элемента контейнера children[i]	3
			∅
3		i++	2

3.9 Алгоритм конструктора класса application

Функционал: параметризированный конструктор.

Параметры: base* root, string name = "base_name".

Алгоритм конструктора представлен в таблице 10.

Таблица 10 – Алгоритм конструктора класса application

№	Предикат	Действия	№ перехода
1		вызов конструктора класса base спараметрами root и name	∅

3.10 Алгоритм метода build_tree класса application

Функционал: построение дерева иерархии.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода build_tree класса application

№	Предикат	Действия	№ перехода
1		объявление переменных tree_parent и tree_child типа string	2
2		объявление вектора created с созданными объектами	3
3		ввод значения имени текущего объекта	4
4		задание ему этого имени с помощью метода set_name()	5
5		добавление в вектор created указателя на текущий объект	6
6		ввод имени головного и подчиненного объектов	7
7	tree_parent != tree_child	прохождение по всем созданным объектам из вектора	8
			13
8	(obj->get_name() == tree_parent) && !(obj->find_name(tree_child))	создание переменной child типа base*	9
			12
9		инициализация объекта child класса cl с передачей аргументов указателя на текущий объект из вектора и имени подчиненного объекта	10
10		добавление созданного объекта в начало вектора	11
11		обрывание цикла for	12
12		ввод имени головного и подчиненного объектов	7
13		вывод на экран имени головного объекта	∅

3.11 Алгоритм метода exec_app класса application

Функционал: запуск приложения.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода exec_app класса application

№	Предикат	Действия	№ перехода
1		вызов метода print_branch()	2
2		возврат 0	∅

3.12 Алгоритм конструктора класса cl

Функционал: параметризированный конструктор.

Параметры: base* parent, string name.

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса cl

№	Предикат	Действия	№ перехода
1		вызов конструктора базового класса с передачей в качестве аргументов указателя на головной объект и имени на текущий	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

