СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объектно-ориентированное программирование (ООП) - это не просто методология программирования, но и целый способ мышления программиста. Все мы живём в объектно-ориентированное мире, ведь всё вокруг нас представлено в виде экзампляров каких-то классов. Например, если мы видим какой-то цветок, то мы представляем его как некий объект класса "цветы". И даже не обязательно знать, как цветок называется, чтобы понять, что он цветок, ведь он наследует множество свойств от своего родительского класса.

ООП - это парадигма, то есть система идей и понятий, с помощью которых программисты реализуют программы, ориентированные на объекты, прямо как в реальном мире. В ООП реализовано множество практик, дающих возможность писать не только красивый код, но и код, который очень просто и удобно поддерживать, что и является основным преимуществом ООП. Помимо этого, концепция объектно-ориентированного программирования даёт ещё и значительный прирост в скорости разработки, ведь разработку программы можно разделять между различными независимыми программистами или группами программистов. Разделение работы достигается благодаря проектированию программного продукта таким образом, что блоки кода работают изолированно друг от друга. Все эти факторы дают нам понять, что ООП актуально и будет актуальным ещё долгое время.

Язык C++ является одним из самых удобных для изучения ООП, ведь он, можно сказать, первопроходец данной парадигмы программирования, и он перенял в себя множество отличных практик. Хотя C++ и преподносится как мультипарадигмальный язык, большинство программистов всё равно

воспринимают его сугубо как объектно-ориентированный язык. C++, также как и все остальные объектно-ориентированные языки, поддерживает три основных ООП концепции: наследование, инкапсуляция, полиморфизм, но, в отличие от остальных языков, C++ помогает отслеживать все этапы «жизни» нашей программы на более низком уровне . Такой низкоуровневый подход (указатели, выделение памяти оператором new и т.д.) в данном высокоуровневом языке может многих раздражать, но люди, которые смогли освоить ООП в C++, в других языках не будут завадать себе вопросов «почему это так?», «как это работает?». Освоение других языков после C++ обычно сводится лишь к освоению их синтаксиса.

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.

Создать базовый класс со следующими элементами:

Свойства:

- наименование объекта (строкового типа);
- указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно 0);
- массив указателей на объекты, подчиненные к текущему объекту в дереве иерархии.

Функционал:

- параметризированный конструктор с параметрами: указатель на головной объект в дереве иерархии и наименование объекта (имеет значение по умолчанию);
- метод определения имени объекта;
- метод получения имени объекта;
- метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз;
- метод переопределения головного объекта для текущего в дереве иерархии;

