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-12-20 | Саакови Д. . |
| Руководитель практики | Ассистент | Рогонова О.Н. |
| Работа представлена | «___»_____ 2021 г. | _____ |
| | | *(подпись студента)* |
| Оценка | | _____ |
| | | *(подпись руководителя)* |

Москва 2021

# СОДЕРЖАНИЕ

# Введение

Была поставлена задача по выполнению курсовой работы по ООП, но перед этим стоит ответить на важные вопросы:

1)Почему ООП актуален и будет актуальным?

Для начала стоит сказать, что ООП это не просто программирование, это, в первую очередь проектирование программ, а уж после её написание.

Когда вы используете ООП первое, что вы должны сделать - научиться объектному мышлению, без него вы не сможете во всю силу использовать ООП.

В ООП есть 3 главных отличия от других видов программирования:

Наследование - его сила в том, что нам не придется переписывать строки кода и это экономит ресурсы системы и самое главное наше время.

Полиморфизм - используя его мы с легкостью поменяем поведения наследника, его полей и методов.

Инкапсуляция - основная особенность инкапсуляции заключается в том, что мы не должна знать, что происходит внутри нашей программы, т.е. мы можем использовать Геттеры и Сеттеры, они помогут нам быстрее и проще взаимодействовать с классами и их методами.

Исходя из всего вышеперечисленного можно сделать вывод, что ООП было и будет очень популярно среди программистов.

2) Почему C++ наиболее удобен для изучения ООП?

C++ - наиболее универсальный и подходящий под большинство современных задач язык программирования.

В нем очень удобно реализованны методы наследования, классы и другие возможности, на основе C++ был создан другой полностью объектно-ориентированный язык Java, но при этом C++ также оставляет возможность и для процедурного программирования , что в понимании новичка проще, чем ООП, тк новички сначала изучают процедурное программирование, а после переходят на ООП с уже имеющимеся знаниями языка.

3)Что дает ООП В C++ для моего направления?

Благодаря ООП мы можем позволяет создавать расширяемые системы, с помощью этого мы можем оптимизировать процессы, которые происходят в программе и, тем самым

уменьшить ресурсоёмкость программы, а так-же повышаем надежость , устойчивость и управляемость кода, а так-же возможности повторного использования некоторых фрагментов программы.

Пред нами стоит задача, по условию которой мы долны найти способ постоения иерархического дерева с помощью методов объкно-ориентированного программирования.

## 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

Потоки ввода/вывода cin, cout

Класс Object

Свойства:

1) string name - имя объекта

2) Object* parent - указатель на головной элемент

3) vector<Base*> children - список указателей на подчинённые объекты

4) vector<Base*>::iterator children_iterator - итератор списка указателей на объекты, подчиненные к текущему объекту

Методы:

1) Object - парамаетризированные конструктор класса по имени и указателю на головной элемент(name и parent)

2) set_name - метод, определяющий имя объекта

3) get_name - метод, возвращающий имя объекта

4) set_parent - метод, определяющий головной элемент

5) get_parent - метод, возвращающий головной элемент

6) print - метод, выводящий в консоль древо иерархии слева направо и сверху вниз

Класс Secondary

Свойства:-

Методы:

1) Secondary - параметризированный конструктор по указателю на головной эелемент

Класс Application

Свойства:

1) Object* curr_parent - указатель на текущий элемент

2) Object* curr_child - указатель на дочерний элемент

Метод:

1) Application - параметризированный конструктор класса по имени и указателю на головной элемент(name и parent)

2) build_tree - метод, строющий древо иерархии

3) exec - метод, запускающий работу программы

# Описание алгоритма

Функция: main

Функционал: основная программа

Параметры: нет

Возвращаемое значение: int

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Создание объекта obj класса Application<br>Вызов метода build_Tree объекта obj<br>Вызов метода exec объекта obj и его возврат | ∅ | |