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

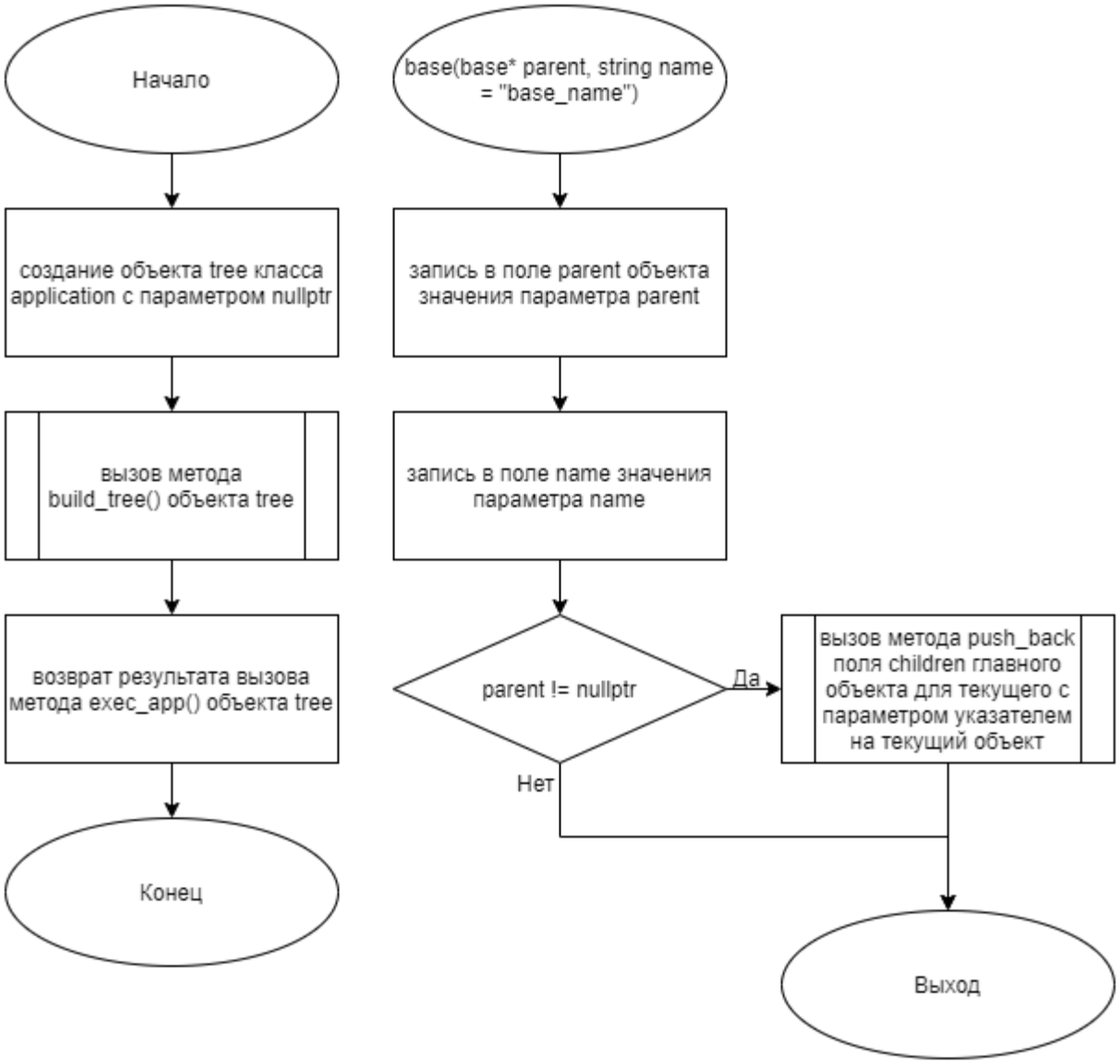

Рисунок 1 – Блок-схема алгоритма

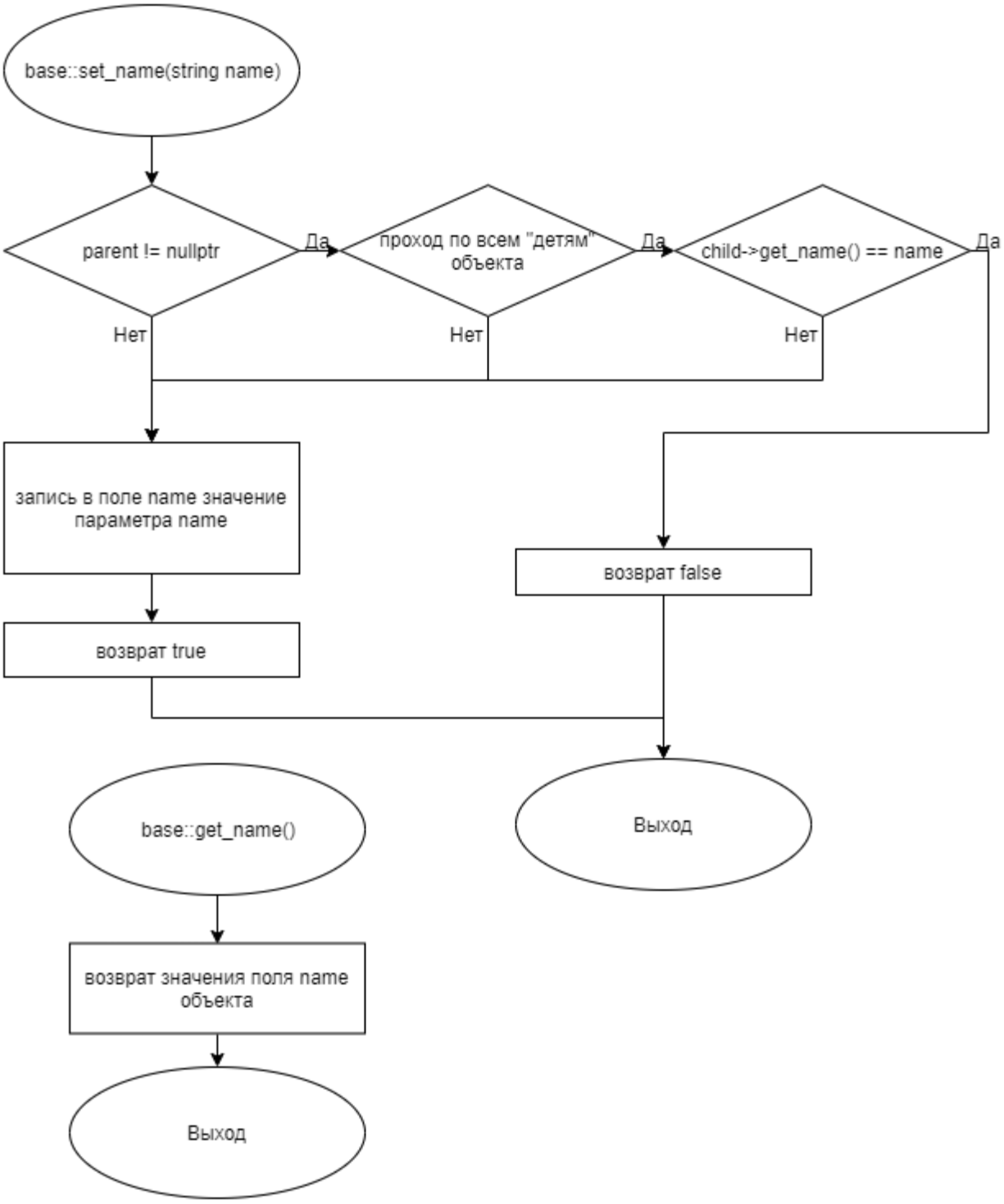

Рисунок 2 – Блок-схема алгоритма

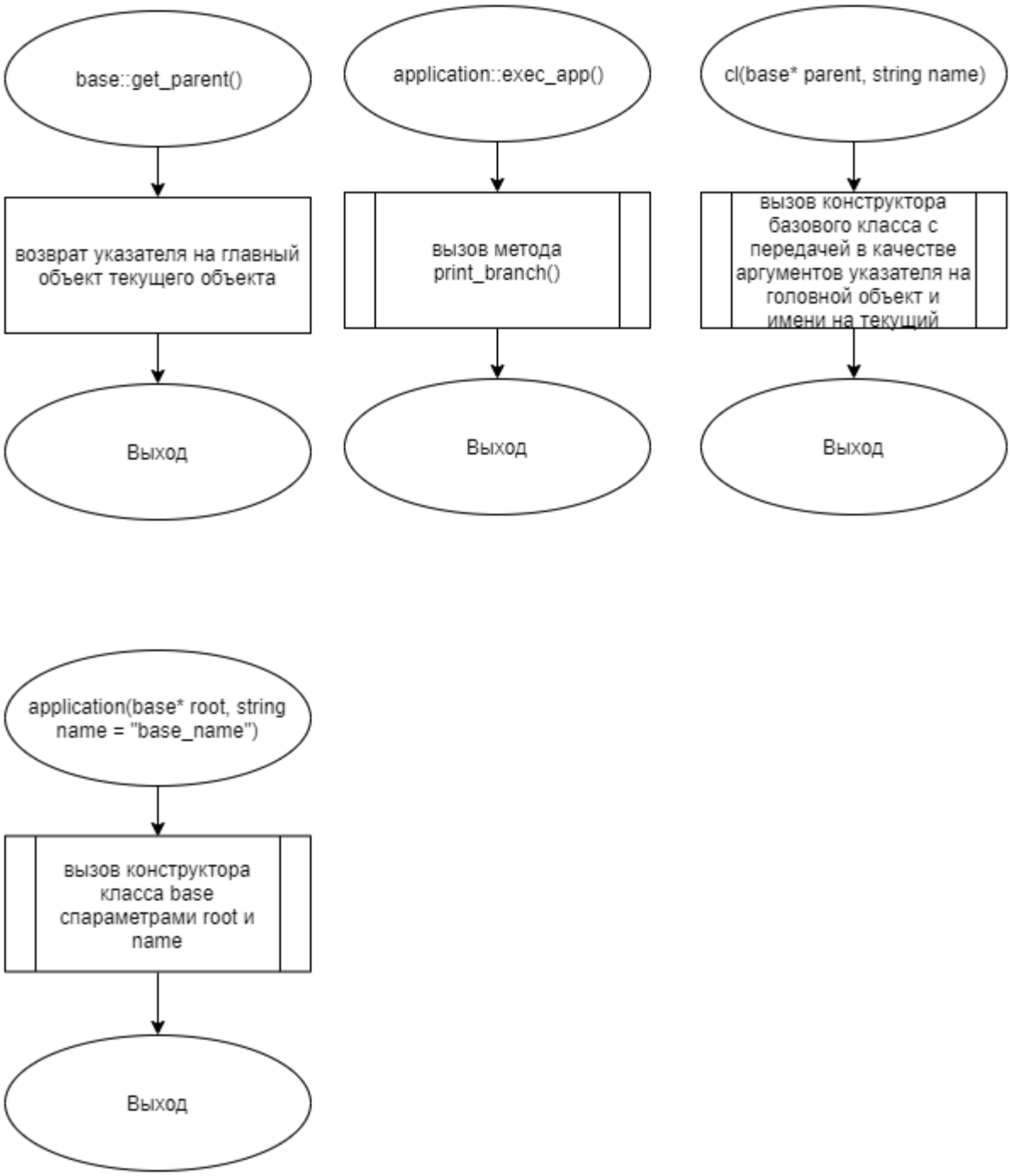

Рисунок 3 – Блок-схема алгоритма

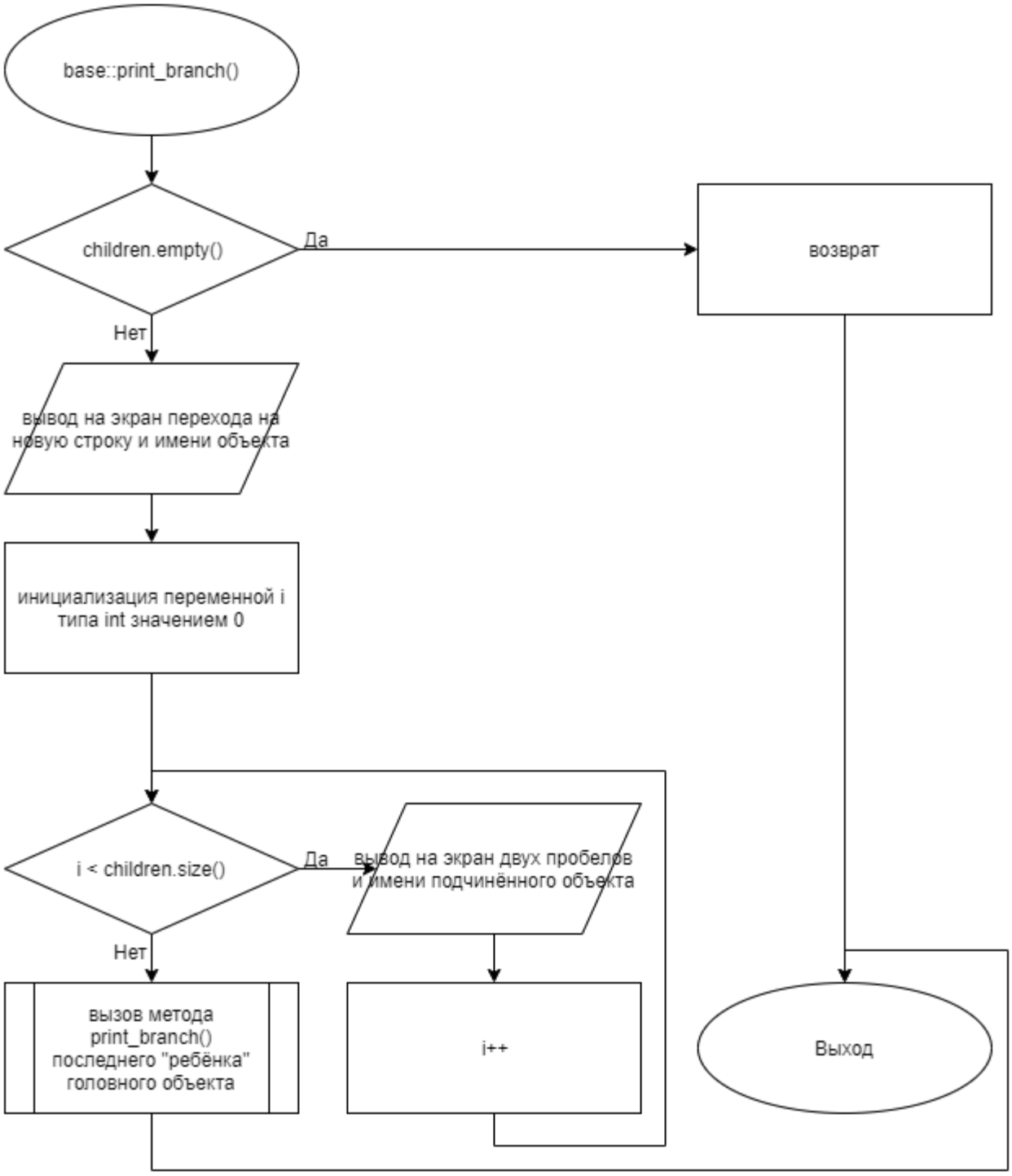

Рисунок 4 – Блок-схема алгоритма

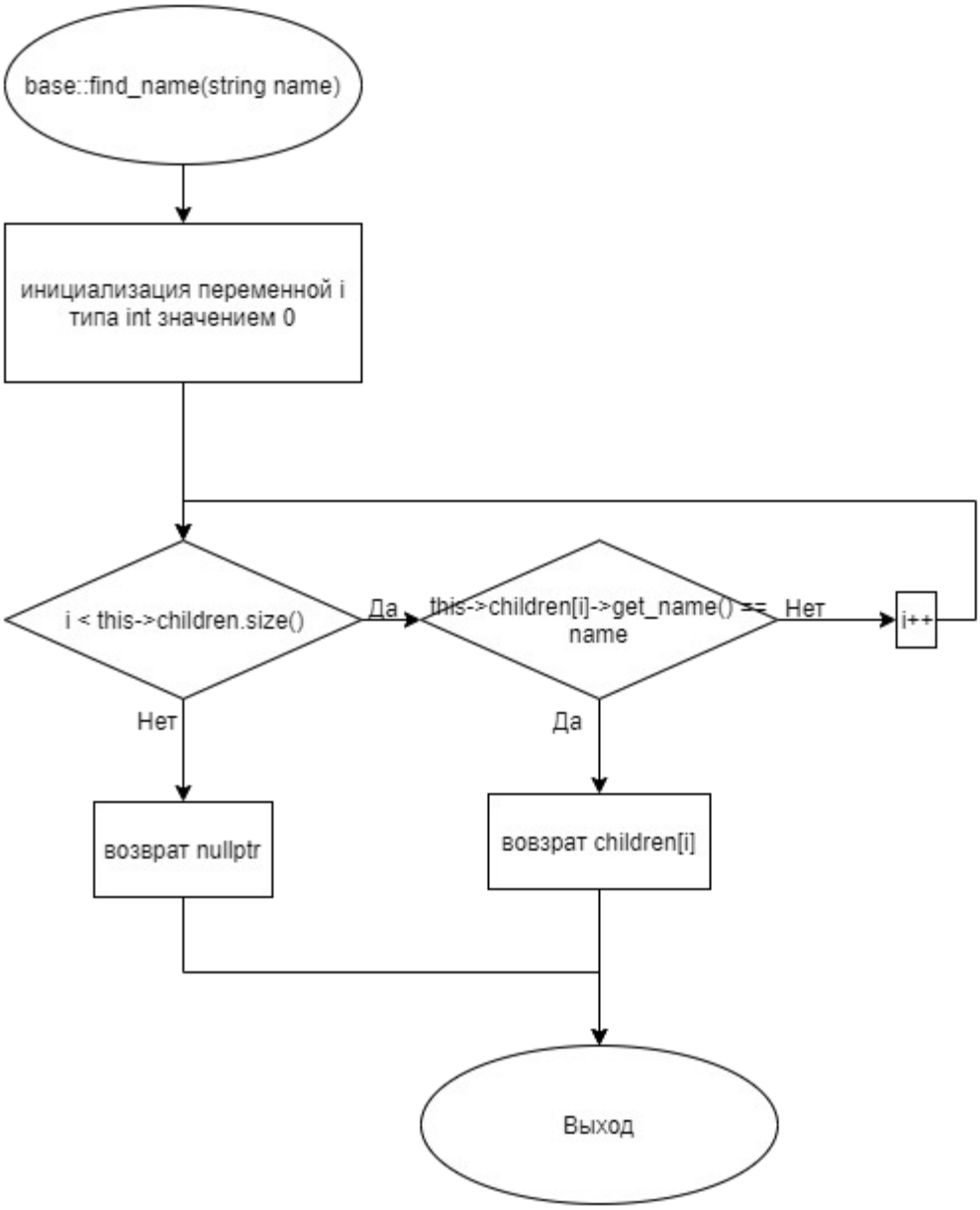

Рисунок 5 – Блок-схема алгоритма

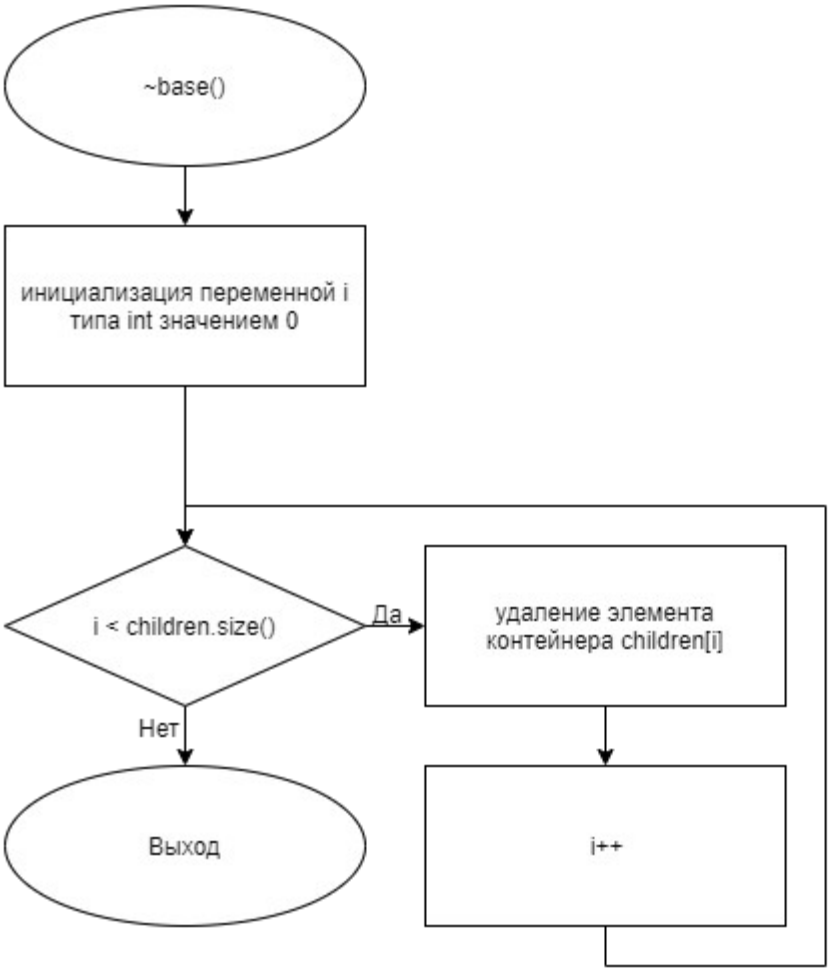

Рисунок 6 – Блок-схема алгоритма

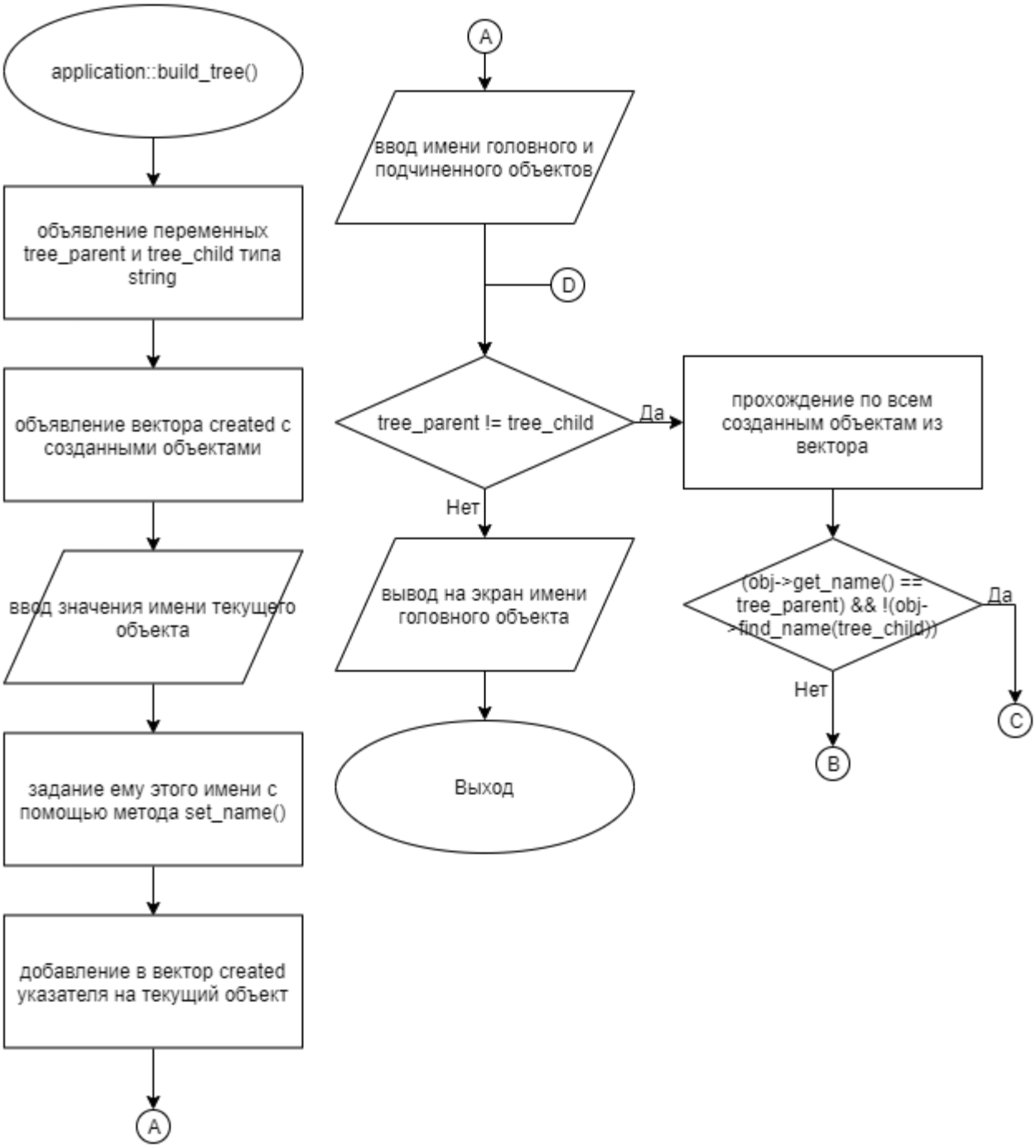

Рисунок 7 – Блок-схема алгоритма

Рисунок 8 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл application.cpp

