МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

УЧЕБНОЕ ЗАДАНИЕ

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« Задание 4_1_1 »

С тудент группы	ИКБО-28-20	Коржов А.А.
Руководитель практики	Старший преподаватель	Перова Ю.П.
Работа представлена	«___»_____ 2021 г.	_____
		(подпись студента)
Оценка		_____
		(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной работе реализован алгоритм проверки готовности объектов к работе. В ходе разработки была использована литература, описанная в конце. Инструкции по построению схем, работой с типами данных, циклами и разветвлениями взяты из учебного пособия "Алгоритмы программы. Язык С++".

Поскольку в данной курсобой работе присутствует такая структура данных, как вектор или список, в коде были подключены уже существующие в языке С++ библиотеки, с подробной информацией о которых можно ознакомиться, прочитав книгу "С++ 17 STL. Стандартная библиотека шаблонов".

В тех случаях, когда требовалось создать свой список со своими параметрами и методами работы с данными, в данной работе были использованы ссылки и указатели.

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 решения задачи используются:

Функции ввода и вывода cin и cout.

Условный оператор if/else

Библиотеки string/vector

класс:base и его наследники cl_application, cl_1, cl_2

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	комментарий
1	cl_1			базовый класс Содержит основные поля и методы		
		cl_base	public		2	
2	cl_base			объявляет переменные и создаёт объекты		
		cl_application	public		3	
3	cl_application			класс приложения		

класс cl_base содержит поля:

1)string object_name - имя объекта

2)cl_base* paren - указатель на родительский объект

3)ststic cl_base* root - указатель на родительский объект

4)vector <cl_base*> children - вектор, хранящий в себе сам объект и его прямых наследников

а также методы:
1)cl_base(string object_name, cl_base* parent) - параметризированный конструктор
2)void set_name(string name) - метод сохранения имени
3)void set_parent(cl_bse* partent) - метод сохранения родительского объекта
4)string get_name() - метод получения имени
5)cl_base* get_object_by_name(string name) - метод получения ссылки на объект по его имени
6)void print_tree() - метод вывода в консоль дерева объектов

класс cl_applicatiom содержит методы:
1)void bild_tree_objects() - метод постройки дерева объектов(базового класса)
2)int exe_app() - метод ввода головного объекта и его дерева наследников (за счет метода базового класса print_tree)

Описание алгоритма

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

Функция: main

Функционал: основной алгоритм программы

Параметры: нет

Возвращаемое значение: int

Алгоритм функции представлен в таблице 2.

Таблица 2. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Создаётся объект класса cl_application	2	
2		вызов метода bil_tree	3	
3		вызывается метод exec_app	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: cl_application

Функционал: Конструктор

Параметры: cl_base* parent

Возвращаемое значение: -

Алгоритм метода представлен в таблице 3.

Таблица 3. Алгоритм метода cl_application класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		cl_base::root->set_parent(parent)	2	
2		cl_base::root->set_name("root")	3	
3		cl_base::root->children.push_back("root")	∅	

Класс объекта: cl_application

Модификатор доступа: public

Метод: bild_tree_objicts

Функционал: строит дерево объектов

Параметры: нет

Возвращаемое значение: -

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода bild_tree_objicts класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		ввод иени корневого элемента	2	
2		создание корневого объекта	3	
3		ввод имени родителя/ввод имени ребёнка	4	
4	name1==name2	вывод объектов дерева	∅	
			5	
5		получение ссылки на объект родителя	6	
6		создание дочернего объекта с передачей в него имени и	3	

		ссылки на родительский объект		

Класс объекта: cl_application

Модификатор доступа: public

Метод: exec_app

Функционал: вывод дерева объектов

Параметры: нет

Возвращаемое значение: int

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода exec_app класса cl_application

№	Предикат	Действия	№ перехода	Комментарий
1		вывод имени корневого объекта	2	
2		вызов метода базового класса print_tree	3	
3		выход из метода, возврат 0	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: cl_base

Функционал: создаёт объект и сохраняет имя и указатель на родителя

Параметры: srting object_name, cl_base* parent

Возвращаемое значение: -

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода cl_base класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняется имя объекта	2	
2	parent==nullptr	сохраняется родительский объект root, в родительский vector сохраняется текущий объект	3	
		сохраняется в родительский объект тот, что был передан параметром, в родительский vector сохраняется текущий объект	3	
3		в vector сохраняется текущий объект	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_name

