Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Первоначальная сборка системы (дерева иерархии объектов, модели системы) осуществляется исходя из входных данных. Данные вводятся построчно. Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. 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).

Поиск головного объекта выполняется от последнего созданного объекта. Первоначально последним созданным объектом считается корневой объект. Если для головного объекта обнаруживается дубляж имени в непосредственно подчиненных объектах, то объект не создается. Если обнаруживается дубляж имени на дереве иерархии объектов, то объект не создается. Если номер класса объекта задан некорректно, то объект не создается.

Вывод иерархического дерева объектов на консоль.

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных моделях систем 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 ветке дереве иерархии от текущего по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на искомой ветке дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод поиска объекта на дереве иерархии по имени (метод возвращает указатель на найденный объект или nullptr). Передается один параметр строкового типа, содержит наименование искомого объекта. Если на дерева иерархии наименование объекта не уникально или отсутствует, то возвращает nullptr;
- метод вывода иерархии объектов (дерева или ветки) от текущего объекта (допускается использовать один целочисленный параметр со значением по-умолчанию);
- метод вывода иерархии объектов (дерева или ветки) и отметок их готовности от текущего объекта (допускается использовать один целочисленный параметр со значением по-умолчанию);
- метод установки готовности объекта, в качестве параметра передается переменная целого типа, содержит номер состояния.

Устаревший метод вывода из задачи КВ_1 убрать.

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 корневого 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
```

1.1 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»
```