Класс объекта: Object

Модификатор доступа: public

Метод: Object

Функционал: определяет головной объект и имя текущего объекта

Параметры: Object*, string

Возвращаемое значение: нет

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Присваиваем ссылку на родительский класс значение NULL<br>Присваиваем имени значение root<br>Вызываем метод set_parent с параметром parent<br>Вызываем метод set_name с параметром name | ∅ | |

Класс объекта: Object

Модификатор доступа: public

Метод: set_name

Функционал: Назначает имя текущего объекта

Параметры: string

Возвращаемое значение: void

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Присваивает свойству name значение параметра name | ∅ | |

Класс объекта: Object

Модификатор доступа: public

Метод: get_name

Функционал: Возвращаем значение свойства name

Параметры: нет

Возвращаемое значение: string

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Возвращаем значение свойства name | ∅ | |

Класс объекта: Object

Модификатор доступа: public

Метод: set_parent

Функционал: Определяет головной объект к текущему объекту

Параметры: Object*

Возвращаемое значение: void

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Удаляем элемент из родительского списка<br>Присваиваем свойству parent указатель на текущий объект | 2 | |
| 2 | Головной объект не нулевой | Вызываем метод push_back для свойства children головного объекта для текущего с указателем на текущий объект | ∅ | |

Класс объекта: Object

Модификатор доступа: public

Метод: get_parent

Функционал: Возвращаем указатель на головной объект

Параметры: нет

Возвращаемое значение: Object*

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Возвращаем указатель на головной объект | ∅ | |

Класс объекта: Object

Модификатор доступа: public

Метод: print

Функционал: Выводим дерево иерархии слева направо и с верху вниз

Параметры: нет

Возвращаемое значение: void

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | У текущего объекта нет дочерних объектов | | ∅ | |
| | | Выводим значение свойства name текущего объекта с новой строки | 2 | |
| 2 | | Записываем в свойство итератора текущего объекта итератора первого подчиненного объекта | 3 | |
| 3 | Выведены не все дочерние объекты | Выводим имена дочерних объектов через двойной пробел Инкрементируем итератор | 3 | |
| | | Декрементируем итератор | 4 | |
| 4 | | Вызываем метод print для текущего дочернего объекта | ∅ | |

Класс объекта: Secondary

Модификатор доступа: public

Метод: Secondary

Функционал: определяет головной объект и имя текущего объекта

Параметры: Object*, string

Возвращаемое значение: нет

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 1 | | Вызывает конструктор класса Object по parent и name | ∅ | |

Класс объекта: Application

Модификатор доступа: public

Метод: exec

Функционал: запускает приложение

Параметры: нет

Возвращаемое значение: int

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Выводим имя текущего объекта<br>Вызываем метод print для текущего объекта | 2 | |
| 2 | | Возвращаем 0 | ∅ | |

Класс объекта: Application

Модификатор доступа: public

Метод: Application

Функционал: Определяет головной объект для текущего элемента

Параметры: Object*

Возвращаемое значение: нет

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Вызываем конструктор Object по parent | ∅ | |

Класс объекта: Application

Модификатор доступа: public

Метод: build_tree

Функционал: Строит дерево иерархии объектов

Параметры: нет

Возвращаемое значение: void

| № | Предикат | Действия | № перехода | Комментарий |
|---|---|---|---|---|
| 1 | | Объявляем строковые переменные parent_name и children_name<br>Вводим parent_name | 2 | |
| 2 | | Вызываем set_name по parent_name | 3 | |
| 3 | | Записываем в curr_parent указатель на текущий объект<br>Вводим имя головного и имя дочернего объектов | 4 | |
| 4 | Имя головного и дочернего объекта совпадают | | ∅ | |
| | | | 5 | |
| 5 | Новое имя головного элемента не совпадает со старым именем головного | | 6 | |
| | | | 7 | |
| 6 | Новое имя головного объекта совпадает с старым именем дочернего | Записываем в свойство curr_parent корнегово объекта curr_child | 7 | |
| | | | 3 | |
| 7 | | Создаём объект класса Secondary с помощью конструктора, с параметрами curr_parent корневого объекта и child_name<br>Указатель на объект сохраняем в свойство curr_child корневого объекта | ∅ | |

