МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

КУРСОВАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« КЛ_3_1 Вывод иерархического дерева »

С тудент группы	ИКБО-13-21	Дамарад Д.В.
Руководитель практики	Ассистент	Асадова Ю.С.

Работа представлена «___»_____2021 г. _____

(подпись студента)

Оценка _____

(подпись руководителя)

Москва 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....

Постановка задачи.....

Метод решения.....

Описание алгоритма.....

Блок-схема алгоритма.....

Код программы.....

Тестирование.....

ЗАКЛЮЧЕНИЕ.....

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ (ИСТОЧНИКОВ).....

ВВЕДЕНИЕ

Постановка задачи

Формирование и работа с иерархией объектов программы-системы.

Создание объектов и построение исходного иерархического дерева объектов. Система собирается из объектов, принадлежащих определенным классам. В тексте постановки задачи классу соответствует уникальный номер. Относительно номера класса определяются требования (свойства, функциональность).
Первоначальная сборка системы (дерева иерархии объектов, программы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Корневой объект объявляется в основной программе (main).
Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:
«Наименование головного объекта»«Наименование очередного объекта»«Номер класса принадлежности очередного объекта»
Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Вывод иерархического дерева объектов на консоль
Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных программах динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

В данной задаче подразумевается, что наименования объектов уникальны. Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Расширить функциональность базового класса:
- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr);
- метод вывода дерева иерархии объектов;
- метод вывода дерева иерархии объектов и отметок их готовности;
- метод установки готовности объекта реализовать (доработать) следующим образом.

Готовность для каждого объекта устанавливается индивидуально. Готовность задается посредством любого отличного от нуля целого числового значения, которое присваивается свойству состояния объекта. Объект переводится в состояние готовности, если все объекты вверх по иерархии до корневого включены, иначе установка готовности игнорируется.
При отключении головного, отключаются все объекты от него по иерархии вниз по ветке. Свойству состояния объекта присваивается значение нуль.

Разработать программу:
1. Построить дерево объектов системы (в методе коневого объекта построения исходного дерева объектов).
2. В методе корневого объекта запуска моделируемой системы реализовать:
2.1. Вывод на консоль иерархического дерева объектов в следующем виде:

root
    ob_1
        ob_2
    ob_3
        ob_4
            ob_5
        ob_6
            ob_7

где: root - наименование корневого объекта (приложения).

2.2. Переключение готовности объектов согласно входным данным (командам).
2.3. Вывод на консоль иерархического дерева объектов и отметок их готовности в следующем виде:
root is ready
    ob_1 is ready
        ob_2 is ready
    ob_3 is ready
        ob_4 is not ready
            ob_5 is not ready
        ob_6 is ready
            ob_7 is not ready

Описание входных данных
Множество объектов, их характеристики и расположение на дереве иерархии. Последовательность ввода организовано так, что головной объект для

очередного вводимого объекта уже присутствует на дереве иерархии объектов.

Первая строка

«Наименование корневого объекта»

Со второй строки

«Наименование головного объекта»«Наименование очередного объекта»«Номер класса принадлежности очередного объекта»

. . . . .

endtree

Со следующей строки вводятся команды включения или отключения объектов

«Наименование объекта»«Номер состояния объекта»

. . . . .

Признаком завершения ввода является конец потока входных данных.

Пример ввода

```
app_root
app_root object_01 3
app_root object_02 2
object_02 object_04 3
object_02 object_05 5
object_01 object_07 2
endtree
```

```
app_root 1
object_07 3
object_01 1
object_02 -2
object_04 1
```

Описание выходных данных

Вывести иерархию объектов в следующем виде:

```
Object tree
«Наименование корневого объекта»
 «Наименование объекта 1»
  «Наименование объекта 2»
 «Наименование объекта 3»
. . . . .
The tree of objects and their readiness
«Наименование корневого объекта»«Отметка готовности»
 «Наименование объекта 1»«Отметка готовности»
  «Наименование объекта 2»«Отметка готовности»
 «Наименование объекта 3»«Отметка готовности»
. . . . .
```

«Отметка готовности» - равно «is ready» или «is not ready»
Отступ каждого уровня иерархии 4 позиции.

Пример вывода

```
Object tree
```

app_root

object_01

object_07

object_02

object_04

object_05

The tree of objects and their readiness

app_root is ready

object_01 is ready

object_07 is not ready

object_02 is ready

object_04 is ready

object_05 is not ready

Метод решения

Для решения поставленной задачи используются:

- Объекты стандартных потоков ввода и вывода cin и cout соответственно для ввода и вывода на экран.
- Объект app класса App.
- Библиотека контейнеров vector для создания вектора, который будет хранить в себе указатели на дочерние объекты.
- Библиотека string для создания переменных строкового типа.
- Класс Base, являющийся базовым классом для классов App, Class2, Class3, Class4, Class5, Class6.
- Классы Class2, Class3, Class4, Class5, Class6, которые являются основными классами для подчиненных объектов.

Изменения, внесенные в архитектуру программы предыдущей программы:

4_1_1:

- Изменены:
    - Конструктор класса Base.
    - Метод PrintTree класса Base.
    - Метода BuildTree класса App.
    - Метода exec класса App.
- Добавлены:
    - Поле status у класса Base.
    - Метод FindPtr класса Base.
    - Метод SetStatus класса Base.
    - Метод ShowReadiness класса Base.
    - Метод SetChildStatus класса Base.

        - Классы Class2, Class3, Class4, Class5, Class6.
        - Делегирующий конструкторы классов Class2, Class3, Class4, Class5, Class6.
    - Оставлены без изменений:
        - Метод SetName класса Base.
        - Метод GetName класса Base.
        - Метод SetHead класса Base.
        - Метод GetHead класса Base.

Класс Base:

- Поля:
    - Поле, хранящее имя объекта:
        - Наименование - name;
        - Тип - string;
        - Модификатор доступа - protected.
    - Поле, хранящее указатель на объект:
        - Наименование - head;
        - Тип - указатель на класс Base;
        - Модификатор доступа - protected.
    - Поле, хранящие статус готовности объекта:
        - Наименование - status;
        - Тип - int;
        - Модификатор доступа - protected.
    - Поле, хранящее контейнер указателей на дочерние объекты:
        - Наименование - children;
        - Тип - контейнер vector с указателями на класс Base;
        - Модификатор доступа - public.

- Методы:
    - Метод Base:
        - Функционал - параметризированный конструктор.
    - Метод SetName:
        - Функционал - присвоение имени объекта.
    - Метод GetName:
        - Функционал - получение поля имени объекта.
    - Метод SetHead:
        - Функционал - установка указателя на головной объект.
    - Метод GetHead:
        - Функционал - получение поля указателя на головной объект.
    - Метод PrintTree:
        - Функционал - вывод дерава иерархии.
    - Метод FindPtr:
        - Функционал - поиск указателя на объект по имени.
    - Метод SetStatus:
        - Функционал - установка статуса объектам в созданной иерархии.
    - Метод ShowReadiness:
        - Функционал - Вывод дерева готовности объектов.
    - Метод SetChildStatus:
        - Функционал - установка статуса подчиненным объектам.

Класс App:

- Поля:
    - Отсутсвуют.
- Методы:

        - Метод App:
            - Функционал - делегирующий конструктор.
        - Метод BuildTree:
            - Функционал - построение дерева иерархии.
        - Метод exec:
            - Функционал - метода запуска приложения.

Класс Class2:

- Поля:
    - Отсутсвуют.
- Методы:
    - Метод Class2:
        - Функционал - делегирующий конструктор.

Класс Class3:

- Поля:
    - Отсутсвуют.
- Методы:
    - Метод Class3:
        - Функционал - делегирующий конструктор.

Класс Class4:

- Поля:
    - Отсутсвуют.
- Методы:
    - Метод Class4:

- Функционал - делегирующий конструктор.

Класс Class5:

- Поля:
  - Отсутсвуют.
- Методы:
  - Метод Class5:
    - Функционал - делегирующий конструктор.

Класс Class6:

- Поля:
  - Отсутсвуют.
- Методы:
  - Метод Class6:
    - Функционал - делегирующий конструктор.

№	Название класса	Класс-наследник	Модификатор доступа	Описание	Номер	Комментарий
1	Base	App	public	Базовый класс	2	
		Class2	public		3	
		Class3	public		4	
		Class4	public		5	
		Class5	public		6	
		Class6	public		7	
2	App			Класс объектов, подчиненных классу Base		
3	Class2			Класс объектов,		

				подчиненных классу Base		
4	Class3			Класс объектов, подчиненных классу Base		
5	Class4			Класс объектов, подчиненных классу Base		
6	Class5			Класс объектов, подчиненных классу Base		
7	Class6			Класс объектов, подчиненных классу Base		

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: Отсутсвуют

Возвращаемое значение: Целочисленное значение - код возврата

Алгоритм функции представлен в таблице 2.