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
```

1.2 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
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект ob_cl_app класса cl_app предназначен для для построения объектов;
- объект потока ввода cin;
- объект потока ввода cout;
- объекты классов cl_2, cl_3, cl_4, cl_5, cl_6 в количестве, определяемом пользователем.

Класс cl_base:

- свойства/поля:
    - поле поле имя объекта:
        - наименование — name;
        - тип — string;
        - модификатор доступа — private;

Класс cl_app:

Собственные свойства и методы отсутствуют.

Класс cl_2:

- функционал:
    - метод cl_2 — создание объекта элемента дерева.

Класс cl_3:

- функционал:
    - метод cl_3 — создание объекта элемента дерева.

Класс cl_4:

- функционал:
    - метод cl_4 — создание объекта элемента дерева.

Класс cl_5:

- функционал:
    - метод cl_5 — создание объекта элемента дерева.

Класс cl_6:

- функционал:
    - метод cl_6 — создание объекта элемента дерева.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base			Родительский класс	
		cl_app	public		2
		cl_2	public		3
		cl_3	public		4
		cl_4	public		5
		cl_5	public		6
		cl_6	public		7
2	cl_app				
3	cl_2				
4	cl_3				
5	cl_4				
6	cl_5				
7	cl_6				

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.1 Алгоритм конструктора класса cl_2

Функционал: Создание объекта элемента дерева.

Параметры: указатель на объект cl_base* parent; string name.

Алгоритм конструктора представлен в таблице 2.

Таблица 2 – Алгоритм конструктора класса cl_2

№	Предикат	Действия	№ перехода
1			∅

3.2 Алгоритм конструктора класса cl_3

Функционал: Создание объекта элемента дерева.

Параметры: указатель на объект cl_base* parent; string name.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса cl_3

№	Предикат	Действия	№ перехода
1			∅

3.3 Алгоритм конструктора класса cl_4

Функционал: Создание объекта элемента дерева.