Функционал: сохраняет имя

Параметры: string name

Возвращаемое значение: -

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода set_name класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет имя объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: set_parent

Функционал: сохраняет указатель на родительский объект

Параметры: cl_base* parent

Возвращаемое значение: -

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода set_parent класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		сохраняет указатель на родительский объект	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_name

Функционал: возврат имени объекта

Параметры: нет

Возвращаемое значение: string

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода get_name класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		возвращение имени объекта	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_object_by_name

Функционал: возврат казателя на объект

Параметры: string name

Возвращаемое значение: указатель на объект

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода get_object_by_name класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1	i<children.size()		2	
			3	
2	hildren[i]->get_name()==name	возврат children[i]	∅	
			1	
3	i<children.size()	вызов get_object_by_name(string name) от имени дочернего объекта	4	
		выход	∅	
4	children[i]->get_object_by_name(string name))->get_name()==name)	выход	∅	
			3	

Класс объекта: cl_base

Модификатор доступа: public

Метод: print_tree

Функционал: выводит дерево объектов

Параметры: нет

Возвращаемое значение: -

Алгоритм метода представлен в таблице 11.

Таблица 11. Алгоритм метода print_tree класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1	i<children[i]()	вывод children[i]	1	
			2	
2	i<children[i]()		3	
		выход	∅	
3	children[i]->children.size()>i	вызов print_tree()	2	
			3	

Класс объекта: cl_base

Модификатор доступа: public

Метод: ~cl_base()

Функционал: деструктор

Параметры: нет

Возвращаемое значение: -

Алгоритм метода представлен в таблице 12.

Таблица 12. Алгоритм метода ~cl_base() класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		очистка памяти	∅	

Класс объекта: cl_base

Модификатор доступа: public

Метод: get_parent

Функционал: возврат указателя на родительский объект

Параметры: нет

Возвращаемое значение: *parent(указатель на родительский объект)

Алгоритм метода представлен в таблице 13.

Таблица 13. Алгоритм метода get_parent класса cl_base

№	Предикат	Действия	№ перехода	Комментарий
1		возврат указателя на родительский объект	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

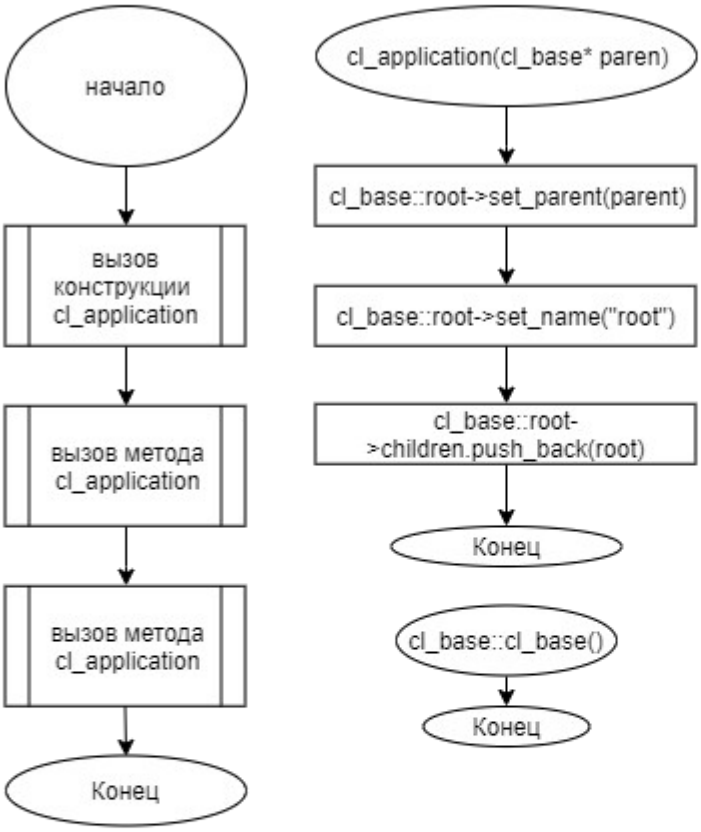

Рис. 1. Блок-схема алгоритма.