Таблица 2. Алгоритм функции main

№	Предикат	Действия	№ перехода	Комментарий
1		Инициализация объекта app типа App и передача в его конструктор указатель nullptr	2	
2		Вызов метода BuildTree у объекта app	3	
3		Возврат результата выполнения метода exec у объекта app	∅	

Конструктор класса: Base

Модификатор доступа: public

Функционал: Параметризированный конструктор

Параметры: Указатель на объект типа Base, строка - название объекта

Алгоритм конструктора представлен в таблице 3.

Таблица 3. Алгоритм конструктора класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Вызов метода SetName с параметром name	2	
2		Вызов метода SetHead с параметром head	3	
3		Присваивание полю status значение 0	4	
4	Головной укзатель на объект не равен 0	Запись в вектор children указателя на вызываемый объект	∅	
			∅	

Класс объекта: Base

Модификатор доступа: public

Метод: SetName

Функционал: Сеттер поля name

Параметры: строка - новое имя объекта

Возвращаемое значение: Отсутсвует

Алгоритм метода представлен в таблице 4.

Таблица 4. Алгоритм метода SetName класса Base

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

Класс объекта: Base

Модификатор доступа: public

Метод: GetName

Функционал: Геттер поля name

Параметры: Отсутсвуют

Возвращаемое значение: строка - имя объекта

Алгоритм метода представлен в таблице 5.

Таблица 5. Алгоритм метода GetName класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат поля name	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: SetHead

Функционал: Сеттер полля head

Параметры: Указатель на тип Base

Возвращаемое значение: Отсутсвует

Алгоритм метода представлен в таблице 6.

Таблица 6. Алгоритм метода SetHead класса Base

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

Класс объекта: Base

Модификатор доступа: public

Метод: GetHead

Функционал: Геттер поля head

Параметры: Отсутсвуют

Возвращаемое значение: Указатель на тип Base

Алгоритм метода представлен в таблице 7.

Таблица 7. Алгоритм метода GetHead класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Возврат поля head	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: PrintTree

Функционал: Вывод дерева иерархии

Параметры: Строка - space - кол-во пробелов

Возвращаемое значение: Отсутсвует

Алгоритм метода представлен в таблице 8.

Таблица 8. Алгоритм метода PrintTree класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	Цикл со счетчиком i		2	
			5	
2	Элемент в векторе детей также имеет своих детей		3	

		Вывод на экран space+4 пробела, вывод итога выполнения метода GetName у элемента индекса i, переход на следующую строку	5	
3		Вывод на экран space+4 пробела, вывод итога выполнения метода GetName у элемента индекса i, переход на следующую строку	4	
4		Вызов метода PrintTree у элемента индекса i	5	
5	i меньше размера вектора	Увеличение i на 1	1	
			∅	

Класс объекта: Base

Модификатор доступа: public

Метод: FindPtr

Функционал: Поиск указателя на объект по имени

Параметры: Строка name, указатель ptr на класс Base

Возвращаемое значение: Указатель на класс Base

Алгоритм метода представлен в таблице 9.

Таблица 9. Алгоритм метода FindPtr класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	ptr не равен 0		2	
		Возврат передаваемого указателя	∅	

2		Объявление указателя temp на тип Base	3	
3	Цикл со счетчиком i	Присвоение temp указателя на ребенка индекса i	4	
			∅	
4	Метода GetName по указателю temp вернул то же название, что требовалось найти	Возврат указателя temp	∅	
			5	
5	У объекта по указателю temp также есть свои дети		6	
			7	
6	Метода GetName с передаваемым значением temp вернул не нулевой указатель	Возврат итога выполнения метода GetName с передаваемым указателем temp,конец цикла	∅	
			7	
7	i меньше размера вектора	Прибавление к i единицы	3	
		Возврат нулевого указателя	∅	

Класс объекта: Base

Модификатор доступа: public

Метод: SetStatus

Функционал: Метод установки готовности объектов

Параметры: Отсутсвуют

Возвращаемое значение: Отсутсвует

Алгоритм метода представлен в таблице 10.

Таблица 10. Алгоритм метода SetStatus класса Base

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление строковой переменной name и целочисленной переменной status	2	
2	Идет ввод с клавиатуры		3	
			∅	
3	Вводится имя не корневого объекта		4	
			8	
4		Инициализация указателя temp на тип Base с присвоением ему значения возвращаемого от метода FindPtr с передаваемыми параметрами name и this	5	
5	Указатель temp не равен 0		6	
			2	
6	Указатель на родителя имеет нулевой статус	Присвоение объекту по указателю temp поле status значение 0	7	
		Присвоение объекту по указателю temp поле status значение status	7	
7	Объект по указателю temp имеет свой вектор детей и его поле status равно 0	Вызов метода SetChildStatus по указзателю temp	2	

			2	
8		Присвоение корневому объекту поле status значение 0. Вызов метода SetChildStatus у корневого объекта	2	

Класс объекта: Base

Модификатор доступа: public

Метод: ShowReadiness

Функционал: Вывод дерева объектов готовности

Параметры: Отсутсвуют

Возвращаемое значение: Отсутсвует

Алгоритм метода представлен в таблице 11.

Таблица 11. Алгоритм метода ShowReadiness класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	Цикл со счетчиком		2	
			∅	
2	Элемент с индексом i имеет свой вектор детей		3	
			6	
3		Вывод на экран пробелов, вызов метода GetName по указателю	4	
4	Элемент имеет 0 статус	Вывод на экран сообщения " is not ready", переход на новую строку	5	

		Вывод на экран сообщения " is ready", переход на новую строку	5	
5		Вызов метода ShowReadiness у элемента по индексу i	8	
6		Вывод на экран пробелов, вызов метода GetName по указателю	7	
7	Элемент имеет статус 0	Вывод на экран сообщения " is not ready", переход на новую строку	8	
		Вывод на экран сообщения " is ready", переход на новую строку	8	
8	i меньше размера вектора children	Увеличение i на 1	1	
			1	

Класс объекта: Base

Модификатор доступа: public

Метод: SetChildStatus

Функционал: Установка статуса подчиненным объектам

Параметры: Целочисленный тип данных - status

Возвращаемое значение: Отсутсвует

Алгоритм метода представлен в таблице 12.

Таблица 12. Алгоритм метода SetChildStatus класса Base

№	Предикат	Действия	№ перехода	Комментарий
1	Цикл со счетчиком i		2	
			∅	

2		Установка статуса элементу children по индексу i	3	
3	Элемент children с индексом i имеет свой вектор детей	Вызов метода SetChildStatus у элемента children с индексом i	4	
			4	
4	i меньше размера вектора children	Увеличение i на 1	1	
			1	

Класс объекта: App

Модификатор доступа: public

Метод: BuildTree

Функционал: Построение дерева иерархии

Параметры: Отсутсвуют

Возвращаемое значение: Отсутсвует

Алгоритм метода представлен в таблице 13.

Таблица 13. Алгоритм метода BuildTree класса App

№	Предикат	Действия	№ перехода	Комментарий
1		Объявление строковых переменных a,b и целочисленной переменной cl	2	
2		Считывание с клавиатуры значения переменной a	3	
3		Вызов у данного объекта метода SetName c	4	

		параметром a		
4	Бесконечный цикл	Считывание с клавиатуры значений переменной a	5	
			5	
5	Значение a не равно "endtree"	Считывание с клавиатуры значений переменных b,cl	6	
			∅	
6	Итог вызовы метода GetName равняется значению a	Создание объекта класса в зависимости от значения переменной cl	4	
			7	
7		Инициализация переменной temp указателя на класс Base с присвоением в нее итога выполнения метода FindPtr	8	
8	temp не нулевой укзатель	Создание объекта класса в зависимости от значения переменной cl	4	
			4	

Класс объекта: App

Модификатор доступа: public

Метод: exec

Функционал: Метода запуска приложения

Параметры: Отсутсвуют

Возвращаемое значение: Целочисленный значение - код возврата

Алгоритм метода представлен в таблице 14.