- метод получения указателя на головной объект текущего 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 программы-системы, изделия);
− 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. Каждый объект имеет уникальное имя.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Основная функция должна иметь следующий вид:

```
int main()
{
   cl_application  ob_cl_application ( nullptr );
   ob_cl_application.bild_tree_objects ( );  // 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.

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.

Примерввода

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

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

Метод решения

Для построения дерева объектов используем методы двух классов: Base и Application, а также объект класса ObjectTree для построения дерева. В классе Base реализованы основные методы для задания указателей на дочерние объекты и на родительский, а также методы для вывода в консоль дерева объектов. Класс Application строит дерево объектов и реализует запуск приложения. Иерархия наследования отображена в таблице 1.

Таблица 1. «Описание иерархии наследования классов»

№	Имена классов	Наследники	Модификатор доступа	Описание	Функционал
1	Base			Базовый класс для объектов дерева.	Основной алгоритм для взаимодейтсвия с объектами дерева.
		Application	public		Построение иерархии объектов
		ObjectTree	public		Класс, реализующий функционал дерева объектов
2	Application			Класс корневого объекта прилоежния	
3	ObjectTree			Класс дерева объектов	

Описание алгоритма

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

Функция: main

Функционал: Точка входа в программу, реализация вызова основных методов и инициализация объектов