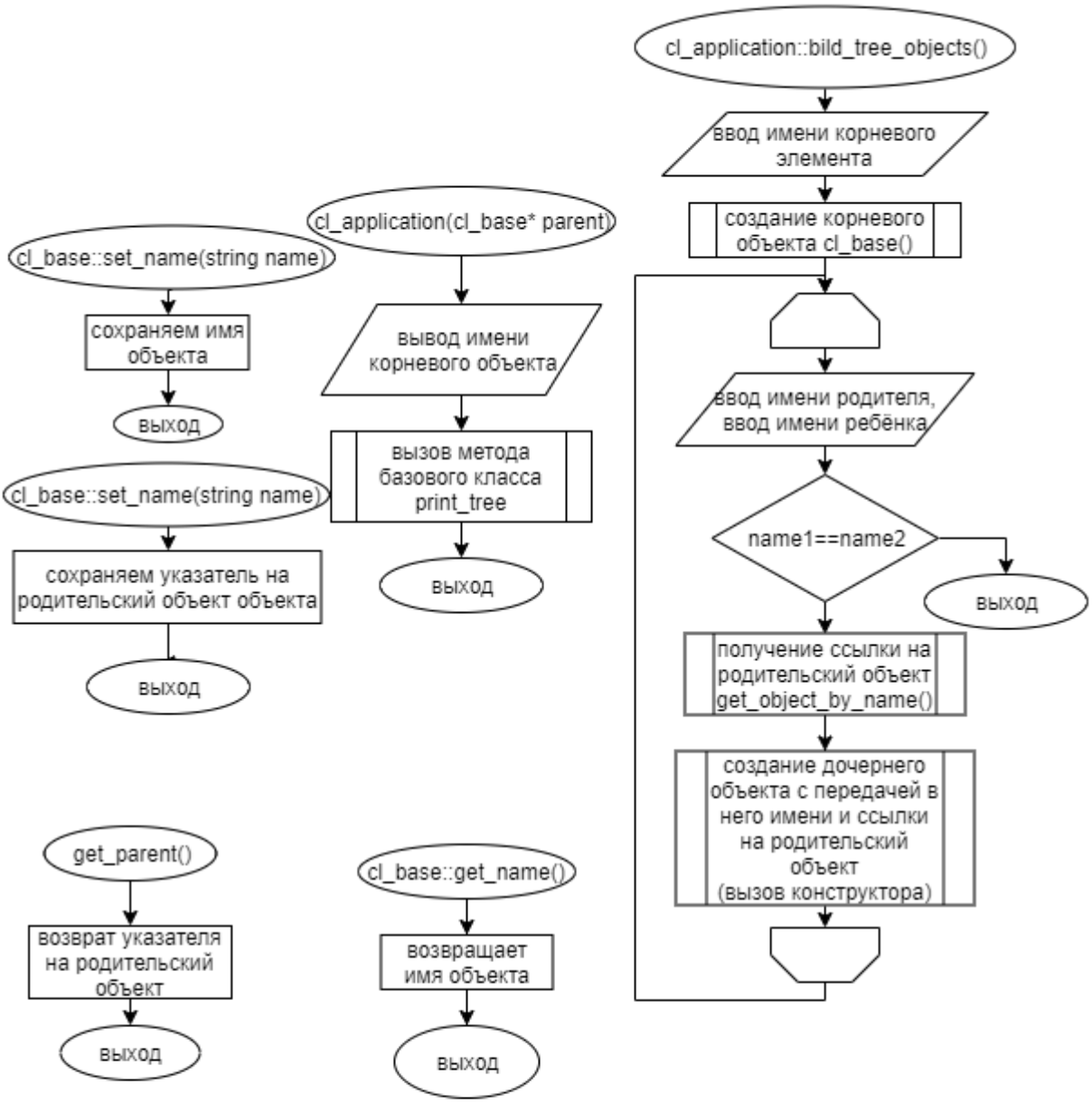

Рис. 2. Блок-схема алгоритма.