Таблица 14. Алгоритм метода exec класса App

№	Предикат	Действия	№ перехода	Комментарий
1		Вывод на экран строки "Object tree" и переход на новую строку	2	
2		Вывод на экран итога выполнения метода GetName и переход на новую строку	3	
3		Вызов метода PrintTree с передаваемым параметром - пустая строка	4	
4		Вызов метода SetStatus.	5	
5		Вызов строки "The of objects and their readiness" и переход на новую строку	6	
6		Вывод на экран итога выполнения метода GetName	7	
7	Статус готовности объекта равен 0	Вывод строки " is not ready" и переход на новую строку	8	
		Вывод строки " is ready" и переход на новую строку	8	
8		Вызов метода ShowRadiness с передаваемым параметром - пустая строка	∅	

Конструктор класса: App

Модификатор доступа: public

Функционал: Делегирующий конструктор

Параметры: Указатель на объект типа Base, строка - имя объекта

Алгоритм конструктора представлен в таблице 15.

Таблица 15. Алгоритм конструктора класса App

№	Предикат	Действия	№ перехода	Комментарий
1		Передача параметров head, name в констуктор Base	∅	

Конструктор класса: Class2

Модификатор доступа: public

Функционал: Делегирующий конструктор

Параметры: Указатель на объект типа Base, строка - имя объекта

Алгоритм конструктора представлен в таблице 16.

Таблица 16. Алгоритм конструктора класса Class2

№	Предикат	Действия	№ перехода	Комментарий
1		Передача параметров ptr, name в констуктор Base	∅	

Конструктор класса: Class3

Модификатор доступа: public

Функционал: Делегирующий конструктор

Параметры: Указатель на объект типа Base, строка - имя объекта

Алгоритм конструктора представлен в таблице 17.

Таблица 17. Алгоритм конструктора класса Class3

№	Предикат	Действия	№ перехода	Комментарий
1		Передача параметров ptr, name в констуктор Base	∅	

Конструктор класса: Class4

Модификатор доступа: public

Функционал: Делегирующий конструктор

Параметры: Указатель на объект типа Base, строка - имя объекта

Алгоритм конструктора представлен в таблице 18.

Таблица 18. Алгоритм конструктора класса Class4

№	Предикат	Действия	№ перехода	Комментарий
1		Передача параметров ptr, name в констуктор Base	∅	

Конструктор класса: Class5

Модификатор доступа: public

Функционал: Делегирующий конструктор

Параметры: Указатель на объект типа Base, строка - имя объекта

Алгоритм конструктора представлен в таблице 19.

Таблица 19. Алгоритм конструктора класса Class5

№	Предикат	Действия	№ перехода	Комментарий
1		Передача параметров ptr, name в констуктор Base	∅	

Конструктор класса: Class6

Модификатор доступа: public

Функционал: Делегирующий конструктор

Параметры: Указатель на объект типа Base, строка - имя объекта

Алгоритм конструктора представлен в таблице 20.

Таблица 20. Алгоритм конструктора класса Class6

№	Предикат	Действия	№ перехода	Комментарий
1		Передача параметров ptr, name в констуктор Base	∅	

Блок-схема алгоритма

Представим описание алгоритмов в графическом виде на рисунках ниже.

Рис. 1. Блок-схема алгоритма.

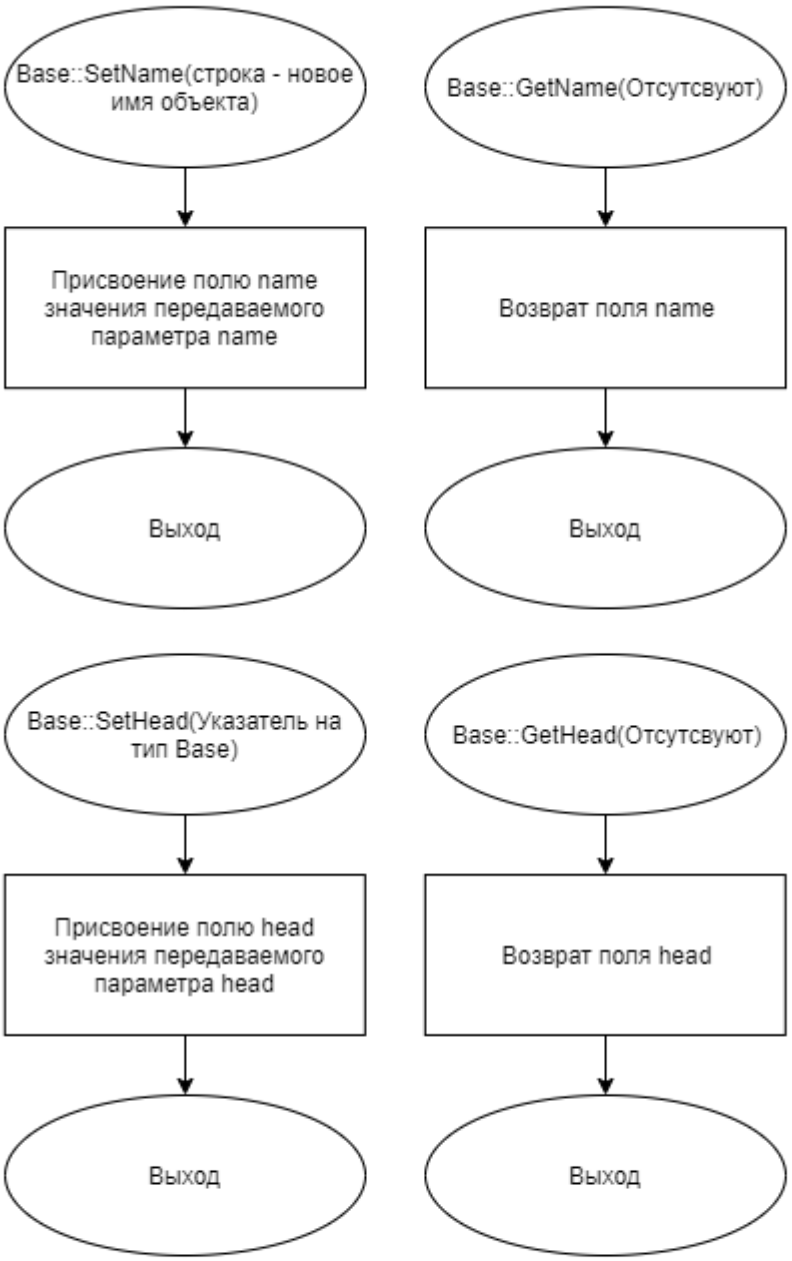

Рис. 2. Блок-схема алгоритма.