Параметры: указатель на объект cl_base* parent; string name.

Алгоритм конструктора представлен в таблице 4.

Таблица 4 – Алгоритм конструктора класса cl_4

№	Предикат	Действия	№ перехода
1			∅

3.4 Алгоритм конструктора класса cl_5

Функционал: Создание объекта элемента дерева.

Параметры: указатель на объект cl_base* parent; string name.

Алгоритм конструктора представлен в таблице 5.

Таблица 5 – Алгоритм конструктора класса cl_5

№	Предикат	Действия	№ перехода
1			∅

3.5 Алгоритм конструктора класса cl_6

Функционал: Создание объекта элемента дерева.

Параметры: указатель на объект cl_base* parent; string name.

Алгоритм конструктора представлен в таблице 6.

Таблица 6 – Алгоритм конструктора класса cl_6

№	Предикат	Действия	№ перехода
1			∅

3.6 Алгоритм функции main

Функционал: основная программа.

Параметры: нет.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 7.

Таблица 7 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_cl_app класса cl_app с подачей nullptr	2
2		вызов метода build у ob_cl_app	3
3		возвращение результата выполлнения метода start_app объекта ob_cl_app	∅

3.7 Алгоритм метода printname класса cl_base

Функционал: вывод иерархии объектов от текущего объекта.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода printname класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация указаеля selectedobject=this	2
2		вывод переноса строки	3
3	selectedobject есть или нет?	вывод в консоль строки " "	4
		вывод имени объекта указателя selectedobject	3
4		устаавление роителя в качесве указаеля selectedobject	5
5		объявление указатедля child на объек класса cl_base	6
6	указатель child проходится по вектору children	вызов метода printame у объекта указателя child	6

№	Предикат	Действия	№ перехода
			∅

3.8 Алгоритм метода godowntotheobject класса cl_base

Функционал: поиск объекта среди родиелей.

Параметры: нет.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода godowntotheobject класса cl_base