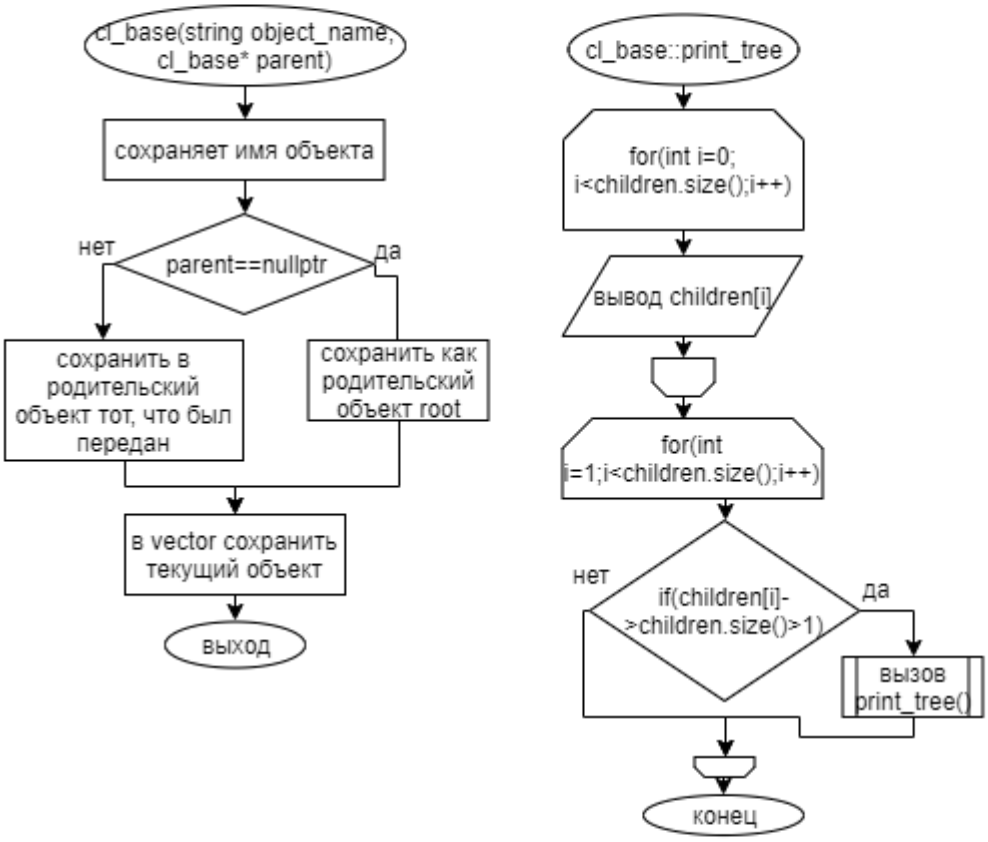

Рис. 3. Блок-схема алгоритма.