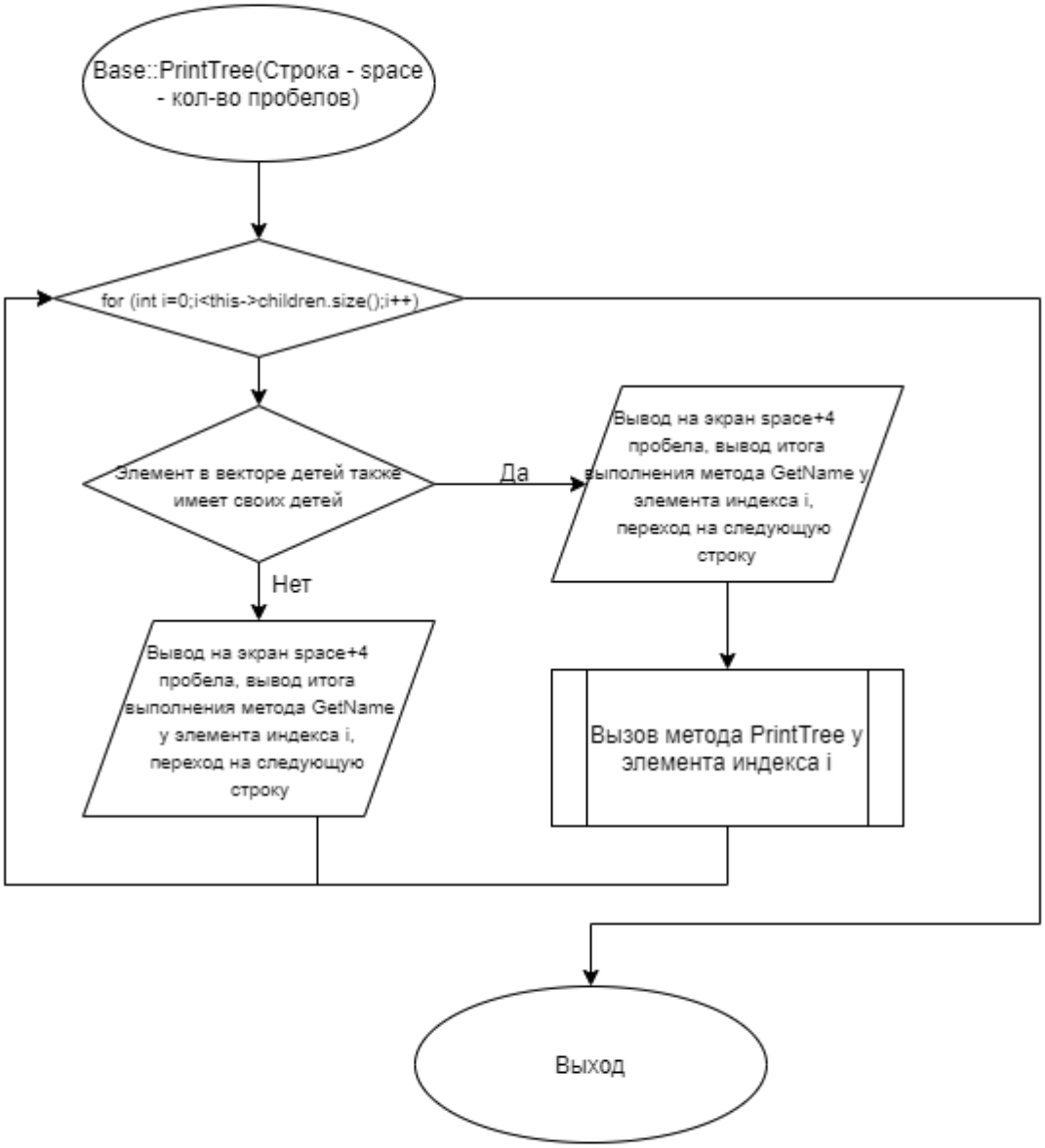

Рис. 3. Блок-схема алгоритма.

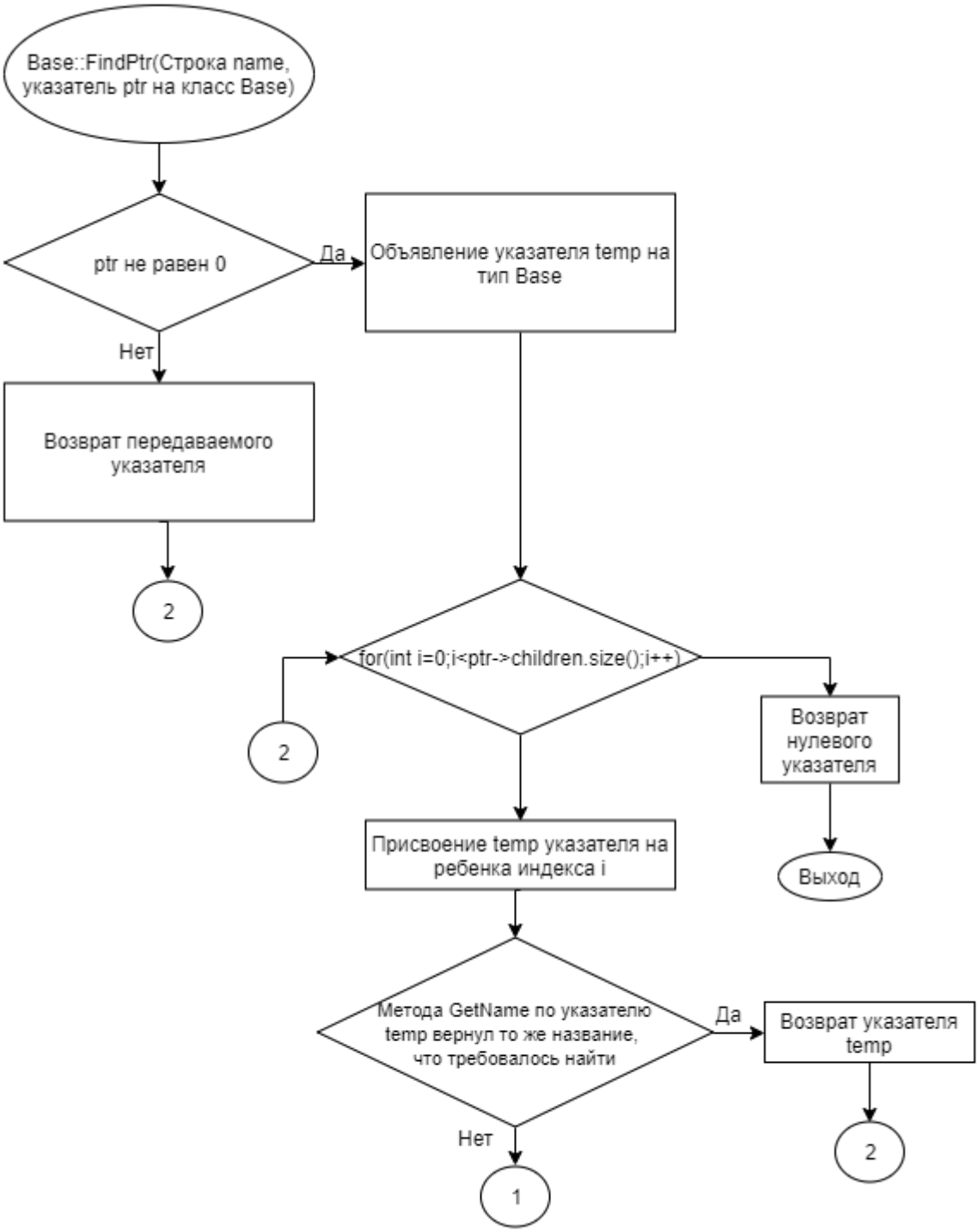

Рис. 4. Блок-схема алгоритма.

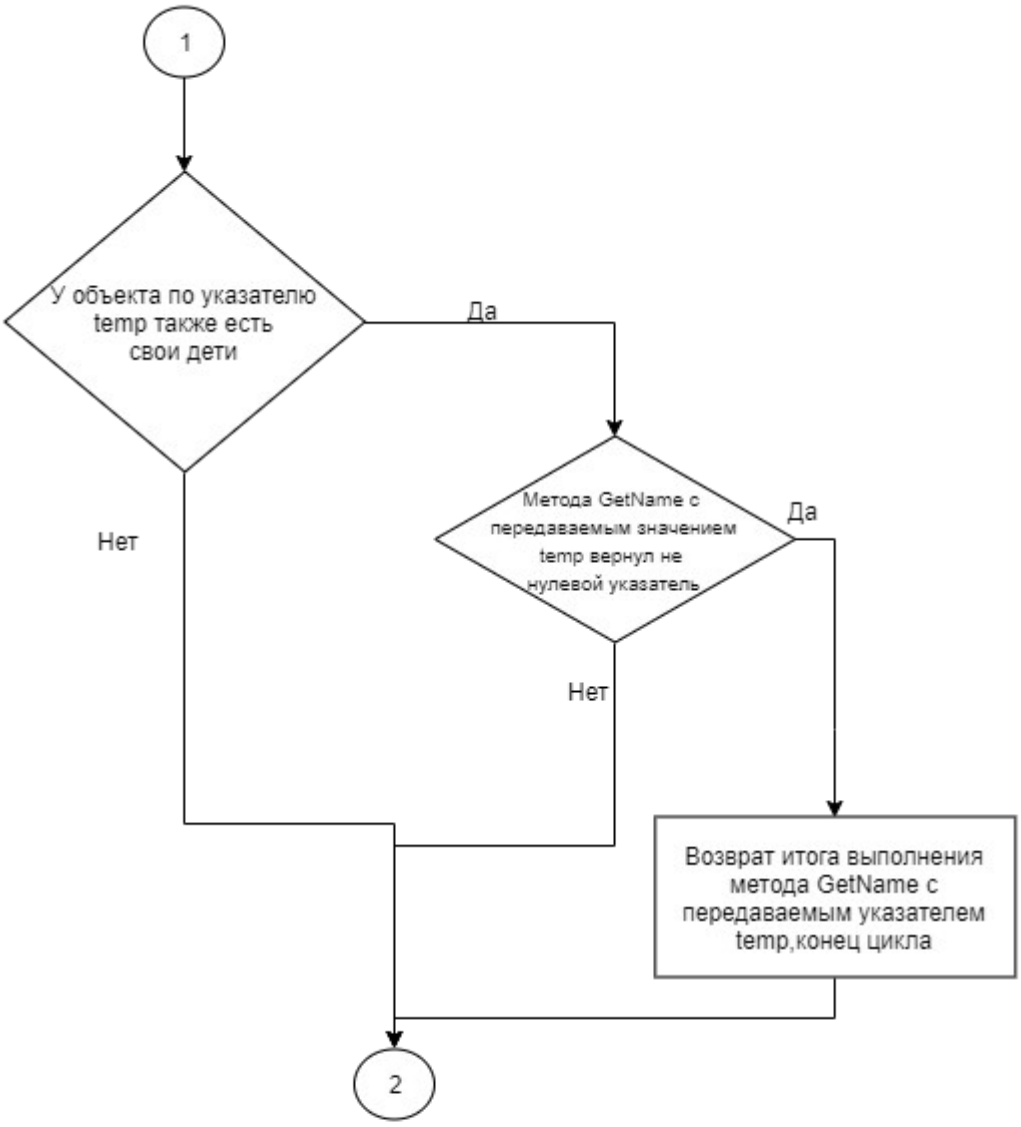

Рис. 5. Блок-схема алгоритма.