Параметры: нет

Возвращаемое значение: Целое число - индикатор успешного завершения программы

Алгоритм функции представлен в таблице 2.

Таблица 2. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление и инициализация объекта application класса Application	2	
2		Вызов метода buildTree объекта application	3	
3		Вызов метода exec() объекта application	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: Base

Функционал: Конструктор по умолчанию

Параметры: Нет

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода Base класса Base

№	Предикат	Действия	№ перехода	Комментарий
1			∅	

Класс объекта: Base

Модификатор доступа: public

Метод: Base

Функционал: Перегруженный конструктор, задание родительского объекта и имени для текущего объекта

Параметры: Указатель на объект класса Base - родительский объект, строка - название текущего объекта

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода Base класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода setName, передача параметра name в качестве аргумента	2	
2		Вызов метода setHead, передача параметра head в качестве	3	

		аргумента		
3	head != nullptr	Вызов метода push_back у родительского объекта, поля children, и передача ему указателя на текущий объект	∅	
			∅	

Класс объекта: Base

Модификатор доступа: public

Метод: setName

Функционал: Сеттер для приватного поля m_name

Параметры: строка name - название объекта

Возвращаемое значение: нет

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода setName класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Присвоение полю m_name значения параметра name	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: setHead

Функционал: Сеттер для приватного поля m_head

Параметры: head - указатель на объект класса Base

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода setHead класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Присвоение полю m_head значения параметра head	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: getName

Функционал: Геттер для приватного поля m_name

Параметры: Нет

Возвращаемое значение: Строка - имя объекта

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода getName класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат значения приватного поля m_name	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: getHead

Функционал: Геттер для приватного поля m_head

Параметры: Нет

Возвращаемое значение: Указатель на объект класса Base - родительский для текущего

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода getHead класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат значения приватного поля m_head	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: findByName

Функционал: Поиск объекта по имени в дереве объектов

Параметры: строка - имя объекта, которого нужно искать в дереве

Возвращаемое значение: Указатель на объект класса Base

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода findByName класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	m_name = name	Возврат указателя на текущий объект	∅	Если название текущего объекта равно названию объекта, которого ищем
			2	
2	Текущий элемент range-based цикла на данной итерации в диапазоне элементов вектора		3	

	children?			
		Возврат указателя на нулевой адрес	∅	
3	Имя элемента равно параметру name	Возврат указателя на объект на данной интерации	∅	