# Блок-схема алгоритма

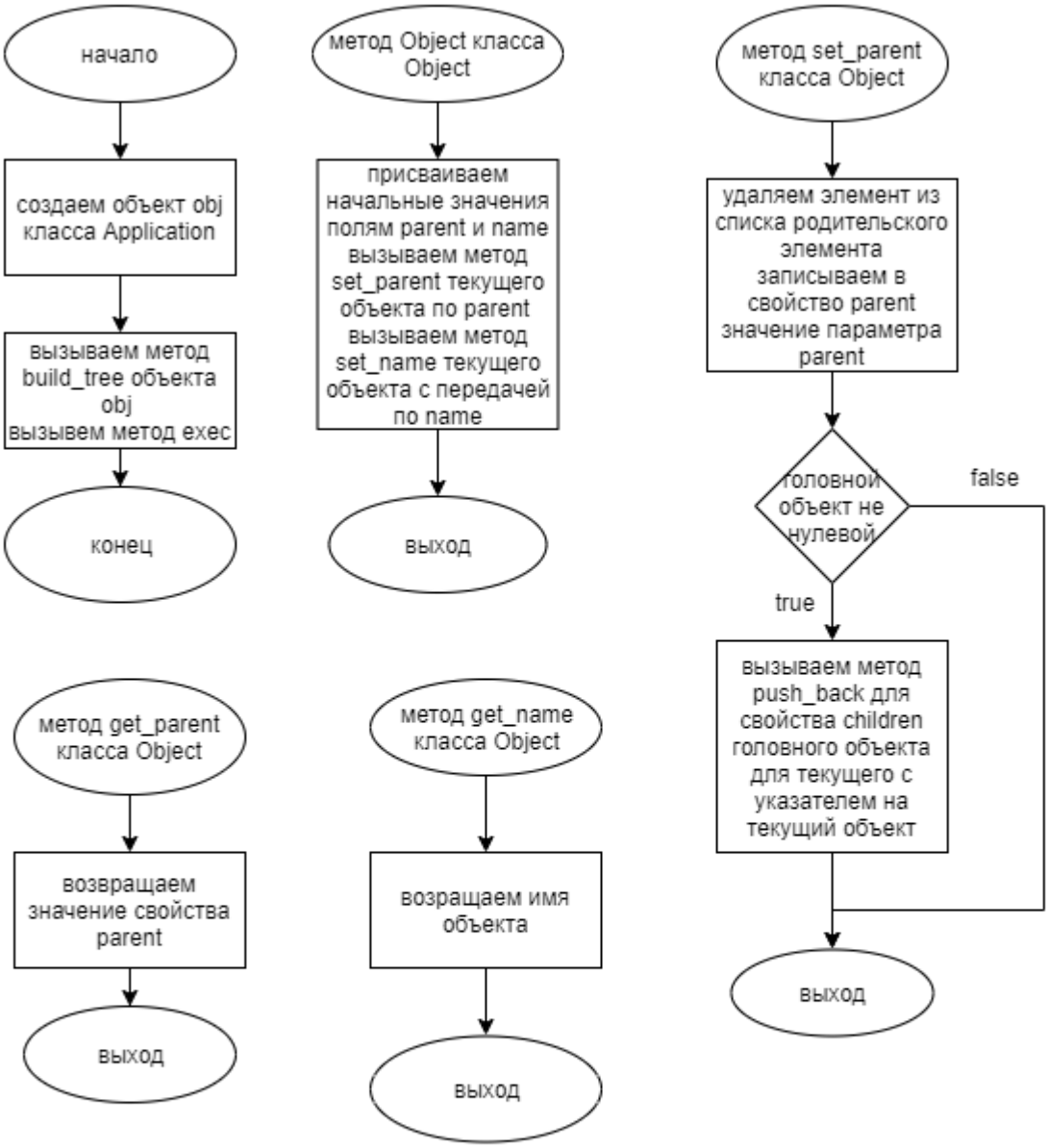


Рис. 1. Блок-схема алгоритма.