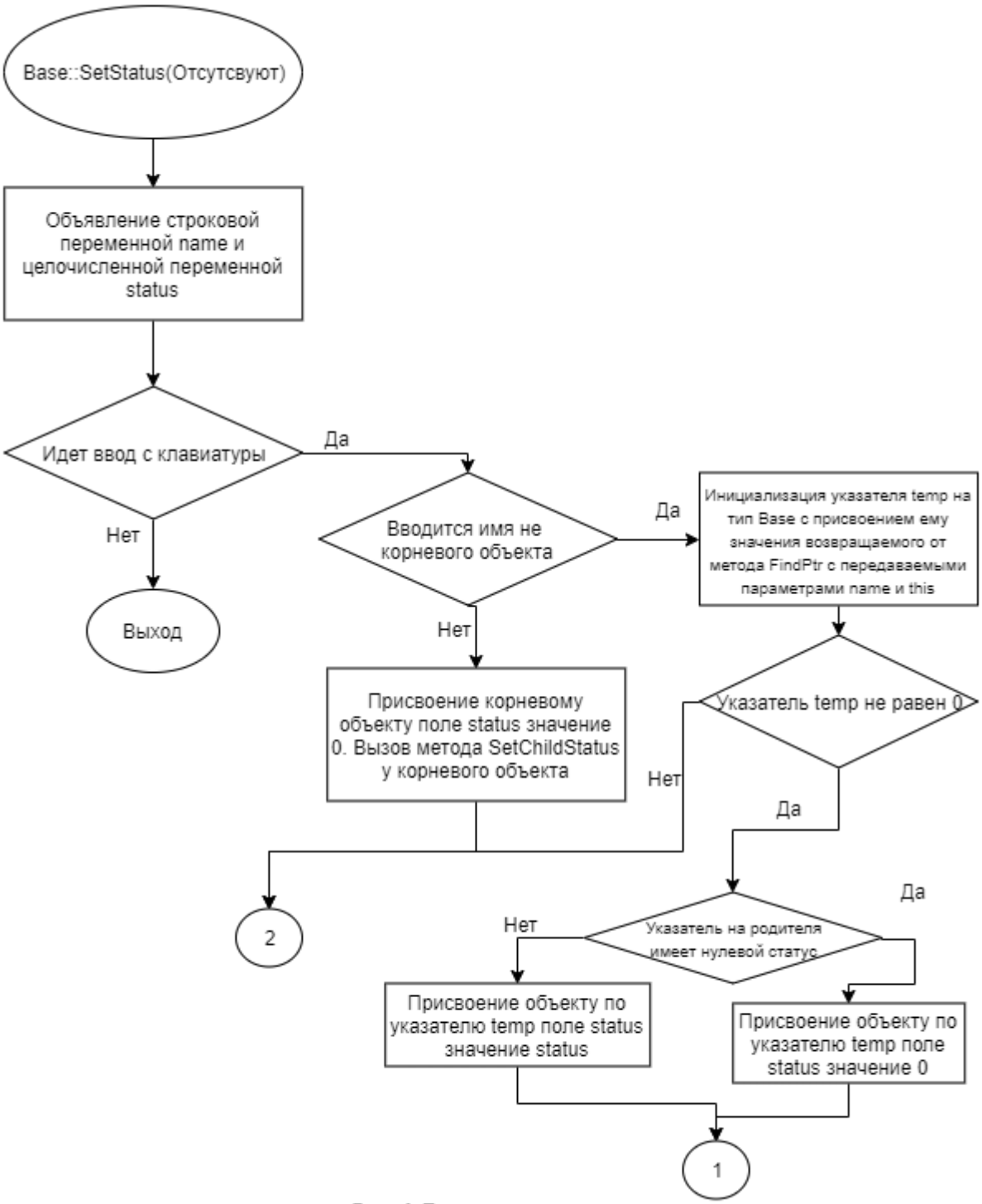

Рис. 6. Блок-схема алгоритма.

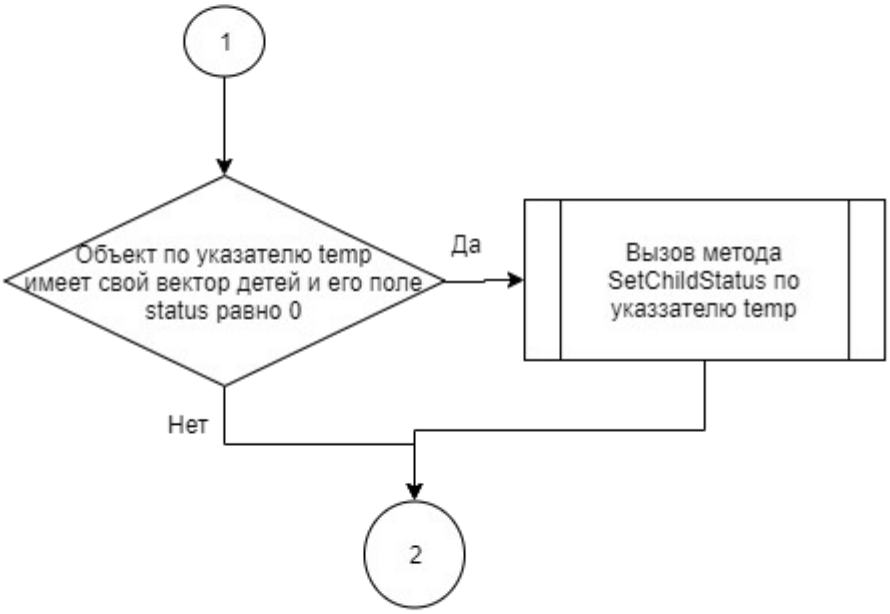

Рис. 7. Блок-схема алгоритма.