№	Предикат	Действия	№ перехода
1	name совпадает с именем родителя	возвращаемое значение this	∅
	объект не имеетт родиеля	возвращение результата выполнения метода gouptotheobject	∅
		возвращение результата выполнения метода godowntotheobjects объекта указателя parent	∅

3.9 Алгоритм метода gouptotheobjects класса cl_base

Функционал: поиск объекта среди дочерних объектов.

Параметры: нет.

Возвращаемое значение: cl_base*.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода gouptotheobjects класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация переменной childindex=0	2
2	childindex меньше значения		3

№	Предикат	Действия	№ перехода
	размера вектора children		
		возвращение nullptr	∅
3	name совпадает с именем children[childindex]	возврат children[childindex]	∅
	у объекта есть дочерние объекта	инициализация указателя cl_base* resultfromup результатом выполнения метода gouptotheobject объекта children[childindex]	4
			5
4	указатель resultfromup не пустой	возвращение указателя resultfromup	∅
			5
5		childindex+1	2

3.10 Алгоритм метода setreadycode класса cl_base

Функционал: усттановка сосояния готовности объекта.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 11 – Алгоритм метода setreadycode класса cl_base

№	Предикат	Действия	№ перехода
1	у объекта нет родителя или объект не готов	присвоение полю readycode значение code	2
			2
2	code==0	присвоение полю readycode значение code	3
			∅
3		инициализация переменной childindex=0	4
4	childindex меньше значения	вызов метода setreadycode у children[childindex] с	5

№	Предикат	Действия	№ перехода
	размера вектора children	подачей 0	
			∅
5		childindex+1	4

3.11 Алгоритм метода printstatus класса cl_base

Функционал: вывод имен дочерних обектов с указаелем их состояния гоовности.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода printstatus класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация указателя selectedobject=this	2