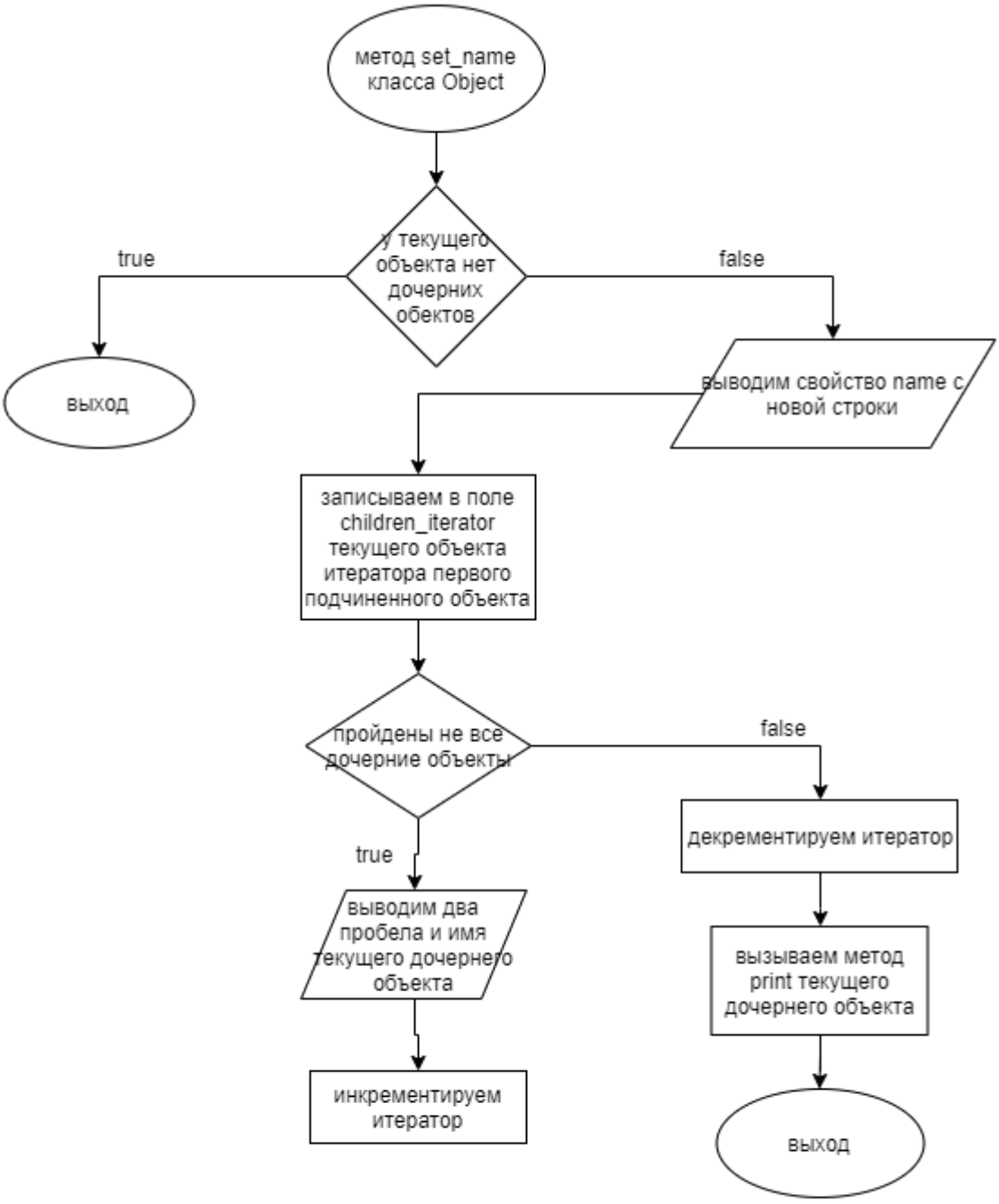


Рис. 2. Блок-схема алгоритма.