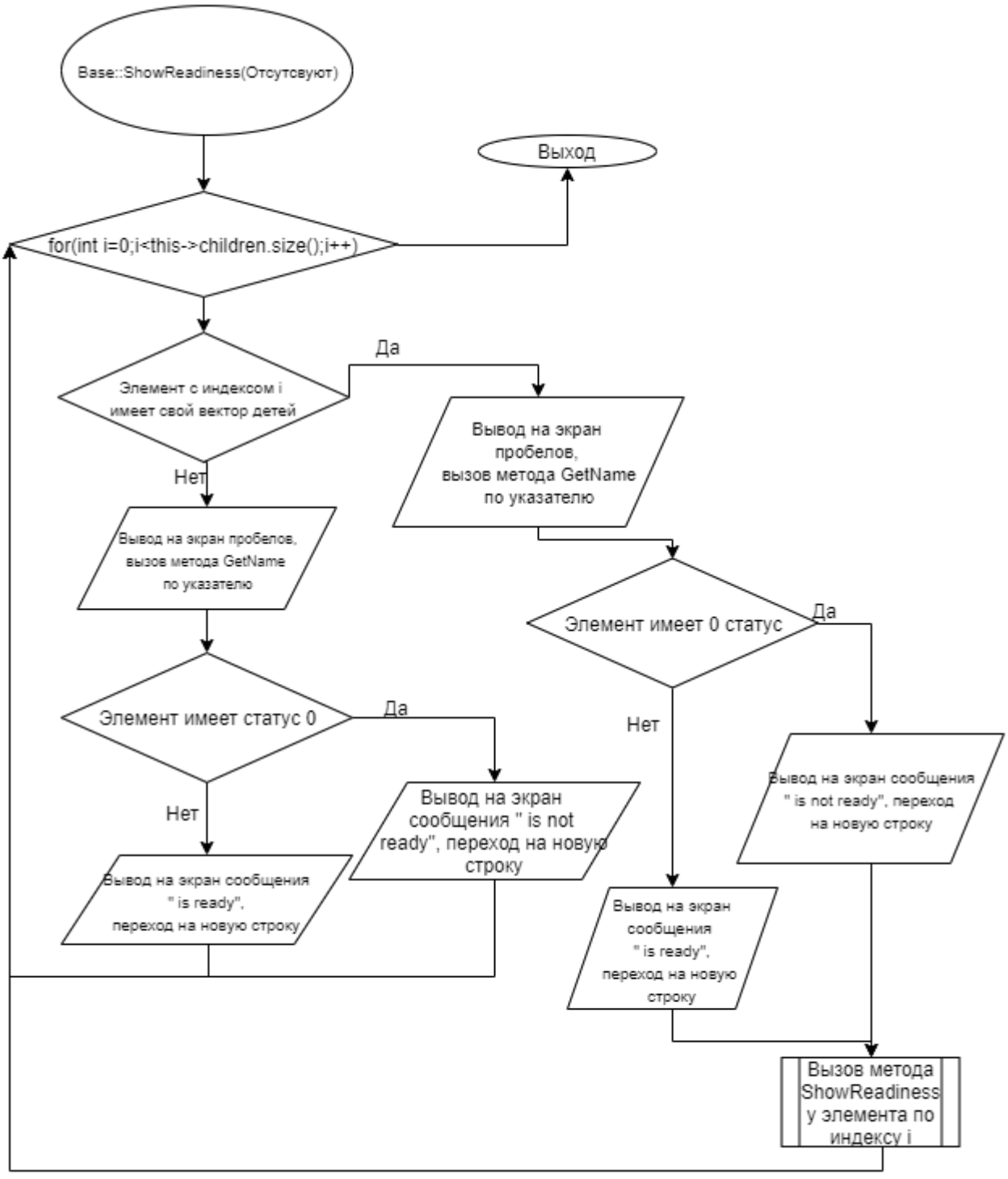

Рис. 8. Блок-схема алгоритма.

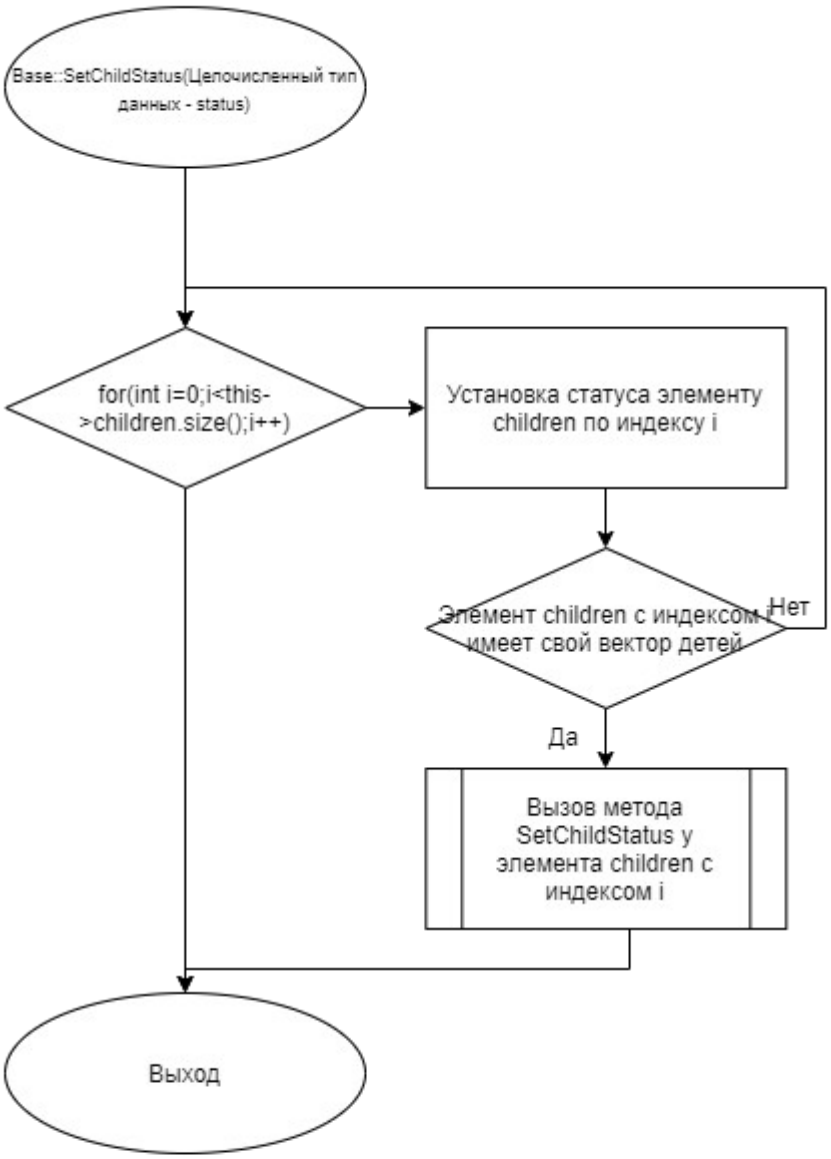

Рис. 9. Блок-схема алгоритма.

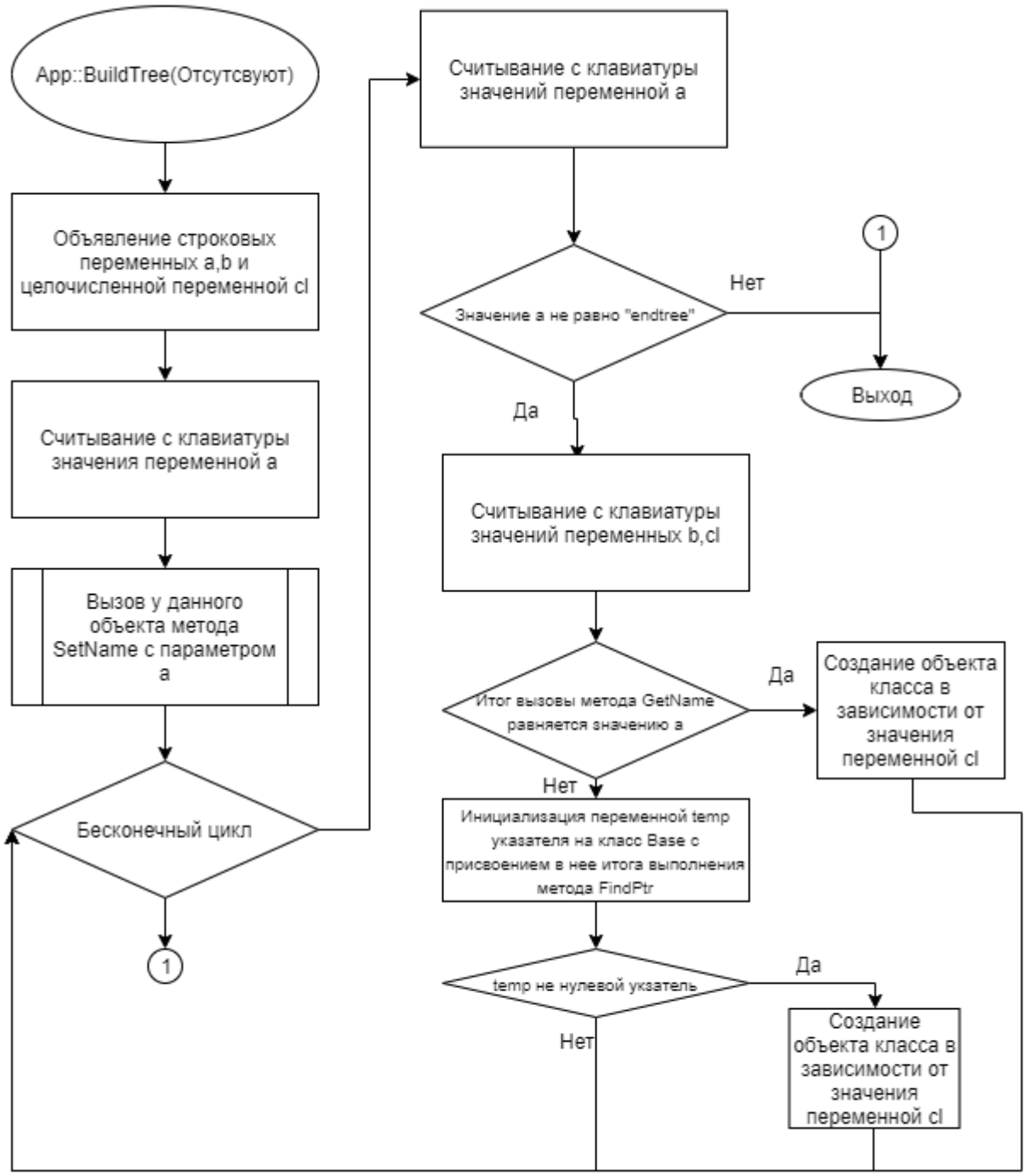

Рис. 10. Блок-схема алгоритма.