Листинг 1 – application.cpp

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

application::application(base* root, std::string name): base(nullptr, name)
{}
void application::build_tree()
{
   std::string tree_parent, tree_child;
   std::vector <base*> created;
   std::cin >> tree_parent;
   set_name(tree_parent);
   created.push_back(this);
   std::cin >> tree_parent >> tree_child;
   while (tree_parent != tree_child) {
      for (base* obj: created) {
         if     ((obj->get_name()     ==     tree_parent)     &&     !(obj-
>find_name(tree_child))) {
            base* child = new cl(obj, tree_child);
            created.insert(created.begin(), child);
            break;
         }
      }
      std::cin >> tree_parent >> tree_child;
   }
   std::cout << get_name();
}
int application::exec_app()
{
   print_branch();
   return 0;
}
```

5.2 Файл application.h

Листинг 2 – application.h

```
#ifndef __APPLICATION__H
#define __APPLICATION__H
#include "base.h"
#include "cl.h"

class application: public base
{
public:
   application(base* root, std::string name = "base_name");
   void build_tree();
   int exec_app();
};

#endif
```

5.3 Файл base.cpp

Листинг 3 – base.cpp

```
#include "base.h"
#include <vector>
#include <iostream>

base::base(base* parent, std::string name)
{
   this->parent = parent;
   this->name = name;
   if (parent)
      parent->children.push_back(this);
}
bool base::set_name(std::string name)
{
   if (parent)
      for (base* child: parent->children)
         if (child->get_name() == name)
            return false;
   this->name = name;
   return true;
}
std::string base::get_name()
{
   return name;
}
base* base::get_parent()
```

```
{
   return parent;
}
void base::print_branch()
{
   if (children.empty())
      return;
   std::cout << std::endl << name;
   for (int i = 0; i < children.size(); i++)
      std::cout << "  " << children[i]->get_name();