2		вывод переноса строки	3
3	существует родитель у selectedobject	вывод " "	4
		вывод имени объекта указателя this	5
4		установление родителя в качестве указателя selectedobject	3
5	объект указателя this готов	вывод " is ready"	6
		вывод " is not ready"	6
6		объявление указателя child на объект класса cl_base	7
7	указатель child проходится по вектору children	вызов метода printreadystatus у объекта указателя child	6
			∅

3.12 Алгоритм метода build класса cl_app

Функционал: посттроеие иерархии объектов.

Параметры: нет.

Возвращаемое значение: none.

Алгоритм метода представлен в таблице 13.

Таблица 13 – Алгоритм метода build класса cl_app

№	Предикат	Действия	№ перехода
1		инициализация указателя на объект класса cl_base selectedobject значением указателя this	2
2		объявление переменных типа string parentname и childname	3
3		объявление переменной classnum типа int	4
4		ввод значения переменной parentname	5
5		вызов метода setname с параметром parentname	6
6		ввод значения переменной parentname	7
7	parentname=="endtree"		11
		ввод значений childname и classnum	8
8	значение classnum не на интервале [2, 6]	присвоение указателю selectedobject результат выполнения метода godowntotheobject с параметром parentname	9
			6
9	указатель selectedobject не пустой и нет объектов с именем childname у подчиненных объектов		10
			6
10	classnum==2	создание динамического объекта класса cl_2 с вызовом конструктора с параметрами	6

№	Предикат	Действия	№ перехода
		selectedobject и childname	
	classnum==3	создание динамического объекта класса cl_3 с вызовом конструктора с параметрами selectedobject и childname	6
	classnum==4	создание динамического объекта класса cl_4 с вызовом конструктора с параметрами selectedobject и childname	6
	classnum==5	создание динамического объекта класса cl_5 с вызовом конструктора с параметрами selectedobject и childname	6