			4	
4	Размер вектора дочерних объектов объекта на данной итерации больше 0?	вызов метода findByName у объекта на данной итерации	5	
			2	
5		Base* find = результат вызова findByName у объекта на данной итерации	6	
6	find != nullptr	Возврат find	∅	
			2	

Класс объекта: Base

Модификатор доступа: public

Метод: printTree

Функционал: Выводит в консоль дерево объектов

Параметры: Нет

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода printTree класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод "\n" + значение поля m_name + пробел	2	
2		Инициализация счётчика цикла size_t i = 0	3	
3	i < размера children		4	
			7	
4		Вывод результата метода getName у i-того объекта в векторе children	5	
5	i != размера children - 1	Вывод пробела	6	
			6	
6		i++	3	
7	Текущий элемент range-based цикла на данной итерации в диапазоне элементов вектора children?		8	
			∅	
8	Размер вектора children объекта на данной итерации > 0?	Вызов метода printTree() для объекта на данной итерации	8	
			∅	

Класс объекта: Base

Модификатор доступа: public

Метод: ~Base

Функционал: Деструктор по умолчанию

Параметры: Нет

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 11.

Таблица 11. Алгоритм метода ~Base класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	Текущий элемент range-based цикла на данной итерации в диапазоне элементов вектора children?	Вызов оператора delete для объекта на данной итерации	1	
			∅	

Класс объекта: Application

Модификатор доступа: public

Метод: Application

Функционал: Конструктор класса

Параметры: Нет

Возвращаемое значение: head - указатель на объект класса Base (родительский)

Алгоритм метода представлен в таблице 12.