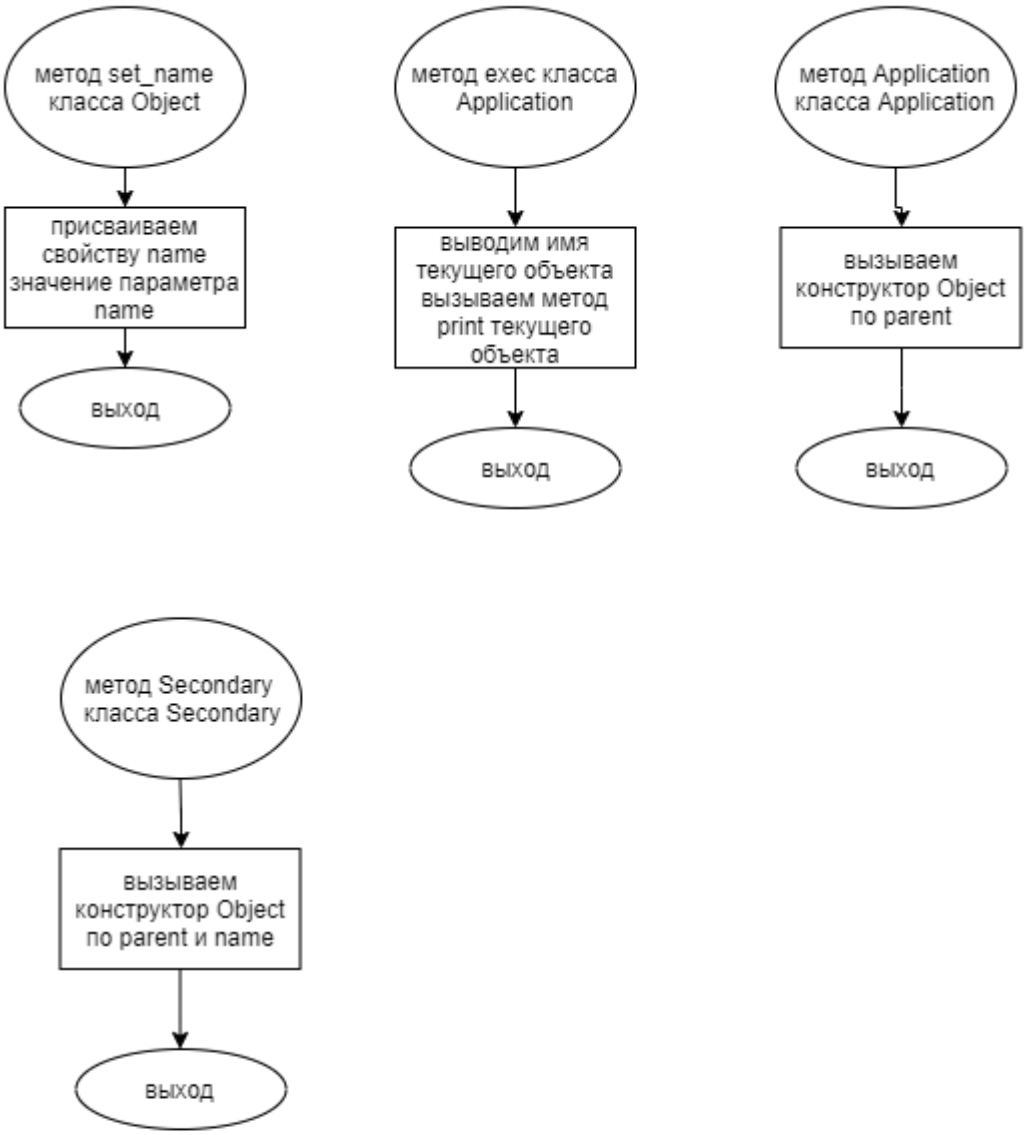


Рис. 3. Блок-схема алгоритма.