	classnum==6	создание динамического объекта класса cl_6 с вызовом конструктора с параметрами selectedobject и childname	6
			6
11		объявление переменной readycode типа int	12
12	ввод не закончился	ввод значения для переменной readycode	13
			∅
13		вызов метода setreadycode у результата выполнения метода godowntotheobject	12

3.13 Алгоритм метода start_app класса cl_app

Функционал: изменение состояния готовности и вывод в консольиерархического дерева объектов и отметок их готовности..

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода start_app класса cl_app

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2
2		вызов метода printnames с параметром 0	3
3		вывод переноса строки + "The tre of objects and their readinse\n"	4
4		вызов метода printreadystatus с параметром 1	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-12.

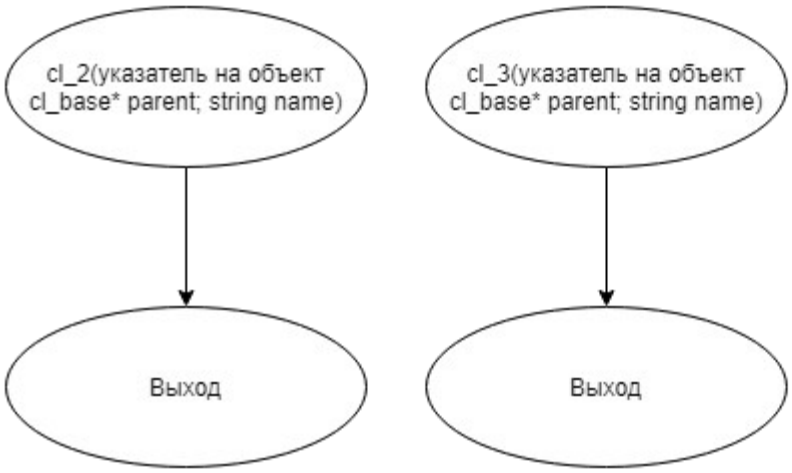

Рисунок 1 – Блок-схема алгоритма

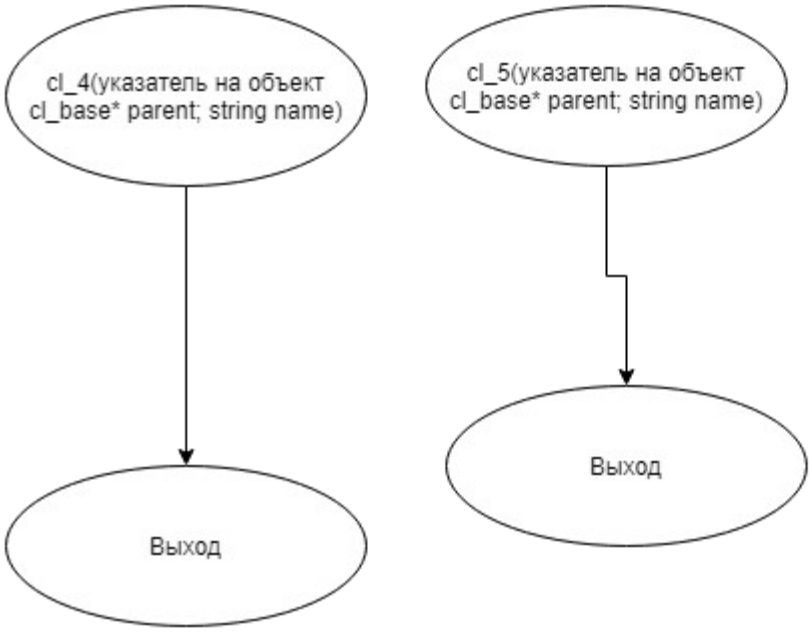

Рисунок 2 – Блок-схема алгоритма

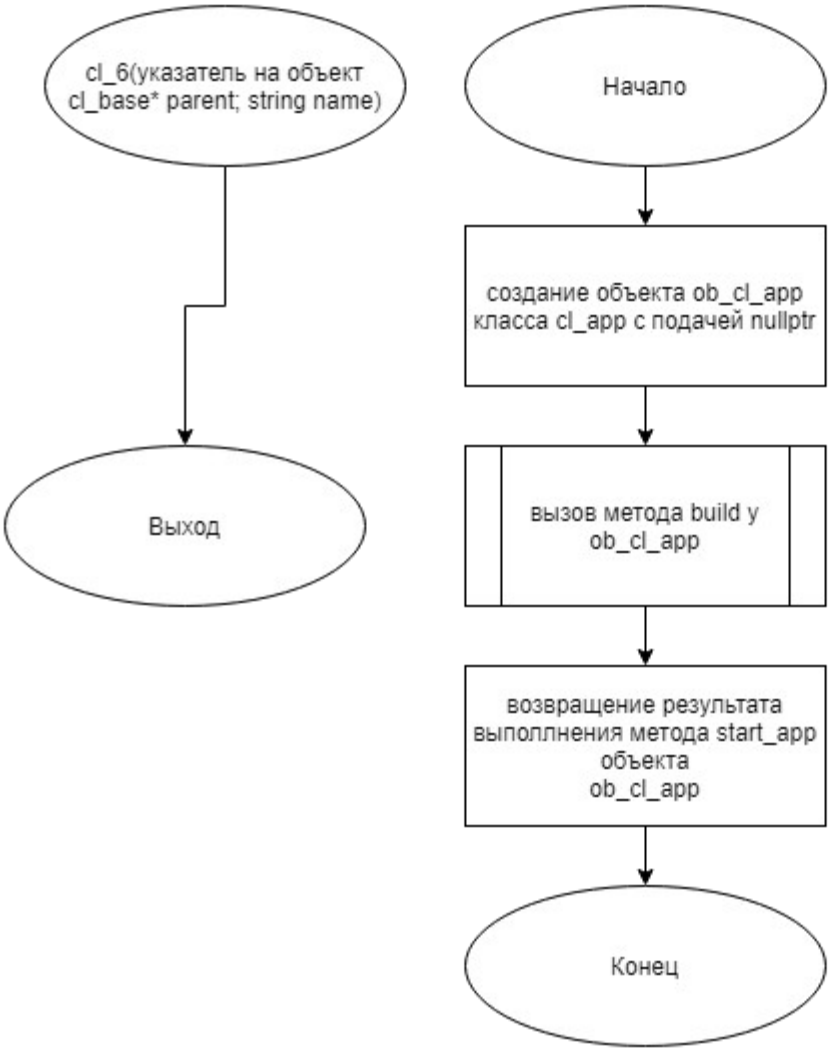

Рисунок 3 – Блок-схема алгоритма

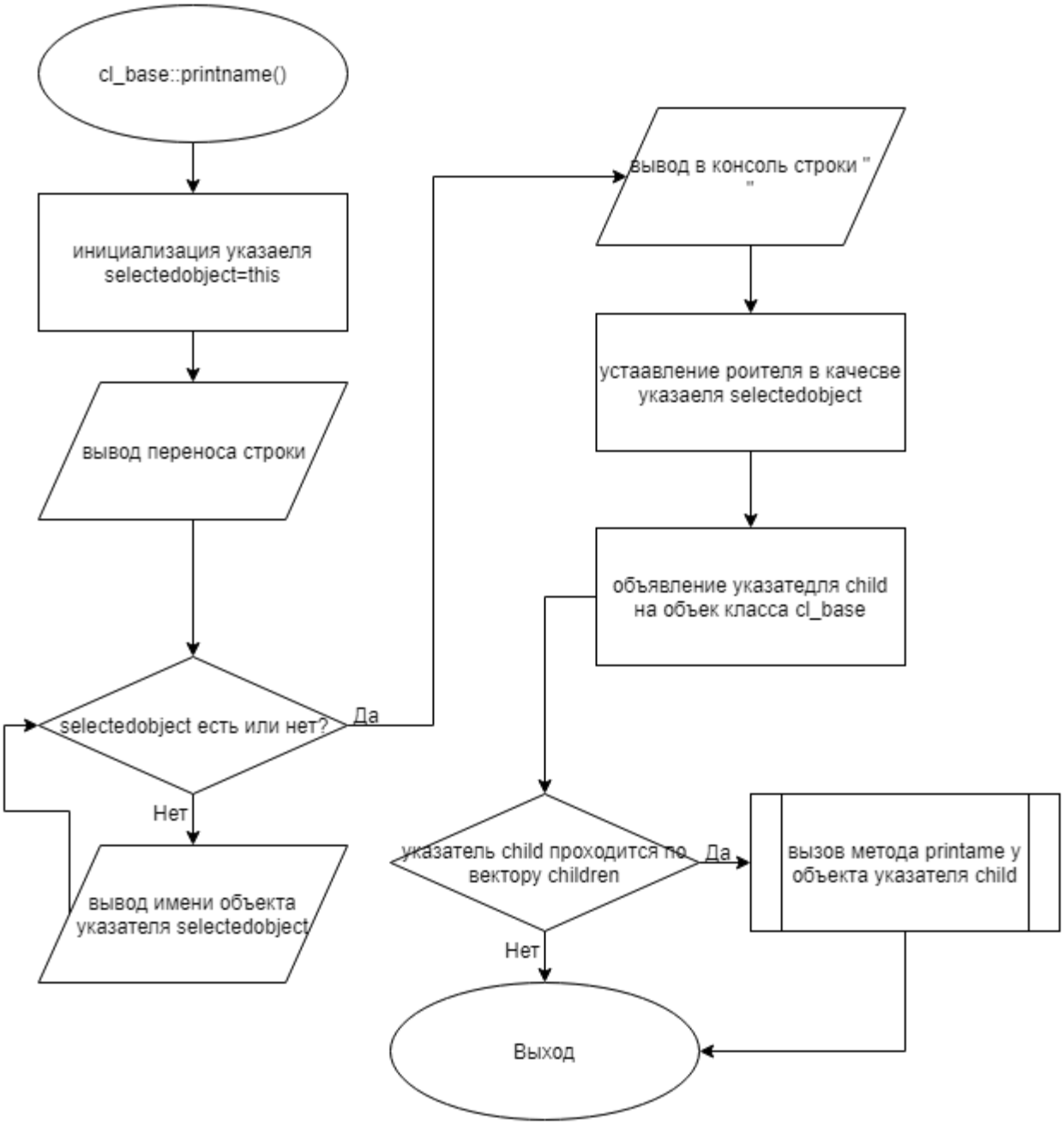

Рисунок 4 – Блок-схема алгоритма

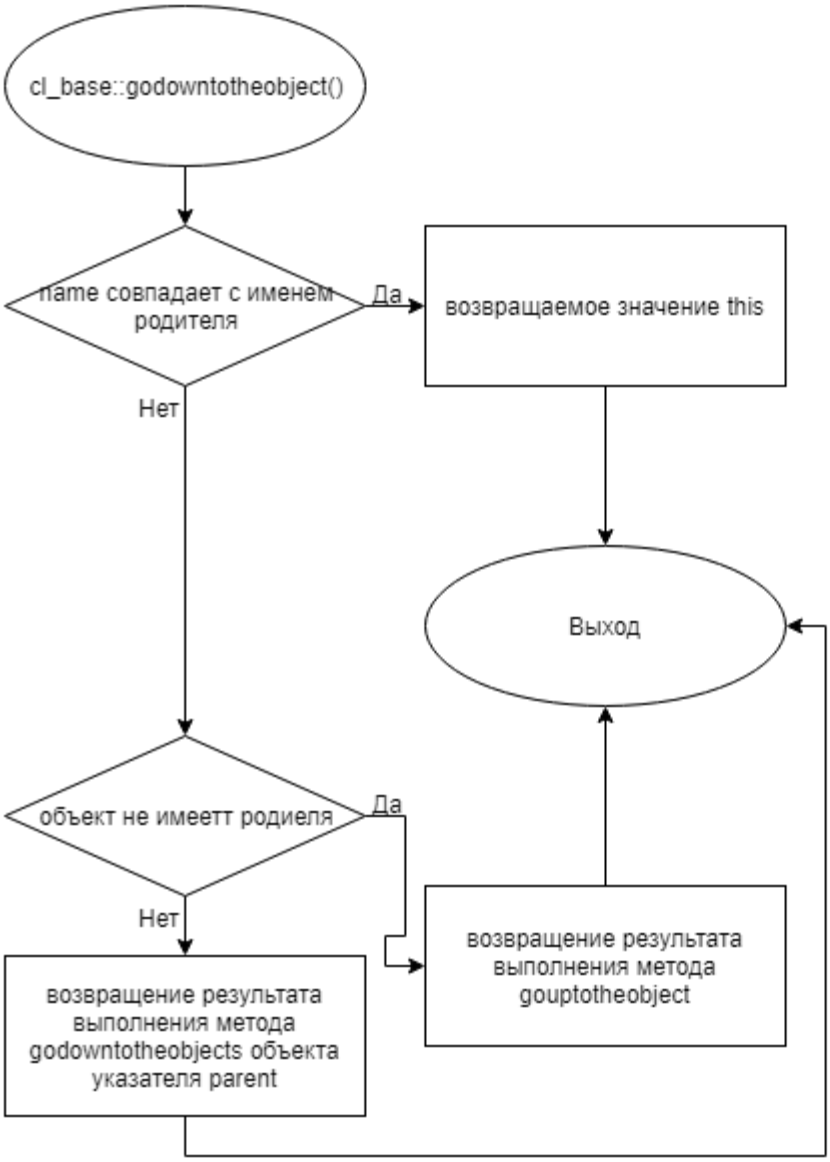

Рисунок 5 – Блок-схема алгоритма

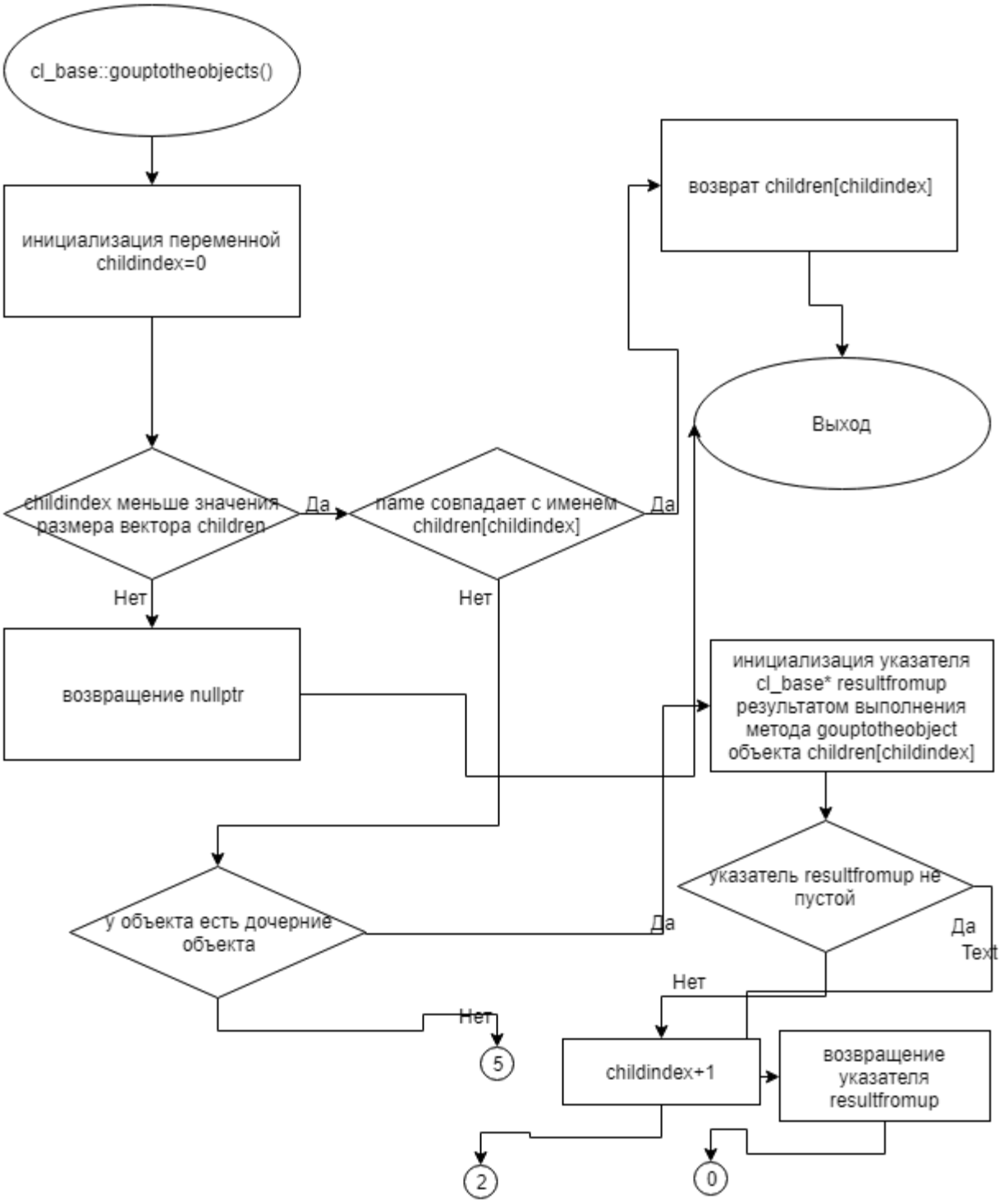

Рисунок 6 – Блок-схема алгоритма

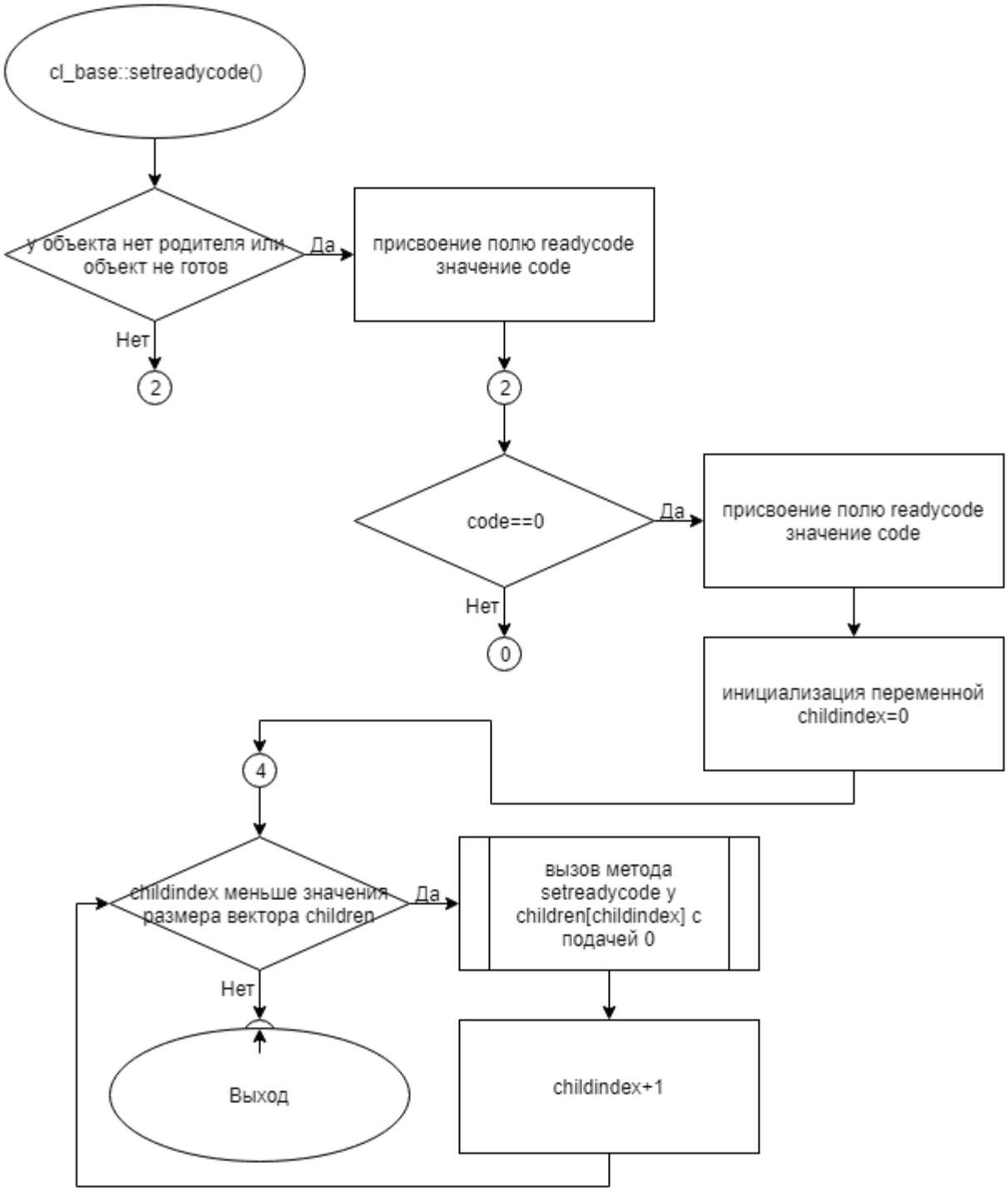

Рисунок 7 – Блок-схема алгоритма

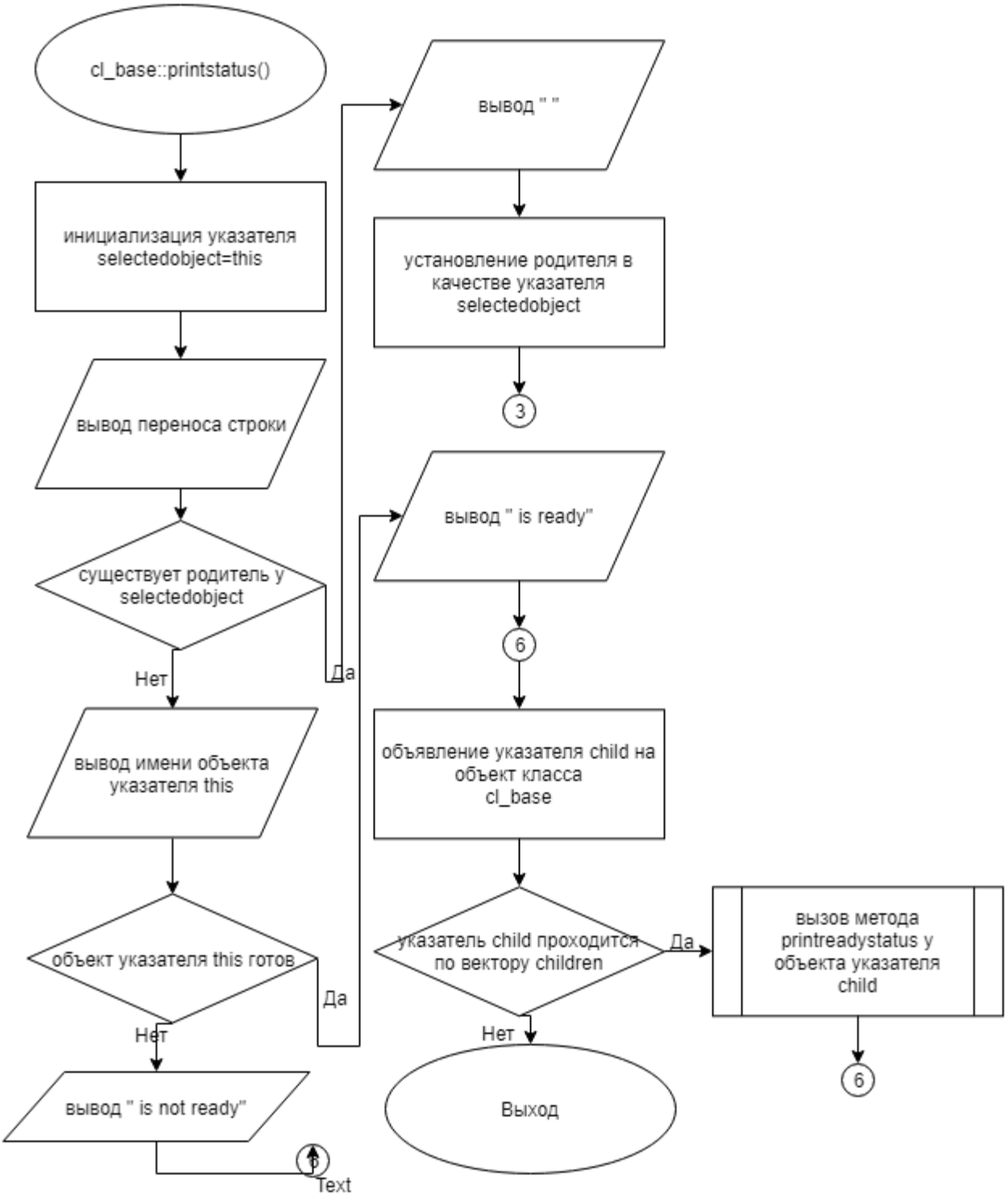

Рисунок 8 – Блок-схема алгоритма

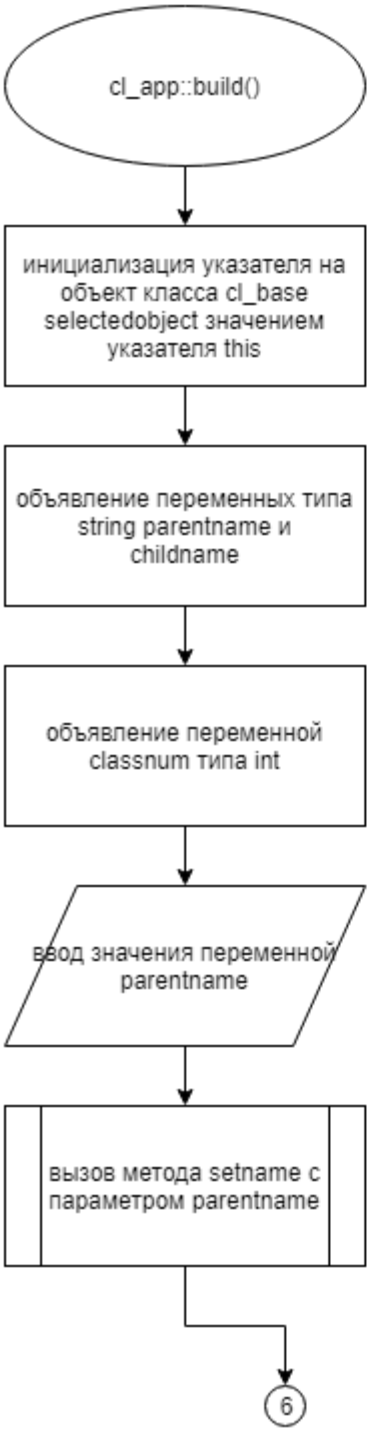

Рисунок 9 – Блок-схема алгоритма

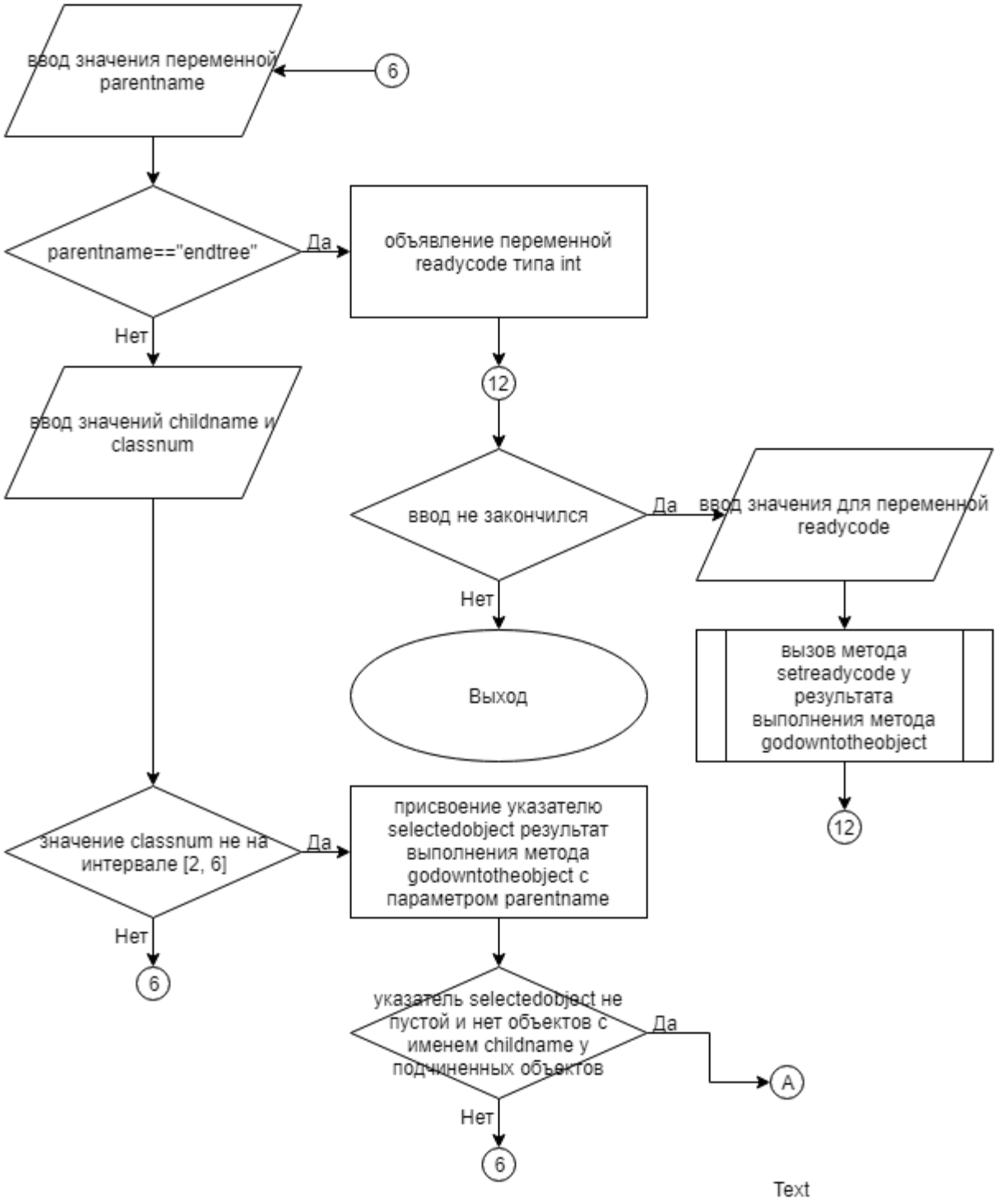

Рисунок 10 – Блок-схема алгоритма

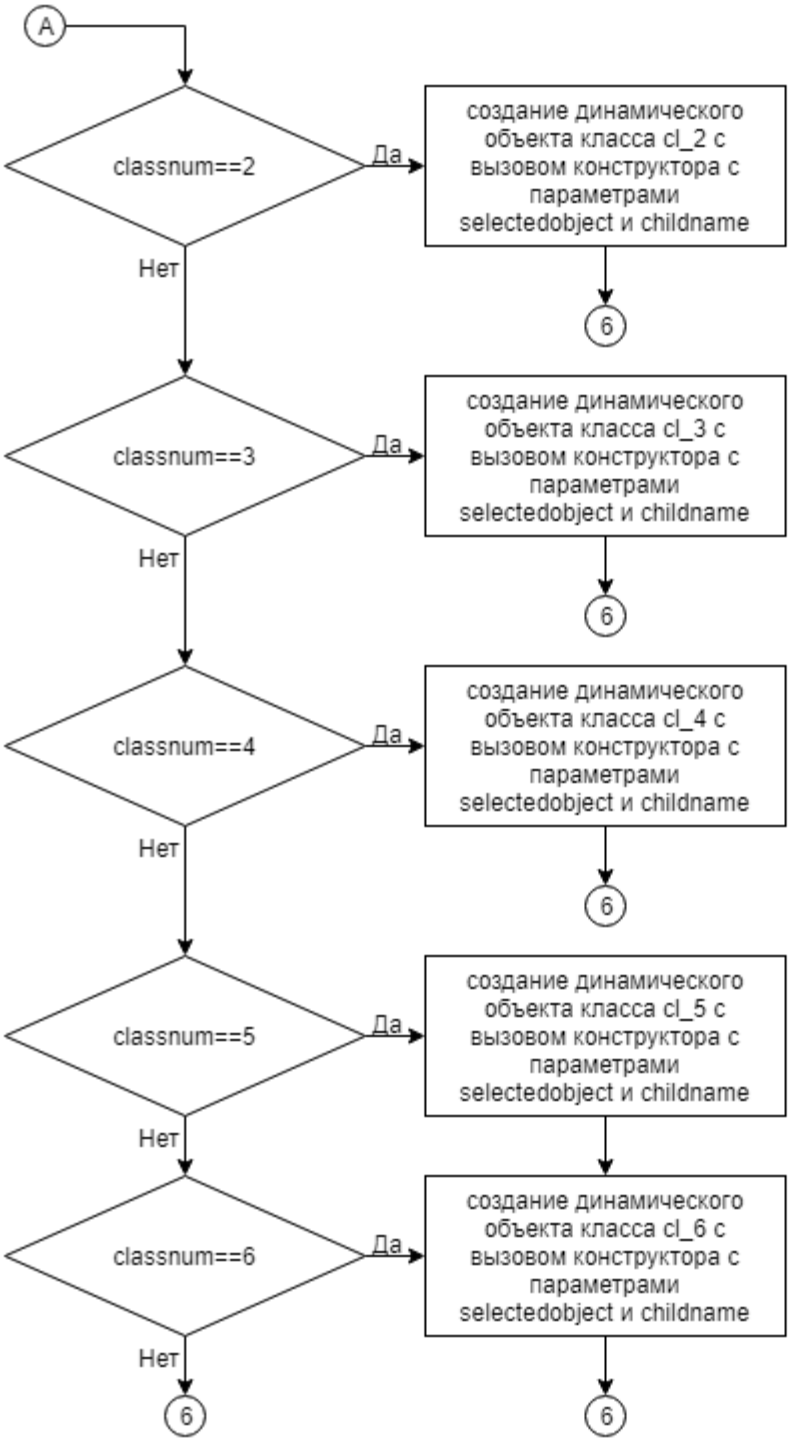

Рисунок 11 – Блок-схема алгоритма

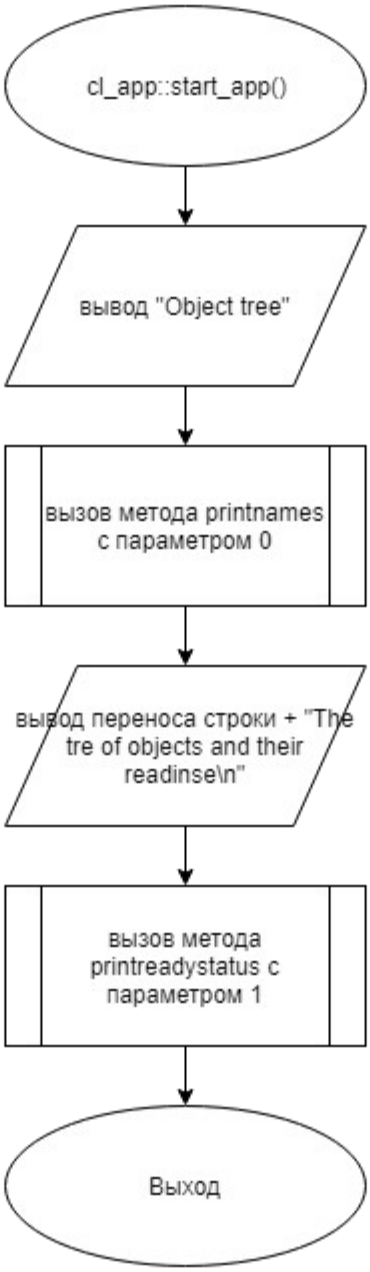

Рисунок 12 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_2.cpp

Листинг 1 – cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(cl_base* parent, string name):cl_base(parent, name){}
```