Таблица 12. Алгоритм метода Application класса Application

№	Предикат	Действия	№ перехода	Комментарий
1	head != nullptr	Вызов метода push_back у объекта из параметра head, поля children, и передача ему параметра head	∅	

			∅	

Класс объекта: Application

Модификатор доступа: public

Метод: ~Application

Функционал: Деструктор по умолчанию

Параметры: Нет

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 13.

Таблица 13. Алгоритм метода ~Application класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов оператора delete для приватного поля m_tree	∅	

Класс объекта: Application

Модификатор доступа: public

Метод: buildTree

Функционал: Построение дерева объектов с помощью входных данных из потока ввода в консоли

Параметры: Нет

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 14.

Таблица 14. Алгоритм метода buildTree класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление строковых переменных objectNameParent, objectNameChild	2	
2		Инициализация objectNameParent с клавиатуры	3	
3		Инициализация m_root, создание объекта класса Base с аргументами: nullptr, objectNameParent	4	
4		Ввод objectNameParent и objectNameChild с клавиатуры	5	
5	objectNameParent != objectNameChild		6	
			∅	
6	Размер вектора children у приватного поля m_tree > 0	Объявление и инициализация переменной parent значением из результата вызова метода findByName поля m_tree. Аргумент: objectNameParent	7	
			8	
7	parent != nullptr	Инициализация объекта класса Base, аргументы конструктора: parent, objectNameChild	9	
			9	
8		Инициализация	9	

		объекта класса Base, аргументы конструктора: m_tree, objectNameChild		
9			4	

Класс объекта: Application

Модификатор доступа: public