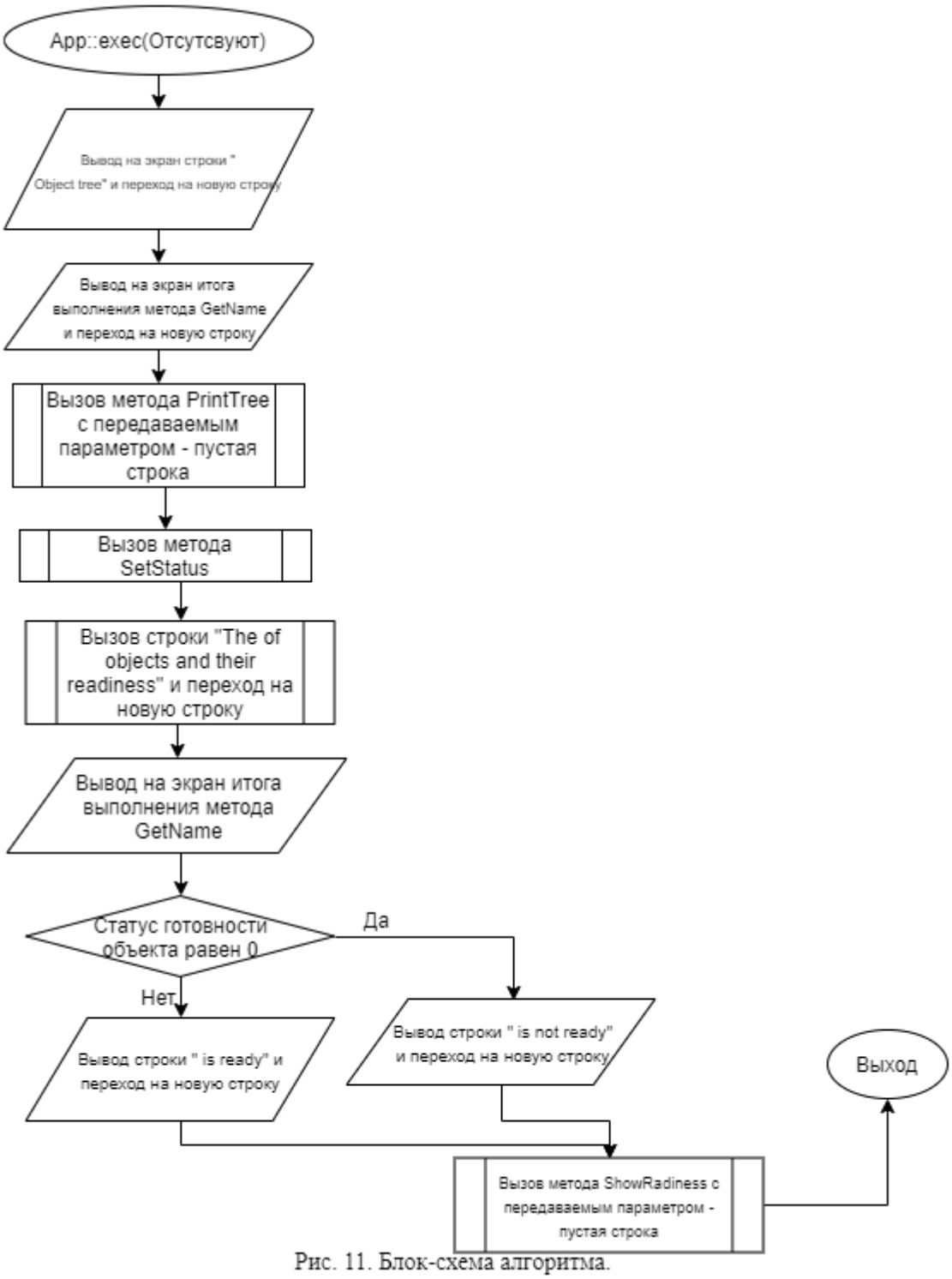

Рис. 11. Блок-схема алгоритма.

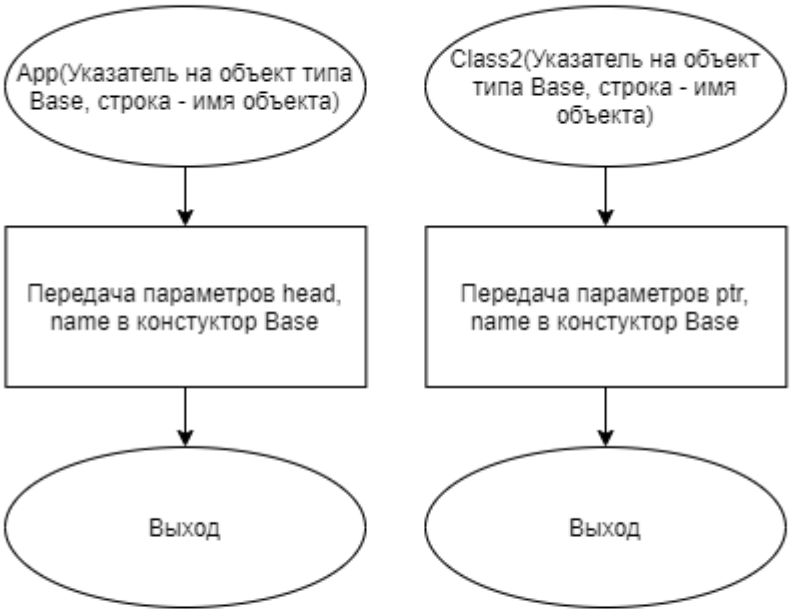

Рис. 12. Блок-схема алгоритма.

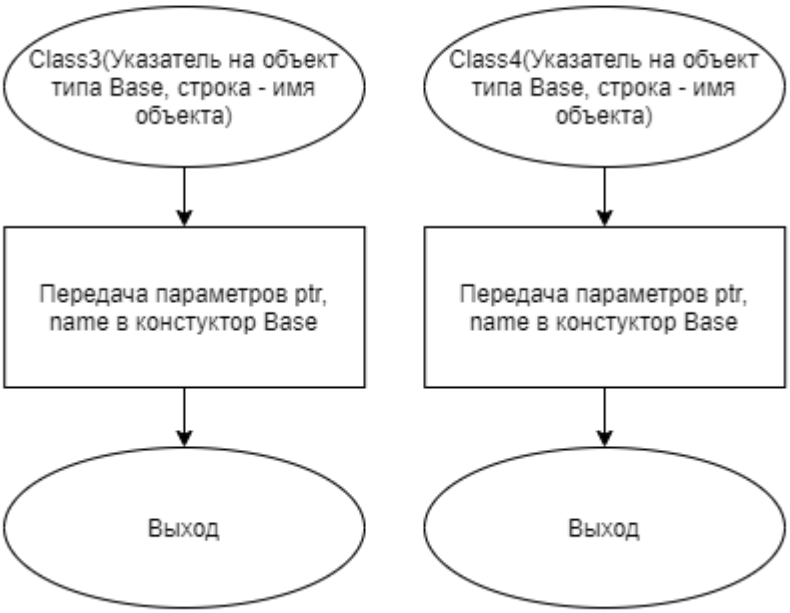

Рис. 13. Блок-схема алгоритма.