   children.back()->print_branch();
}
base* base::find_name(std::string name)
{
   for (int i = 0; i < this->children.size(); i++)
      if (this->children[i]->get_name() == name)
         return children[i];
   return nullptr;
}
base::~base()
{
   for (int i = 0; i < children.size(); i++)
      delete children[i];
}
```

5.4 Файл base.h

Листинг 4 – base.h

```
#ifndef __BASE__H
#define __BASE__H
#include <iostream>
#include <string>
#include <vector>

class base
{
private:
   std::string name;
   base* parent;
   std::vector <base*> children;
public:
   base(base* parent, std::string name = "base_name");
   bool set_name(std::string name);
   std::string get_name();
   base* get_parent();
   void print_branch();
   base* find_name(std::string name);
   ~base();
};
```

```
#endif
```

5.5 Файл cl.cpp

Листинг 5 – cl.cpp

```
#include "cl.h"

cl::cl(base* parent, std::string name): base(parent, name) {}
```

5.6 Файл cl.h

Листинг 6 – cl.h

```
#ifndef __CL__H
#define __CL__H
#include "base.h"

class cl: public base {
public:
   cl(base* parent, std::string name);
};

#endif
```

5.7 Файл main.cpp

Листинг 7 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "base.h"
#include "application.h"

int main()
{
   application tree(nullptr);
   tree.build_tree();
   return tree.exec_app();
```

```
    }
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 14.

Таблица 14 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_3 Object_5 Object_6 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5
Petr Petr Alexandr Petr Elena Elena Daniil end end	Petr Petr Alexandr Elena Elena Daniil	Petr Petr Alexandr Elena Elena Daniil
Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_2 Object_4 Object_2 Object_5 Object_3 Object_6 end end	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_6	Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_6

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).