Метод: exec

Функционал: Главный метод вывода построенного дерева в консоль

Параметры: Нет

Возвращаемое значение: Целое - индикатор успешного завершения

Алгоритм метода представлен в таблице 15.

Таблица 15. Алгоритм метода exec класса Application

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод результата метода getName у поля m_tree	2	
2		Вызов метода printTree у поля m_tree	3	
3		Возврат 1	∅	

Класс объекта: ObjectTree

Модификатор доступа: public

Метод: ObjectTree

Функционал: Конструктор с параметрами, вызов конструктора наследуемого объекта

Параметры: head - указатель на головной объект класса Base, string name - имя

объекта

Возвращаемое значение: Нет

Алгоритм метода представлен в таблице 16.

Таблица 16. Алгоритм метода ObjectTree класса ObjectTree

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов конструктора наследуемого объекта с аргументами head и name	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

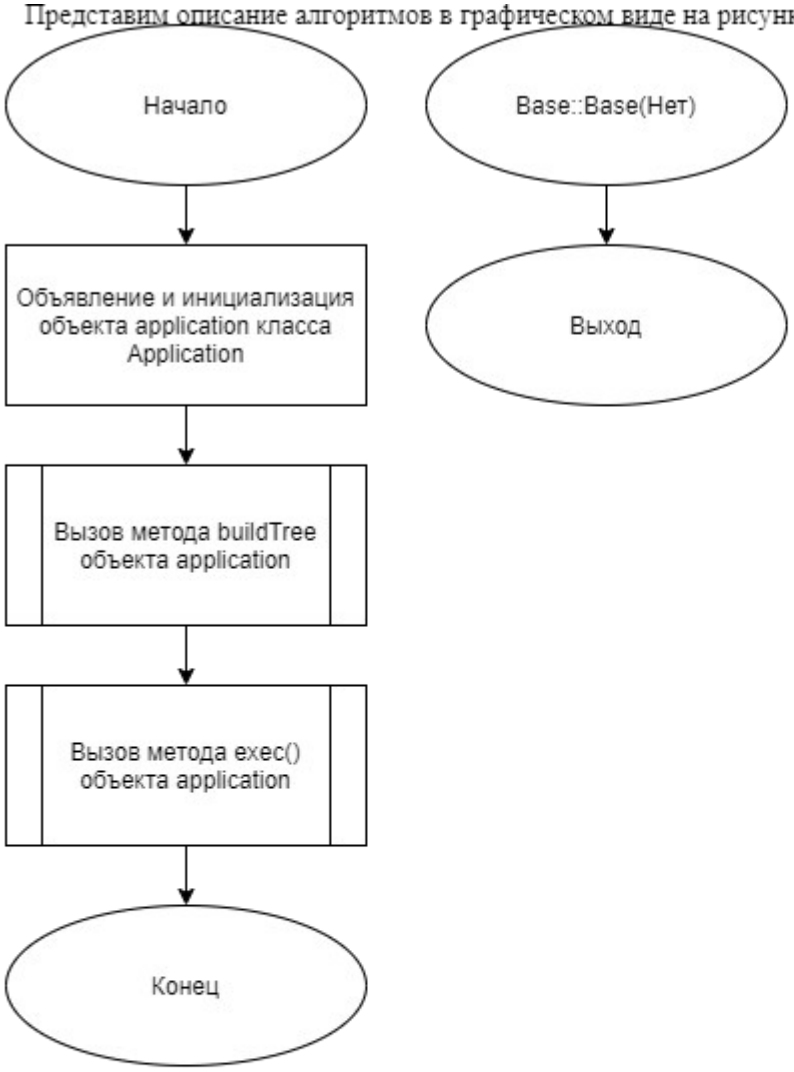

Рис. 1. Блок-схема алгоритма.

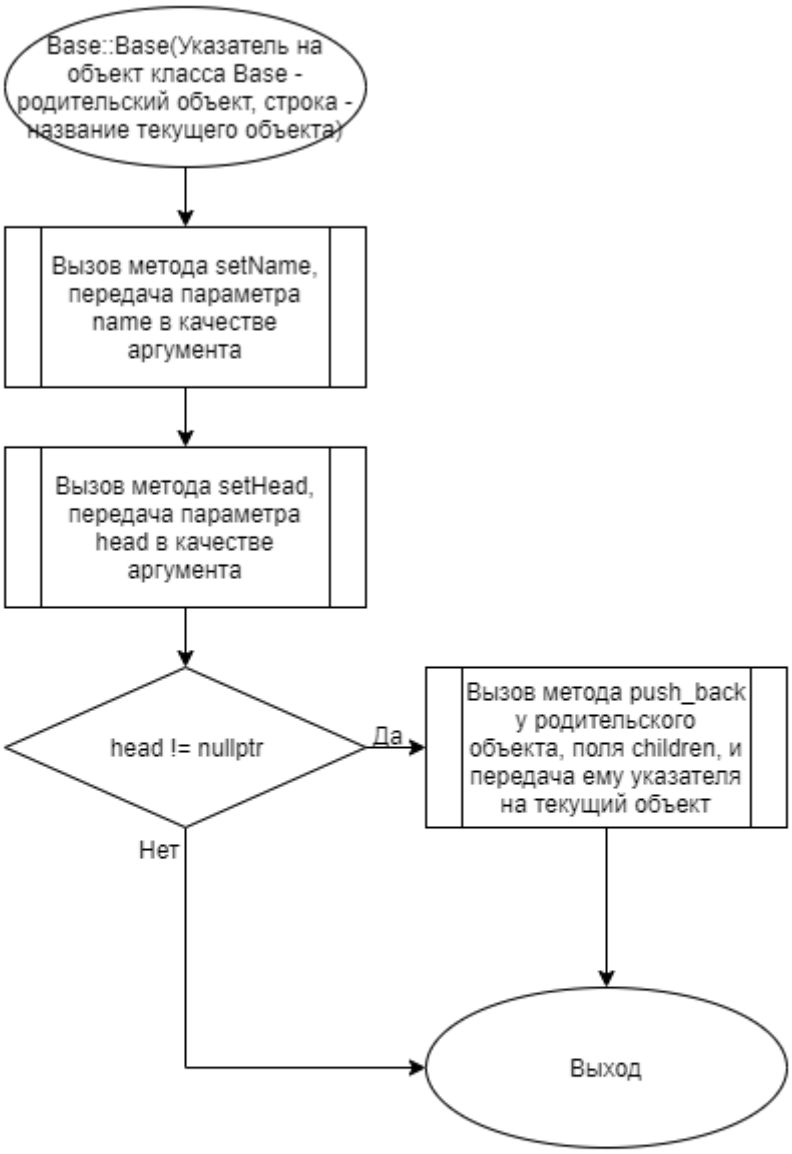

Рис. 2. Блок-схема алгоритма.

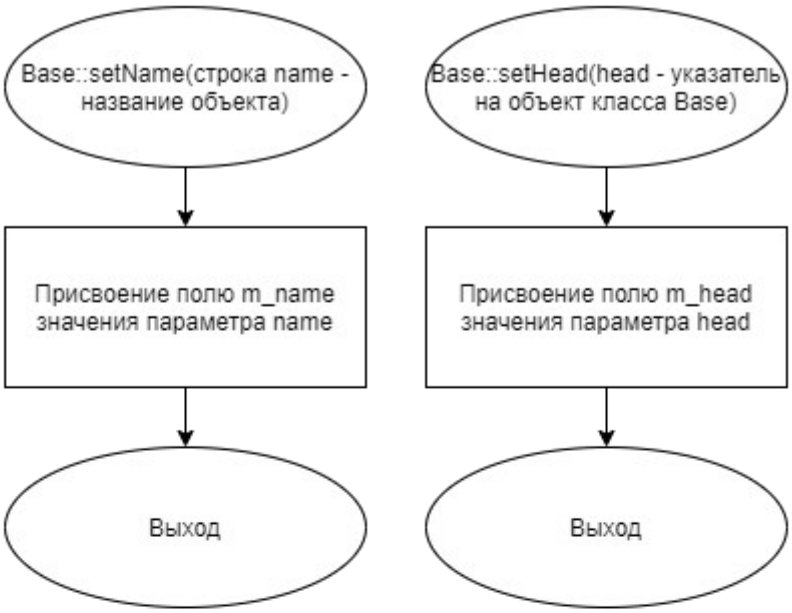

Рис. 3. Блок-схема алгоритма.

Рис. 4. Блок-схема алгоритма.

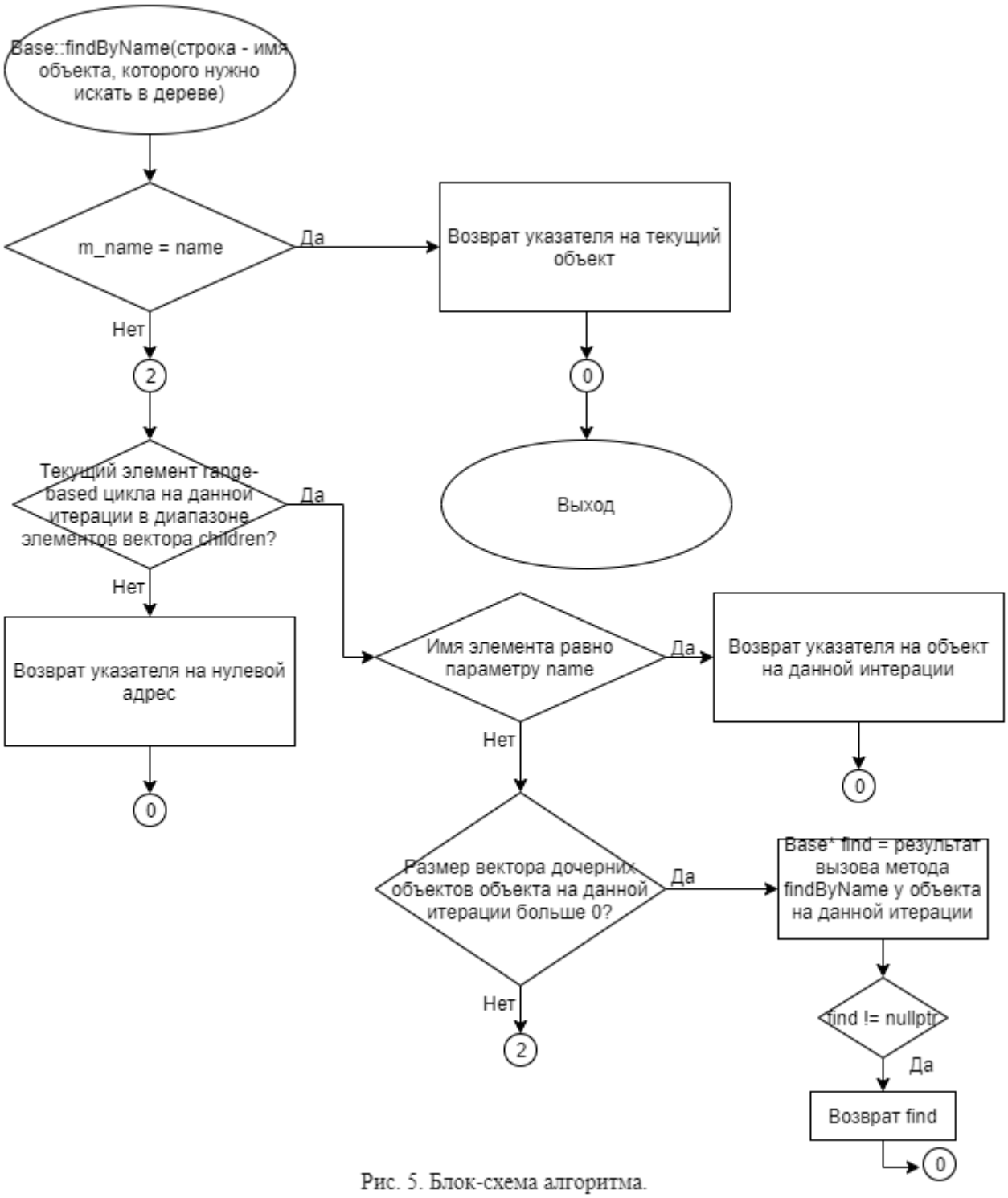

Рис. 5. Блок-схема алгоритма.

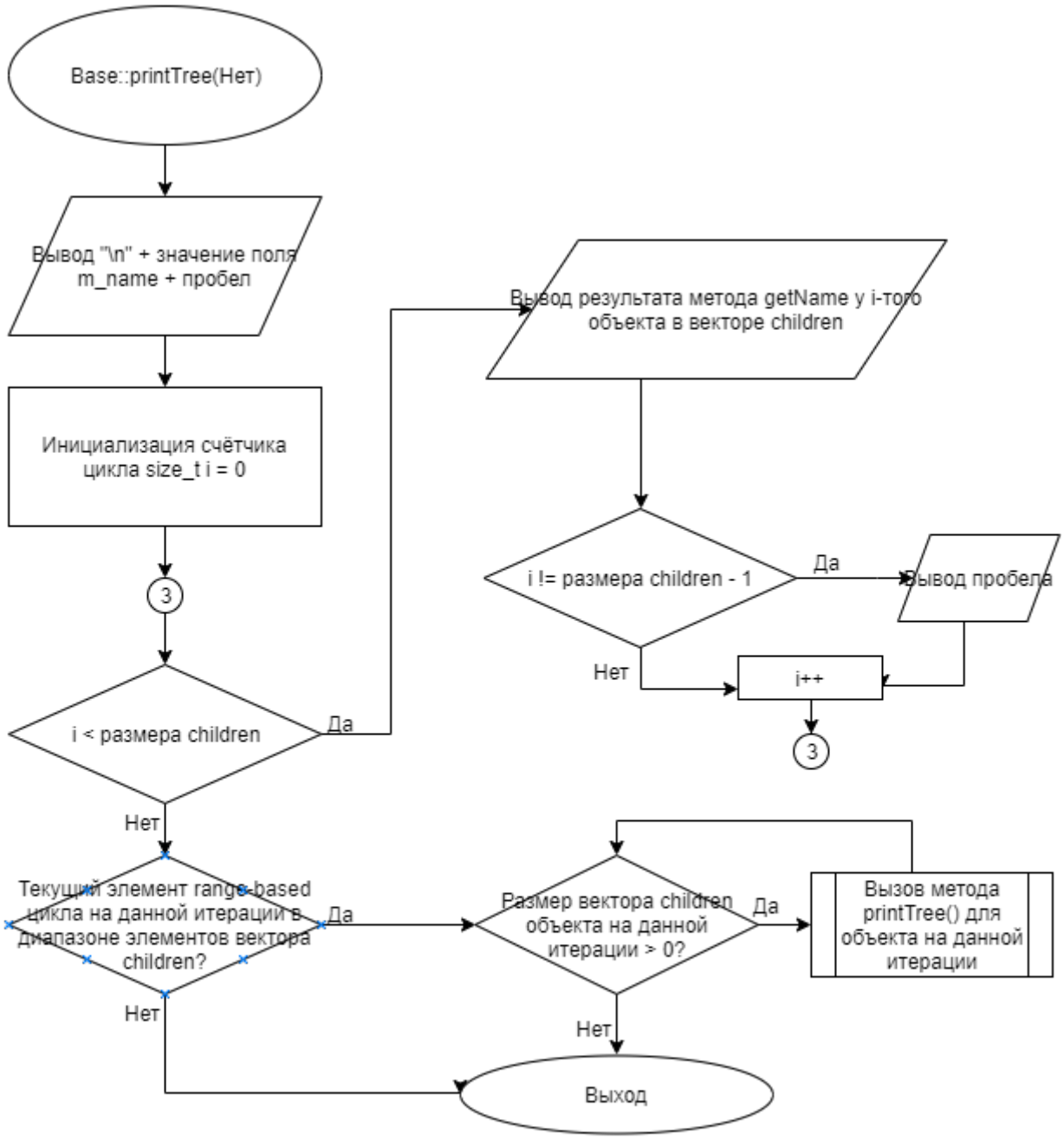

Рис. 6. Блок-схема алгоритма.

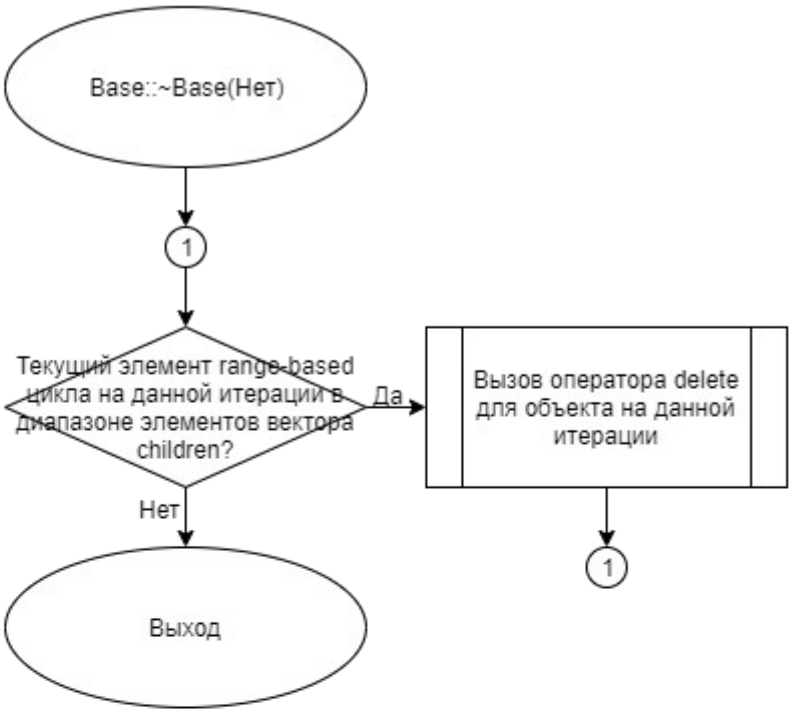

Рис. 7. Блок-схема алгоритма.

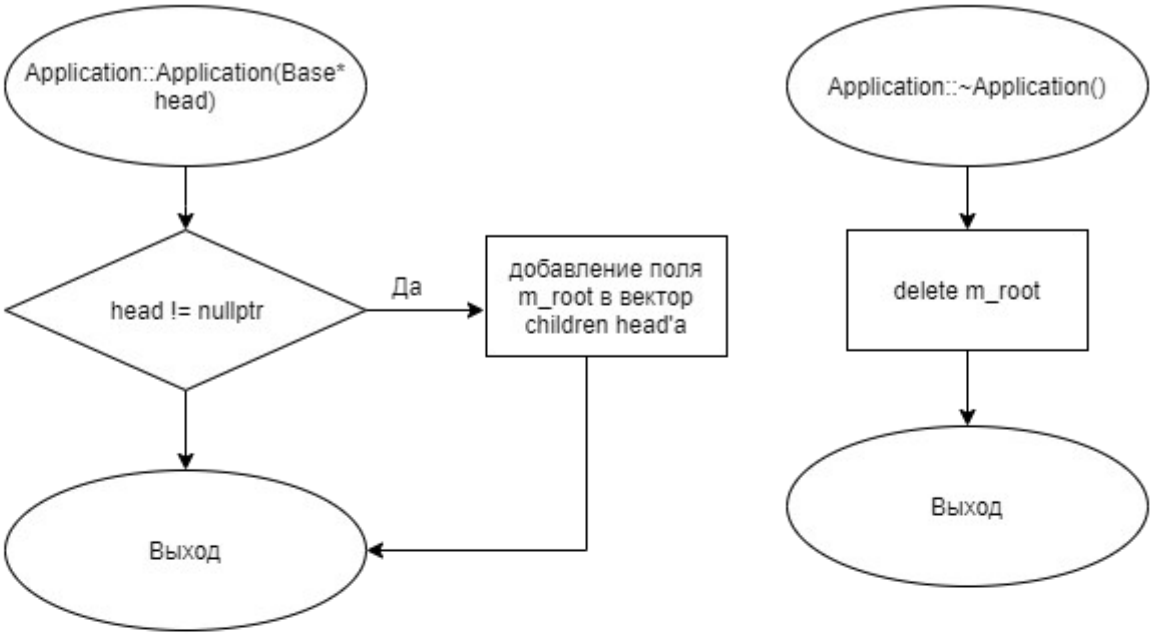

Рис. 8. Блок-схема алгоритма.

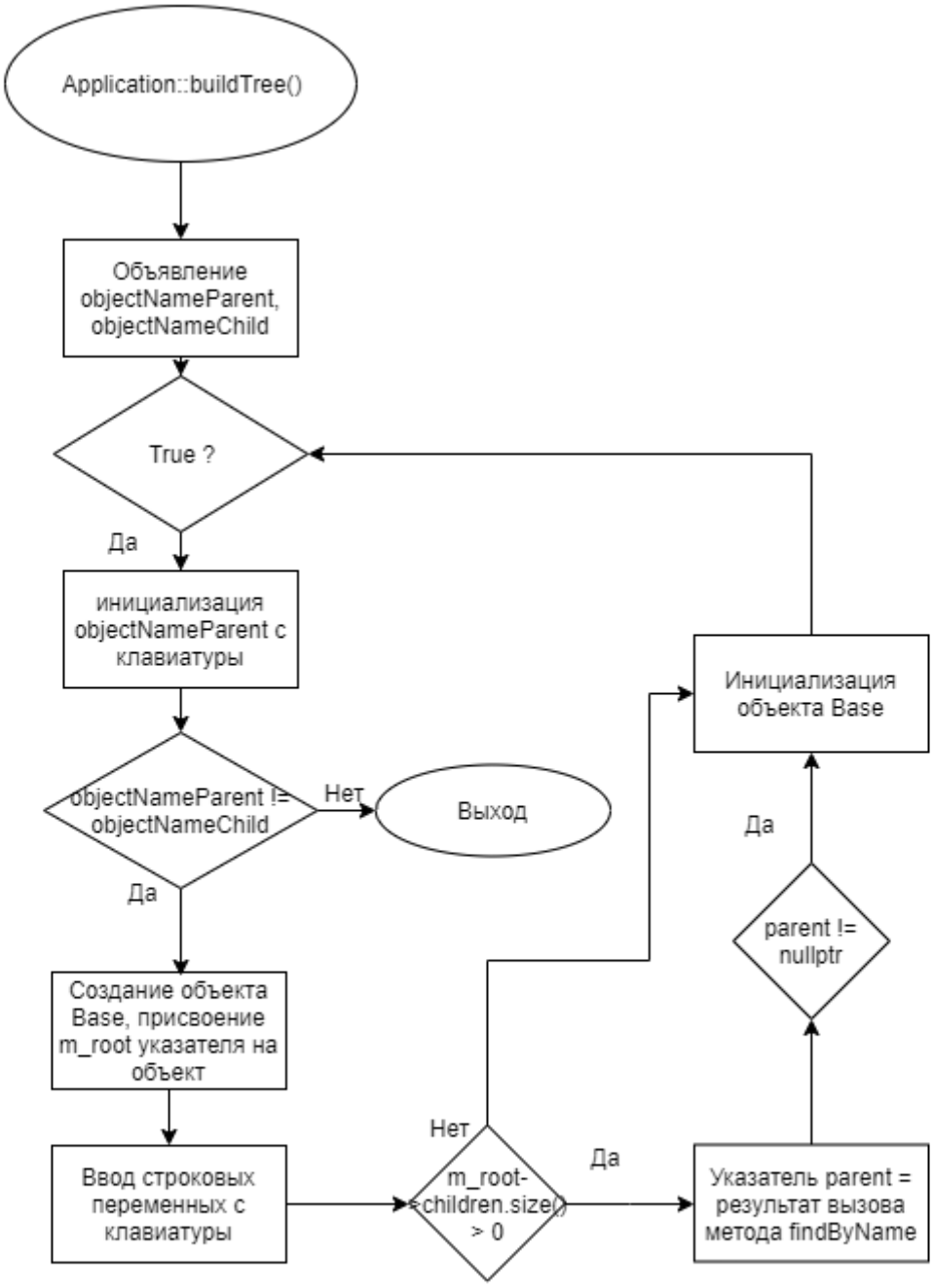

Рис. 9. Блок-схема алгоритма.

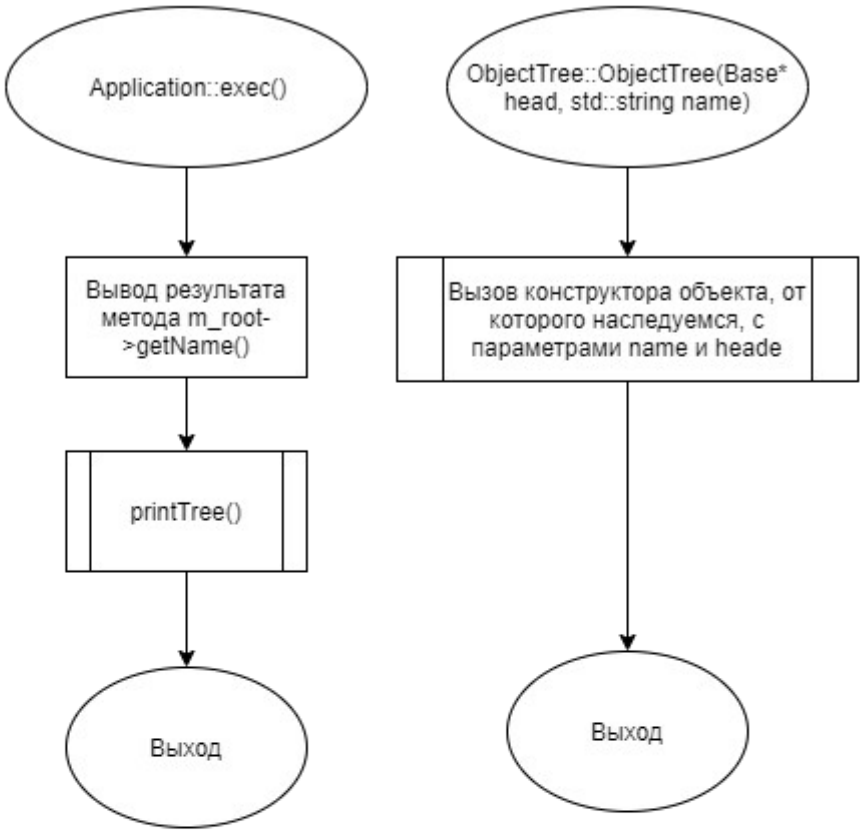

Рис. 10. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл Application.cpp