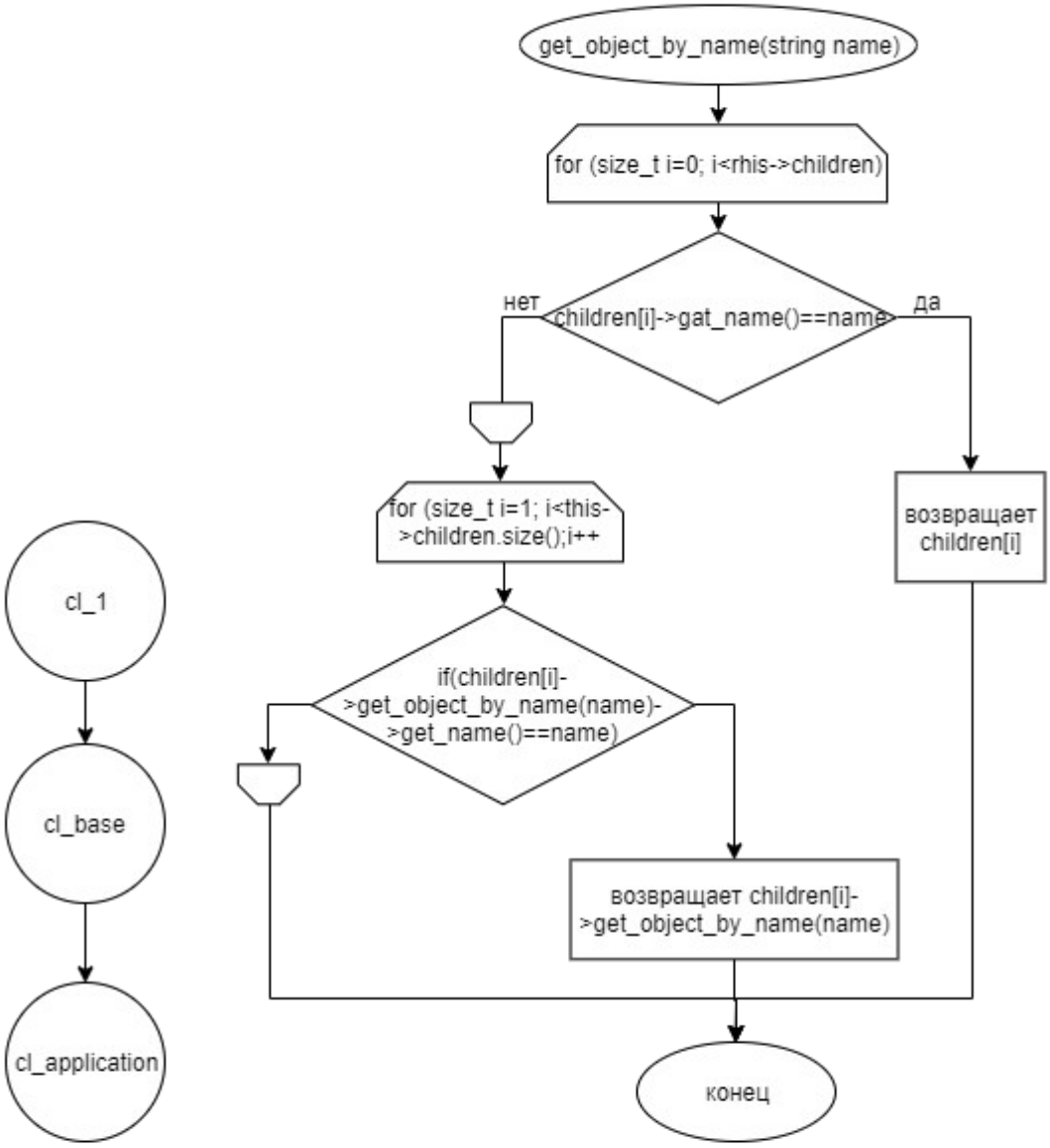

Рис. 4. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл cl_1.h

```
#ifndef CL_2_H
#define CL_2_H
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <vector>

class cl_1{
public:

};
#endif
```

Файл cl_application.cpp

```
#include "cl_application.h"

cl_application::cl_application(cl_base* parent = nullptr)
{
        cl_base::root->set_parent(parent);
        cl_base::root->set_name("root");
        cl_base::root->children.push_back(root);
}
void cl_application::bild_tree_objects() //функция постройки эллементов дерева
{
        string name1, name2;
        cin >> name1; //вводим корень дерева
        cl_base* child = new cl_base(name1, nullptr); //создание корневого
объекта
        while (true)
        {
                cin >> name1 >> name2; //вводим имя головного объекта и
подчинённого
                if (name1 == name2)//если имя головного и подчиненного
совпадают
                {
                        return; //заканчивается цикл ввода эллементов дерева
                }
                        cl_base* child2 = new cl_base(name2, root ->
get_object_by_name(name1));//создание дочернего объекта
                        child = child2;
        }
```

```
}
int cl_application::exec_app()
{
        cout << root->children[1]->get_name();//выводим корневой объект
        root->children[1]->print_tree(); //выводим иерархическое дерево
        return 0;
}
```

Файл cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H
#include "cl_base.h"
#include <string>

using namespace std;

class cl_application : public cl_base
{
public:
        cl_application(cl_base* parent); //указателб на родителя создаёт
объект
        void bild_tree_objects(); //построение/генерация дерева
        int exec_app(); //вывод корневого объекта и дерева в консоль
};
#endif
```

Файл cl_base.cpp