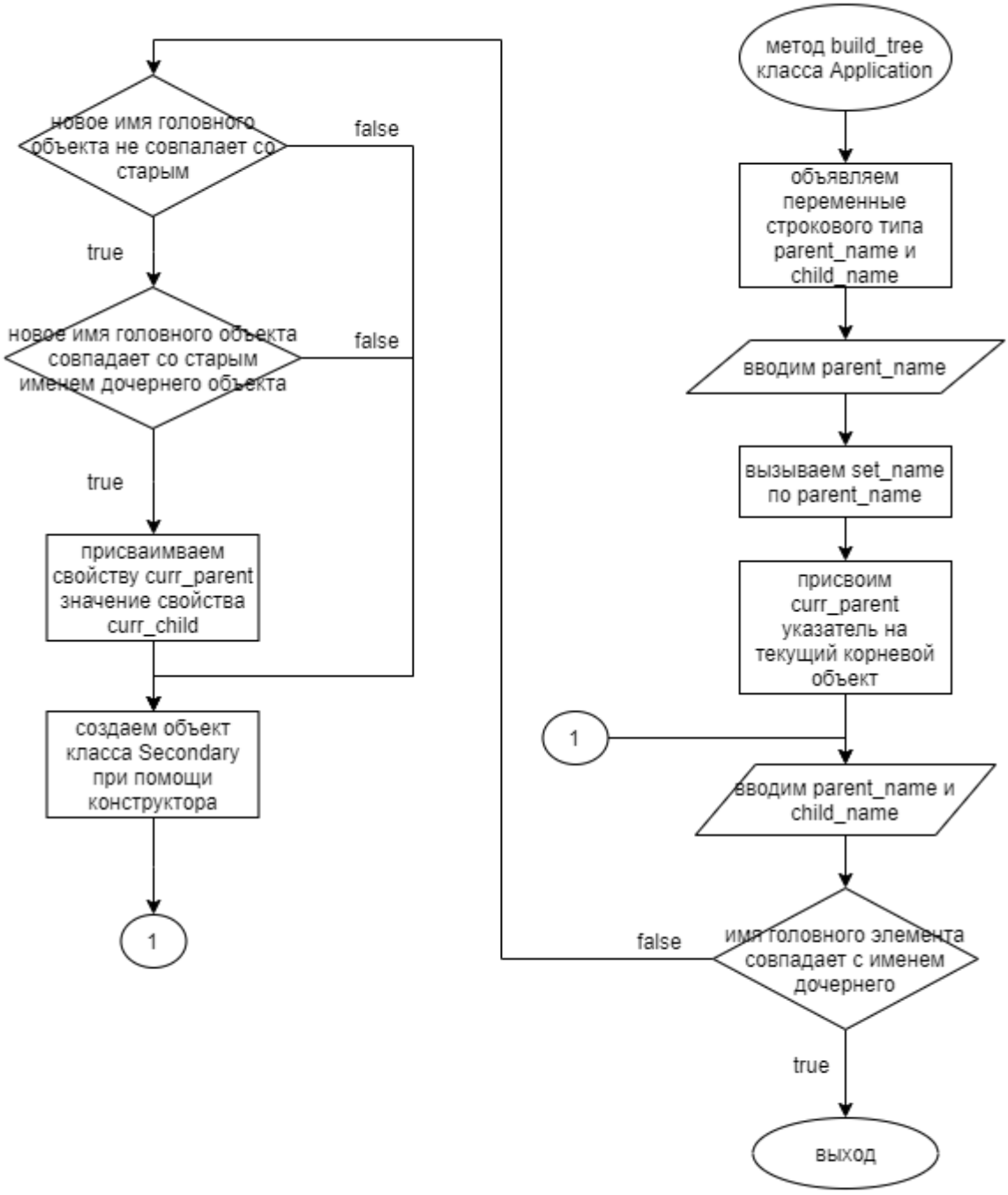


Рис. 4. Блок-схема алгоритма.