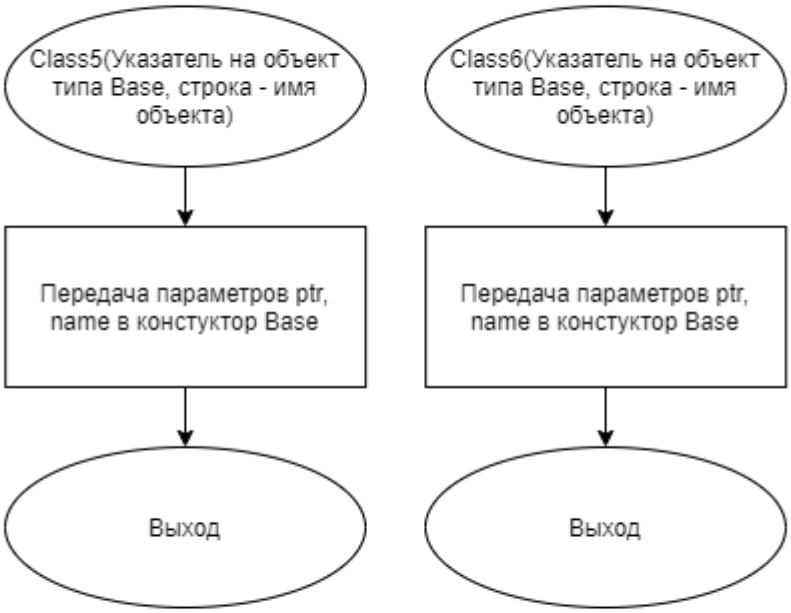

Рис. 14. Блок-схема алгоритма.

Код программы

Программная реализация алгоритмов для решения задачи представлена ниже.

Файл App.cpp

```
#include "App.h"
#include "Class2.h"
#include "Class3.h"
#include "Class4.h"
#include "Class5.h"
#include "Class6.h"
#include <string>
#include <iostream>
using namespace std;
int App::exec(){
        cout<<"Object tree"<<endl;
        cout<<this->GetName()<<endl;
        PrintTree("");
        SetStatus();
        cout<<"The tree of objects and their readiness"<<endl;
        cout<<this->GetName();
        if (this->status==0){
                cout<<" is not ready";
        }
        else{
                cout<<" is ready";
        }
        ShowReadiness("");
        return 0;
}
void App::BuildTree(){
        string a,b;
        int cl;
        cin>>a;
        this->SetName(a);
        while(true){
                cin>>a;
                if (a=="endtree"){
                        break;
                }
                cin>>b>>cl;
                if (this->GetName()==a){
                        if (cl==2) new Class2(this, b);
                        if (cl==3) new Class3(this, b);
                        if (cl==4) new Class4(this, b);
                        if (cl==5) new Class5(this, b);
                        if (cl==6) new Class6(this, b);
                }
                else{
                        Base* temp=FindPtr(a, this);
```

```
                        if (temp!=nullptr){
                                if (cl==2) new Class2(temp, b);
                                if (cl==3) new Class3(temp, b);
                                if (cl==4) new Class4(temp, b);
                                if (cl==5) new Class5(temp, b);
                                if (cl==6) new Class6(temp, b);
                        }
                }
        }
}
```

Файл App.h

```
#ifndef APP_H
#define APP_H
#include "Base.h"
class App : public Base{
public:
        App(Base* head, string name=""):Base(head, name){};
        void BuildTree();
        int exec();
};
#endif
```

Файл Base.cpp

```
#include "Base.h"
#include <iostream>
using namespace std;
Base::Base(Base* head, string name){
        this->name=name;
        this->head=head;
        this->status=0;
        if (head!=nullptr){
                head->children.push_back(this);
        }
}
void Base::SetName(string name){
        this->name=name;
}
string Base::GetName(){
        return name;
}
void Base::SetHead(Base* head){
        this->head=head;
}
Base* Base::GetHead(){
        return head;
}
void Base::PrintTree(string space){
        for(int i=0;i<this->children.size();i++){
```

```
                if (children[i]->children.size()>0){
                        cout<<space+"    "<<children[i]->GetName()<<endl;
                        ((Base*)children[i])->PrintTree(space+"    ");
                }
                else{
                        cout<<space+"    "<<children[i]->GetName()<<endl;
                }
        }
}
Base* Base::FindPtr(string name, Base* ptr){
        if(ptr!=nullptr){
                Base* temp;
                for(int i=0;i<ptr->children.size();i++){
                        temp=ptr->children[i];
                        if(temp->GetName()==name){
                                return temp;
                        }
                        else if(temp->children.size()>0){
                                if(FindPtr(name,temp)!=nullptr){
                                        return FindPtr(name,temp);
                                        break;
                                }
                        }
                }
                return nullptr;
        }
        return nullptr;
}
void Base::SetStatus(){
        string name; int status;
        while(cin>>name>>status){
                if(this->GetName()!=name){
                        Base* temp=FindPtr(name,this);
                        if (temp!=nullptr){
                                if (temp->GetHead()->status==0){
                                        temp->status=0;
                                }
                                else{
                                        temp->status=status;
                                }
                                if(temp->children.size()>0 and status==0){
                                        temp->SetChildStatus(status);
                                }
                        }
                }
                else{
                        this->status=status;
                        if(this->children.size()>0 and status==0){
                                this->SetChildStatus(status);
                        }
                }
        }
}
void Base::ShowReadiness(string space){
        for(int i=0;i<this->children.size();i++){
                if (children[i]->children.size()>0){
                        cout<<endl;
```

```
                        cout<<space+"    "<<children[i]->GetName();
                        if(children[i]->status==0){
                                cout<<" is not ready";
                        }
                        else{
                                cout<<" is ready";
                        }
                        ((Base*)children[i])->ShowReadiness(space+"    ");
                }
                else{
                        cout<<endl;
                        cout<<space+"    "<<children[i]->GetName();
                        if(children[i]->status==0){
                                cout<<" is not ready";
                        }
                        else{
                                cout<<" is ready";
                        }
                }
        }
}
void Base::SetChildStatus(int status){
        for(int i=0;i<this->children.size();i++){
                children[i]->status=status;
                if(this->children.size()>0 and status==0){
                        children[i]->SetChildStatus(status);
                }
        }
}
```

Файл Base.h

```
#ifndef BASE_H
#define BASE_H
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class Base{
protected:
        string name;
        Base* head;
        int status;
public:
        Base(Base* head, string name="");
        void SetName(string name); //метод определения имени объекта
        string GetName(); //метод получения имени объекта
        void PrintTree(string space); //метод вывода наименований объектов в
дереве иерархии
        void SetHead(Base* head); //метод переопределения головного объекта
для текущего в дереве иерархии
        Base* GetHead(); //метод получения указателя на головной объект
текущего объекта
        vector<Base*>children; //массив указателей на объекты, подчиненные к
текущему объекту в дереве иерархии
        Base* FindPtr(string name, Base* ptr=nullptr); //- метод поиска
```

```
объекта на дереве иерархии по имени
        void SetStatus(); //метод установки готовности объекта
        void ShowReadiness(string space);
        void SetChildStatus(int status);
};
#endif
```

Файл Class2.h

```
#ifndef _CLASS2_H
#define _CLASS2_H
#include "Base.h"
class Class2:public Base {
public:
        Class2(Base* ptr, string name=""):Base(ptr, name){};
};
#endif
```

Файл Class3.h

```
#ifndef _CLASS3_H
#define _CLASS3_H
#include "Base.h"
class Class3:public Base {
public:
        Class3(Base* ptr, string name=""):Base(ptr, name){};
};
#endif
```

Файл Class4.h

```
#ifndef _CLASS4_H
#define _CLASS4_H
#include "Base.h"
class Class4:public Base {
public:
        Class4(Base* ptr, string name=""):Base(ptr, name){};
};
#endif
```

Файл Class5.h

```
#ifndef _CLASS5_H
#define _CLASS5_H
#include "Base.h"
class Class5:public Base {
public:
        Class5(Base* ptr, string name=""):Base(ptr, name){};
};
#endif
```

Файл Class6.h

```
#ifndef _CLASS6_H
#define _CLASS6_H
#include "Base.h"
class Class6:public Base {
public:
        Class6(Base* ptr, string name=""):Base(ptr, name){};
};
#endif
```

Файл main.cpp

```
#include "App.h"
int main(){
        App app(nullptr);
        app.BuildTree();
        return app.exec();
}
```

Тестирование

Результат тестирования программы представлен в следующей таблице.

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
app_root app_root object_01 3 app_root object_02 2 object_02 object_04 3 object_02 object_05 5 object_01 object_07 2 endtree app_root 1 object_07 3 object_01 1 object_02 -2 object_04 1	Object tree app_root object_01 object_07 object_02 object_04 object_05 The tree of objects and their readiness app_root is ready object_01 is ready object_07 is not ready object_02 is ready object_04 is ready object_05 is not ready	Object tree app_root object_01 object_07 object_02 object_04 object_05 The tree of objects and their readiness app_root is ready object_01 is ready object_07 is not ready object_02 is ready object_04 is ready object_05 is not ready

ЗАКЛЮЧЕНИЕ

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).