```
#include "cl_base.h"
#include <string>

cl_base * cl_base::root = new cl_base();
cl_base::cl_base()
{
        parent = nullptr; //обнуляет родителя
}
cl_base::cl_base(string object_name, cl_base* parent)//конструктор
{
        this->object_name = object_name;//присваивает имя объекта
        if (parent == nullptr)//Если объект не первый, то добавляется
        {
                set_parent(root);//Используем функцию для установки нового
родительского элемента
                (this->parent)->children.push_back(this);//добавляет текущий
объект в вектор
        }
```

```
        else
        {
                this->parent = parent; //сохраняем родителя
                parent->children.push_back(this); //добавляет текущий объект в
вектор
        }
        children.push_back(this); //добавляет текущий объект в вектор children
        index = (this->parent)->children.size() - 1;
}
void cl_base::set_name(string name)
{
        this->object_name = name; //присвоение имени объекту
}
void cl_base::set_parent(cl_base* parent)
{
        this->parent = parent; //сохраняет родителя
}
cl_base* cl_base::get_parent()
{
        return parent; //возвращает имя родителя
}
string cl_base::get_name()
{
        return object_name; //возврат имени объекта
}
cl_base* cl_base::get_object_by_name(string name) //находит объект по имени
{
        cl_base* val = nullptr;
        bool chek = false;
        for (size_t i = 0; i < this->children.size(); i++)
        {
                val = children[i]; //присваивает бъект вектора children
                if (children[i]->get_name() == name)
                {
                        chek = true;
                        return children[i]; //возвращает найденный по имени
объект
                }
        }
        for (size_t i = 1; i < children.size(); i++)
        {
                val = (children[i]->get_object_by_name(name));
                if ((children[i]->get_object_by_name(name))->get_name() ==
name)
                {
                return (children[i]->get_object_by_name(name));
                }
        }
        return val;
}
void cl_base::print_tree() //выводит иерархическое дерево
{
        for (int i = 0; i < children.size(); i++)
        {
                if (i == 0) cout << endl; //переход на новую строку
                cout << children[i]->get_name(); //выводит имя объекта в
векторе
                if (i + 1 < children.size()) cout << "  "; //вывод двойного
пробела
```

```
        }
        for (size_t i = 1; i < children.size(); i++)
        {
                if (children[i]->children.size() > 1)
                {
                        children[i]->print_tree();
                }
        }
}
cl_base::~cl_base() //деструктор
{}
```

Файл cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <vector>
#include "cl_1.h"

using namespace std;

class cl_base
{
        string object_name = ""; //имя объекта
        cl_base* parent; //указатель на родителя
        int index = 0; //индекс
public:
        int iterator;
        static cl_base* root; //головной объект
        vector <cl_base*> children; //ребёнок
        cl_base(); //конструктор класса cl_base
        cl_base(string object_name, cl_base* parent); //конструктор,
заполняющий переменные
        void set_name(string name); //назначение имени
        void set_parent(cl_base* parent); //сохраняет родителя
        cl_base* get_parent(); //возвращение родителя объекта
        string get_name();  //возвращение имени объекта
        cl_base* get_object_by_name(string name); //находит объект по имени
        void print_tree(); //строит дерево
        ~cl_base(); //применение деструктора
};
#endif
```

Файл main.cpp

```
#include "cl_application.h"

using namespace std;

int main()
{
setlocale(0,"");
cl_application ob_cl_application(nullptr);
ob_cl_application.bild_tree_objects();
return ob_cl_application.exec_app();
}
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
NAME NAME 1 NAME 2 1 q 1 w 1 e 1 r 1 t 1 y 2 u 2 i 2 o 2 p 3 3	NAME NAME 1 2 1 q w e r t y 2 u i o p	NAME NAME 1 2 1 q w e r t y 2 u i o p
1 1 0 1 2 2 5 2 6 2 7 9 9	1 1 0 2 2 5 6 7	1 1 0 2 2 5 6 7

ЗАКЛЮЧЕНИЕ

В данной курсовой работе реализована проверка готовности объектов к работе, при написании кода которой были использованы различные типы данных.

для осуществления задуманного алгоритма была изучена специальная литература, включающая в себя различные статьи и учебники по информационным технологиям.

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).