## Код программы

### Файл Application.cpp

```
#include "Application.h"
#include "Secondary.h"
using namespace std;
void Application::build_Tree()
{
        Object* curr_parent,* curr_child;
        string parent_name, child_name;
        cin >> parent_name;
        set_name(parent_name);
        curr_parent = this;
        cin >> parent_name >> child_name;
        while(true)
        {
                if(parent_name == child_name) return;
                if(parent_name != curr_parent -> get_name())
                        {
                                if(parent_name != curr_parent -> get_name())
                                {
                                        curr_parent = curr_child;
                                }
                                else
                                        continue;
                        }
                curr_child = new Secondary(curr_parent, child_name);
                cin >> parent_name >> child_name;
        }
}
Application::Application(Object* parent) : Object(parent) {}
int Application::exec()
{
        print();
        return 0;
}
```

### Файл Application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H
#include "Object.h"
class Application : public Object
{
public:
        void build_Tree();
        Application(Object* parent = 0);
        int exec();
};
```

```
#endif
```

## Файл main.cpp

```
#include "Application.h"
int main()
{
	setlocale(LC_ALL, "RUS");
	Application obj;
	obj.build_Tree();
	return obj.exec();
}
```

## Файл Object.cpp

```
#include "Object.h"
using namespace std;
Object::Object(Object* parent, string name)
{
	this->parent = NULL;
	this->name = "root";
	set_parent(parent);
	set_name(name);
}
void Object::set_parent(Object* p_parent)
{
	if(parent)
	{
		for(int i=0;i<parent->children.size();i++)
		{
			if(parent->children[i]==this)
				parent->children.erase(children.begin()+i);
		}
	}
	parent = p_parent;
	if(parent)
		parent->children.push_back(this);
}
Object* Object::get_parent()
{
	return this->parent;
}
void Object::set_name(string name)
{
```

```
        this->name=name;
}
string Object::get_name()
{
        return name;
}
void Object::print()
{
        if(!parent) cout << get_name();
        vector<Object*>::iterator children_iterator;
        if(children.empty()) return;
        cout << endl << name;
        children_iterator = children.begin();
        while(children_iterator != children.end())
        {
                cout << "  " << (*children_iterator) -> get_name();
                children_iterator++;
        }
        children_iterator--;
        (*children_iterator)->print();
}
Object::~Object()
{
        for(int i=0;i<children.size();i++)
        {
                delete children[i];
        }
}
```

## Файл Object.h

```
#ifndef OBJECT_H
#define OBJECT_H
#include <iostream>
#include <vector>
#include <string>
using namespace std;
class Object
{
        string name;
        Object* parent;
        vector<Object*> children;
public:
        Object(Object*, string = "root");
        void set_parent(Object*);
        Object* get_parent();
        void set_name(string);
        string get_name();
        void print();
        ~Object();
};
#endif
```

## Файл Secondary.cpp

```
#include <iostream>
#include <string>
#include "Secondary.h"
using namespace std;
Secondary::Secondary(Object* parent, string name) : Object(parent, name) {}
```

## Файл Secondary.h

```
#ifndef SECONDARY_H
#define SECONDARY_H
#include "Object.h"
class Secondary : public Object
{
public:
        Secondary(Object*, string);
};
#endif
```

## Тестирование

| Входные данные | Ожидаемые выходные данные | Фактические выходные данные |
|---|---|---|
| I I tired_to_do tired_to_do it_normal fail fail | I I tired_to_do tired_to_do it_normal | I I tired_to_do tired_to_do it_normal |
| 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 4 2.1 2.1 2.2 2.2 3.1 3.1 3.2 3.2 4.1 4.1 4.2 4.2 4.2.1 | 1 1 2 3 4 4 2.1 2.1 2.2 2.2 3.1 3.1 3.2 3.2 4.1 4.1 4.2 4.2 4.2.1 |

# Заключение

В ходе выполнения курсовой работы я научился:

- усовершенствовал навыки в обращении с указателями;
- научился работать с динамической памятью;
- усовершенствовал навыки в работе с большими программами;
- 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;
- разрабатывать алгоритм вывода дерева иерархии объектов по уровням;
- построению дерева иерархии объектов;
- освоил алгоритмы обработки структур данных в виде дерева.

Такой опыт позволит в будущем с лучшим пониманием решать задачи в роли программиста, а также более систематически подходить к тем или иным вопросам.

## 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://online-edu.mirea.ru/mod/resource/view.php?id=247030 (дата обращения 05.05.2021).

4. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

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).