5.2 Файл cl_2.h

Листинг 2 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "cl_base.h"
class cl_2:public cl_base
{
public:
   cl_2(cl_base* parent, string s_name);
};
#endif
```

5.3 Файл cl_3.cpp

Листинг 3 – cl_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* parent, string name):cl_base(parent, name){}
```

5.4 Файл cl_3.h

Листинг 4 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "cl_base.h"
class cl_3:public cl_base
{
public:
   cl_3(cl_base* parent, string s_name);
};
#endif
```

5.5 Файл cl_4.cpp

Листинг 5 – cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* parent, string name):cl_base(parent, name){}
```

5.6 Файл cl_4.h

Листинг 6 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include "cl_base.h"
class cl_4:public cl_base
{
public:
   cl_4(cl_base* parent, string s_name);
};
#endif
```

5.7 Файл cl_5.cpp

Листинг 7 – cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* parent, string name):cl_base(parent, name){}
```

5.8 Файл cl_5.h

Листинг 8 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include "cl_base.h"
class cl_5:public cl_base
{
public:
   cl_5(cl_base* parent, string s_name);
};
#endif
```

5.9 Файл cl_6.cpp

Листинг 9 – cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* parent, string name):cl_base(parent, name){}
```

5.10 Файл cl_6.h

Листинг 10 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "cl_base.h"
class cl_6:public cl_base
```

```
{
public:
   cl_6(cl_base* parent, string s_name);
};
#endif
```

5.11 Файл cl_app.cpp

Листинг 11 – cl_app.cpp

```
#include "cl_app.h"
#include <iostream>
#include <string>
using namespace std;
cl_app::cl_app(cl_base* parent):cl_base(parent){}
void cl_app::build()
{
   //построение иерархии объектов
   cl_base* selectedobject=this;//инициализация
   string parentname, childname;
   int classnum;
   cin>>parentname;
   setname(parentname);
   while (true)
   {
      cin>>parentname;
      if (parentname=="endtree")
      {
         break;
      }
      cin>>childname>>classnum;
      if (classnum<2||classnum>6)
      {
         continue;
      }
      selectedobject=godowntotheobject(parentname);
      if          (selectedobject!=nullptr          &&          selectedobject-
>gouptotheobject(childname)==nullptr)
      {
         switch(classnum)
         {
            //создание динамического объекта ...
            case 2:
               new cl_2(selectedobject, childname);
               break;
            case 3:
               new cl_3(selectedobject, childname);
               break;
            case 4:
               new cl_4(selectedobject, childname);
```