```
#include "Application.h"

using namespace std;

Application::Application(Base* head) : Base(head, "")
{
	if (head != nullptr) {
		head->children.push_back(m_tree);
	}
}

Application::~Application()
{
	delete m_tree;
}

void Application::buildTree()
{
	try {
		string objectNameParent = "";
		string objectNameChild = "";

		// инициализация имени корневого объекта
		cin >> objectNameParent;
		m_tree = new ObjectTree(nullptr, objectNameParent);
		while (true) {
			cin >> objectNameParent >> objectNameChild;
			// если у корневого объекта есть хотя бы один
			// дочерний, то проверяем, возможно текущий
			// объект станет дочерним для одного из дочерних
			// корневого
			if (objectNameParent != objectNameChild) {
				if (m_tree->children.size() > 0) {
					Base* parent = m_tree-
>findByName(objectNameParent);
					if (parent != nullptr)
						new Base(parent,
objectNameChild);
				}
				else {
					// добавляем нашему корневому объекту
новый
					// дочерний
					new Base(m_tree, objectNameChild);
				}
				// если последний объект не имеет дочерних,
				// то зависимости кончились. Выходим.
			}
```

```
                        else {
                                return;
                        }
                }
        }
        catch (std::bad_alloc ex) {
                return;
        }
}

int Application::exec()
{
        cout << m_tree->getName();
        m_tree->printTree();
        return 1;
}
```

Файл Application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H

#include "Base.h"
#include "ObjectTree.h"

class Application : public Base {
private:
        ObjectTree* m_tree = nullptr;

public:
        ~Application();
        Application(Base* head);

        /**
        * Построение дерева объектов. На вход в поток
        * ввода в консоль поступают имена объектов.
        *
        * 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
        *
        */
        void buildTree();

        /**
        * Главный метод. Вызывает метод вывода дерева
```

```
        * объектов в консоль.
        *
        * \return int 1 - код успешного завершения
        */
        int exec();
};

#endif
```

Файл Base.cpp

```
#include "Base.h"

using namespace std;

Base::~Base()
{
        for (const auto obj : children)
                delete obj;
}

Base::Base(Base* head, string name)
{
        setName(name);
        setHead(head);

        if (head != nullptr) {
                head->children.push_back(this);
        }
}

Base::Base(Base* head)
{
        setName("");
        setHead(head);

        if (head != nullptr) {
                head->children.push_back(this);
        }
}

void Base::setName(string name)
{
        m_name = name;
}

void Base::setHead(Base* head)
{
        m_head = head;
}

string Base::getName()
{
        return m_name;
}
```

```
void Base::printTree()
{
	cout << "\n" << m_name << "  ";
	for (size_t i = 0; i < children.size(); i++) {

		cout << children[i]->getName();
		if (i != children.size() - 1)
			cout << "  ";
	}

	// если у объекта есть хотя бы один дочерний,
	// то рекурсивно выводим дерево
	for (const auto obj : children) {
		if (obj->children.size() > 0) {
			obj->printTree();
		}
	}
}

Base* Base::findByName(string name)
{
	if (m_name == name)
		return this;

	// рекурсивный поиск по всем дочерним объектам
	for (const auto obj : children) {
		if (obj->getName() == name)
			return obj;
		else if (obj->children.size() > 0) {
			Base* find = obj->findByName(name);
			if (find != nullptr)
				return find;
		}
	}

	return nullptr;
}

Base* Base::getHead()
{
	return m_head;
}
```

Файл Base.h

```
#ifndef BASE_H
#define BASE_H

#include <iostream>
#include <vector>
```

```
class Base {
public:
        ~Base();
        Base(Base* head);
        Base(Base* head, std::string name = "");

        /**
        * Установить название текущему объекту.
        *
        * \param name
        */
        void setName(std::string name);

        /**
        * Получить название текущего объекта.
        *
        * \return std::string - название
        */
        std::string getName();

        /**
        * Вывод дерева объектов относительно текущего
        * объекта. Первая строка - это имя корневого
        * объекта. Вторая строка и последующие строки -
        * - имена головного и подчинённых объектов
        * очередного уровня, разделенных двумя пробелами.
        */
        void printTree();

        /**
        * Устанавливает головной объект текущему
        * объекту.
        *
        * \param head - указатель на объект класса Base
        */
        void setHead(Base* head);

        /**
        * Получить головной объект текущего объекта.
        *
        * \return - указатель на экземпляр класса Base
        */
        Base* getHead();

        /**
        * Поиск объекта в дереве объектов по его имени.
        * Поиск происходит в дереве относительно
        * текущего объекта, а не корневого.
        *
        * \param name - название объекта
        * \return - указатель на экземпляр класса Base
        */
        Base* findByName(std::string name);

        /**
        * Указатели на дочерние объекты
        */
        std::vector<Base*> children;

private:
```

```
        std::string m_name = "";
        Base* m_head = nullptr;
};

#endif
```

Файл main.cpp

```
#include "Application.h"

int main()
{
        Application application(nullptr);
        application.buildTree();
        return application.exec();
}
```

Файл ObjectTree.h

```
#ifndef OBJECTTREE_H
#define OBJECTTREE_H

#include "Base.h"

class ObjectTree : public Base {
public:
        ObjectTree(Base* head, std::string name) : Base(head, name) {};
};

#endif
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root root o1 o1 o2 o2 o3 o3 o4 o5 o6 o7 o7	root root o1 o1 o2 o2 o3 o3 o4	root root o1 o1 o2 o2 o3 o3 o4
root root o1 root o2 o3 o3	root root o1 o2	root root o1 o2
root root o1 o2 o2	root root o1	root root o1
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
1 1 2 1 3 1 4 2 2.1 2 2.2 3 3.1 3 3.2 4 4.1 4 4.2 4.2 4.2.1 5 5	1 1 2 3 4 2 2.1 2.2 3 3.1 3.2 4 4.1 4.2 4.2 4.2.1	1 1 2 3 4 2 2.1 2.2 3 3.1 3.2 4 4.1 4.2 4.2 4.2.1
1 1 2 2 3 4 4	1 1 2 2 3	1 1 2 2 3
root root 1 2 2	root root 1	root root 1

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы я получил множество практических и теоретических навыков, в том числе я научился:

- разрабатывать базовый класс для объектов;
- определять общий функционал для используемых в рамках приложения объектов;
- разрабатывать операции добавления, удаления, изменения позиции объекта в рамках иерархического дерева;
- освоил алгоритмы обработки структур данных в виде дерева;
- построению дерева иерархии объектов;
- переключению состояния объектов и определению их готовности к работе;
- выводить на печать иерархию объектов.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (ИСТОЧНИКОВ)

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения 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).

4. 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).

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).