```
                  break;
               case 5:
                  new cl_5(selectedobject, childname);
                  break;
               case 6:
                  new cl_6(selectedobject, childname);
                  break;
               default:
                  break;
            }
         }
      }
   }
   int cl_app::start_app()
   {
      //изменение сосояния готовности и вывод в консоль
      //иерархии дерева объектов и отмеок их гоовности
      cout<<"Object tree";
      printnames();
      int readycode;
      string parentname;
      while (cin>>parentname>>readycode)
      {
         cl_base* searchobject=godowntotheobject(parentname);
         if (searchobject!=nullptr)
         {
            searchobject->setreadycode(readycode);
         }
      }
      cout<<endl<<"The tree of objects and their readiness";
      printreadystatus();
      return 0;
   }
```

5.12 Файл cl_app.cpp.bak

Листинг 12 – cl_app.cpp.bak

```
#include "cl_app.h"
#include <iostream>
#include <string>
using namespace std;
cl_app::cl_app(cl_base* parent):cl_base(parent){}
void cl_app::build()
{
   //построение иерархии объектов
   cl_base* selectedobject=this; //инициализация
   string parentname, childname;
   int classnum;
   cin>>parentname;
   setname(parentname);
   while (true)
   {
      cin>>parentname;
      if (parentname=="endtree")
      {
         break;
      }
      cin>>childname>>classnum;
      if (classnum<2||classnum>6)
      {
         continue;
      }
      selectedobject=godowntotheobject(parentname);
      if         (selectedobject!=nullptr         &&         selectedobject-
>gouptotheobject(childname)==nullptr)
      {
         switch(classnum)
         {
            //создание динамического объекта ...
            case 2:
               new cl_2(selectedobject, childname);
               break;
            case 3:
               new cl_3(selectedobject, childname);
               break;
            case 4:
               new cl_4(selectedobject, childname);
               break;
            case 5:
               new cl_5(selectedobject, childname);
               break;
            case 6:
               new cl_6(selectedobject, childname);
               break;
            default:
               break;
         }
```

```
      }
   }
}
int cl_app::start_app()
{
   //изменение состония готовности и вывод в консоль
   // иерархии дерева объектов и отметок их готовности
   cout<<"Object tree";
   printnames();
   int readycode;
   string parentname;
   while (cin>>parentname>>readycode)
   {
      cl_base* searchobject=godowntotheobject(parentname);
      if (searchobject!=nullptr)
      {
         searchobject->setreadycode(readycode);
      }
   }
   cout<<endl<<"The tree of objects and their readiness";
   printreadystatus();
   return 0;
}
```

5.13 Файл cl_app.h

Листинг 13 – cl_app.h

```
#ifndef __CL_APP__H
#define __CL_APP__H
#include "cl_base.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"
class cl_app: public cl_base
{
public:
   cl_app(cl_base* parent);
   void build();
   int start_app();
};
#endif
```

5.14 Файл cl_base.cpp

Листинг 14 – cl_base.cpp

```
#include "cl_base.h"
cl_base::cl_base(cl_base* parent, string name):parent(parent), name(name)
{
   if (parent!=nullptr)
   {
      parent->children.push_back(this);
   }
}
cl_base::~cl_base()
{
   for (int childindex=0; childindex<children.size(); childindex++)
   {
      delete children[childindex];
   }
}
string cl_base::getname()
{
   return name;
}
cl_base* cl_base::getparent()
{
   return parent;
}
bool cl_base::setname(string newname)
{
   if (getparent()!=nullptr && getparent()->getchild(newname))
   {
      return false;
   }
   name=newname;
   return true;
}
void cl_base::printnames()
{
   //вывод иерархии объектов от екущего объекта
   cl_base* selectedobject=this;
   cout<<endl;
   while (selectedobject->getparent())//проверка родителя
   {
      cout<<"    ";
      selectedobject=selectedobject->getparent();
   }
   cout<<this->getname();
   for (cl_base* child:this->children)
   {
      child->printnames();
   }
}
cl_base* cl_base::getchild(string name)
{
```

```
   for (cl_base* child:children)
   {
      if (child->name==name)
      {
         return child;
      }
   }
   return nullptr;
}
cl_base* cl_base::gouptotheobject(string name)
{
   //поиск объекта среди дочерних объеков
   for (int childindex=0; childindex<children.size(); childindex++)
   {
      if (name==children[childindex]->getname())
      {
         return children[childindex];
      }
      if (children[childindex]->children.size()>0)
      {
         cl_base* resultfromup=children[childindex]->gouptotheobject(name);
         if (resultfromup!=nullptr)
         {
            return resultfromup;
         }
      }
   }
   return nullptr;
}
cl_base* cl_base::godowntotheobject(string name)
{
   //поиск объекта среди родиелей
   if (name==getname())//совподает с именем родителя
   {
      return this;
   }
   if (parent==nullptr)//не имеет родителя?
   {
      return gouptotheobject(name);
   }
   return parent->godowntotheobject(name);
}
void cl_base::setreadycode(int code)
{
   //установка состония готовности объекта
   if (parent==nullptr||parent->readycode!=0)//не готов?
   {
      this->readycode=code;
   }
   if (!code)
   {
      this->readycode=code;
      for (int childindex=0; childindex<children.size(); childindex++)
      {
```

```
            children[childindex]->setreadycode(code);
         }
      }
   }
   void cl_base::printreadystatus()
   {
      //вывод имен дочерних объектов с указателдем
      //их состояния готовности
      cl_base* selectedobject=this;
      cout<<endl;
      while(selectedobject->getparent())//есть ли родитель?
      {
         cout<<"    ";
         selectedobject=selectedobject->getparent();
      }
      cout<<this->getname();
      if (this->readycode!=0)
      {
         cout<<" is ready";
      }
      else
      {
         cout<<" is not ready";
      }
      for (cl_base* child: this->children)
      {
         child->printreadystatus();
      }
   }
```

5.15 Файл cl_base.h

Листинг 15 – cl_base.h

```
   #ifndef __CL_BASE__H
   #define __CL_BASE__H
   #include <vector>
   #include <string>
   #include <iostream>
   using namespace std;
   class cl_base
   {
   private:
      string name;
      cl_base* parent=nullptr;
      vector <cl_base*> children;
      int readycode=0;
   public:
      cl_base(cl_base* parent, string name="Object_Root");
      ~cl_base();
```

```
        bool setname(string name);
        cl_base* getparent();
        string getname();
        void printnames();
        cl_base* getchild(string name);
        cl_base* gouptotheobject(string name);
        cl_base* godowntotheobject(string name);
        void setreadycode(int code);
        void printreadystatus();
    };
    #endif
```

5.16 Файл main.cpp

Листинг 16 – main.cpp

```
    #include <stdlib.h>
    #include <stdio.h>
    #include "cl_app.h"
    int main()
    {
        cl_app ob_cl_app(nullptr);
        ob_cl_app.build();
        return ob_cl_app.start_app();
    }
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 15.

Таблица 15 – Результат тестирования программы

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
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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).