Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

Расширить функциональность базового класса:

- метод переопределения головного объекта для текущего в дереве иерархии. Метод должен иметь один параметр, указатель на объект базового класса, содержащий указатель на новый головной объект. Переопределение головного объект для корневого объекта недопустимо. Недопустимо создать второй корневой объект. Недопустимо при переопределении, чтобы у нового головного появились два подчиненных объекта с одинаковым наименованием. Новый головной объект не должен принадлежать к объектам из ветки текущего. Если переопределение выполнено, метод возвращает значение «истина», иначе «ложь»;
- метод удаления подчиненного объекта по наименованию. Если объект не найден, то метод завершает работу. Один параметр строкового типа, содержит наименование удаляемого подчиненного объекта;
- метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задаться в следующем виде:
    - / - корневой объект;
    - //«имя объекта» - поиск объекта по уникальной имени от корневого (для однозначности уникальность требуется в рамках дерева);
    - . - текущий объект;
    - .«имя объекта» - поиск объекта по уникальной имени от текущего (для однозначности уникальность требуется в рамках ветви дерева от

текущего объекта);

- «имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;
- /«имя объекта 1»[/«имя объекта 2»] . . . - абсолютная координата от корневого объекта.

Примеры координат:

```
/
//ob_3
.
.ob_2
ob_2/ob_3
/ob_1/ob_2/ob_3
```

Если координата - пустая строка или объект не найден или определяется неоднозначно (дубляж имен на ветке, на дереве), тогда вернуть нулевой указатель.

Наименование объекта не содержит символы «.» и «/».

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие. В строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему. При построении дерева уникальность наименования относительно множества непосредственно подчиненных объектов для любого головного объекта необходимо соблюдать. Если это требование исходя из входных данных нарушается, то соответствующий подчиненный объект не создается.

Добавить проверку допустимости исходной сборки. Собрать дерево невозможно, если по заданной координате головной объект не найден (например, ошибка в наименовании или еще не расположен на дереве объектов). Если номер класса объекта задан некорректно, то объект не создается.

Собранная система отрабатывает следующие команды:

- SET «координата» – устанавливает текущий объект;
- FIND «координата» – находит объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» задает новый головной объект;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим. При вводе данных в названии команд ошибок нет. Если при переопределении головного объекта нарушается уникальность наименований подчиненных объектов для нового головного, переопределение не производится.

1.1 Описание входных данных

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие. В строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

После ввода состава дерева иерархии построчно вводятся команды:

- SET «координата» – установить текущий объект;
- FIND «координата» – найти объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» соответствует новому головному объекту;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершить функционирование системы (выполнение программы).

Команды SET, FIND, MOVE и DELETE вводятся произвольное число раз.

Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов:

```
rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4 object_7 3
endtree
FIND object_2/object_4
SET /object_2
FIND //object_7
FIND object_4/object_7
FIND .
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева как в работе версия №2.

При ошибке определения головного объекта, прекратить сборку, вывести иерархию уже построенного фрагмента дерева, со следующей строки сообщение:

```
The head object «координата головного объекта» is not found
```

и прекратить работу программы с кодом возврата 1.

Если при построении при попытке создания объекта обнаружен дубляж, то вывести:

«координата головного объекта» Dubbing the names of subordinate objects

Если дерево построено, то далее построчно вводятся команды.

Для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

The object was not found at the specified coordinate: «искомая координата объекта»

Для команд FIND вывести:

«искомая координата объекта» Object name: «наименование объекта»

Если объект не найден, то:

«искомая координата объекта» Object is not found

Для команд MOVE вывести:

New head object: «наименование нового головного объекта»

Если головной объект не найден, то:

«искомая координата объекта» Head object is not found

Если переопределить головной объект не удалось, то:

«искомая координата объекта» Redefining the head object failed

Если у нового головного объекта уже есть подчиненный с таким же именем, то вывести:

«искомая координата объекта» Dubbing the names of subordinate objects

При попытке переподчинения головного объекта к объекту на ветке, вывести:

«координата нового головного объекта» Redefining the head object failed

Для команды DELETE:

Если подчиненный объект удален, то вывести:

The object «абсолютный путь удаленного объекта» has been deleted

Если объект не найден, то ничего не выводить.

После команды END с новой строки вывести:

```
Current object hierarchy tree
```

Со следующей строки вывести текущую иерархию дерева.

Пример вывода иерархии дерева объектов:

```
Object tree
rootela
    object_1
        object_7
    object_2
        object_4
            object_7
        object_5
        object_3
    object_3
object_2/object_4     Object name: object_4
Object is set: object_2
//object_7     Object is not found
object_4/object_7     Object name: object_7
.     Object name: object_2
.object_7     Object name: object_7
object_4/object_7     Object name: object_7
.object_7     Redefining the head object failed
Object is set: object_7
//object_1     Dubbing the names of subordinate objects
New head object: object_3
Current object hierarchy tree
rootela
    object_1
        object_7
    object_2
        object_4
        object_5
        object_3
    object_3
        object_7
```

2 МЕТОД РЕШЕНИЯ

Класс baseclass:

- функционал:
    - метод printnames — вывод иерархии объектов;
    - метод printreadystatus — вывод иерархии объектов с указанием состояния готовности каждого объекта;
    - метод findobjectbycoordinate — поиск объекта в иерархии по координате;
    - метод removechild — вычеркивание из списка одного объекта из дочерних;
    - метод changenewparent — изменение головного объекта на другой.

Класс app:

- функционал:
    - метод build — построение иерархии объектов;
    - метод start_app — вывод иерархии объектов в консоль.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	baseclass				
		app	public		2
2	app				

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 функции main

Функционал: основная функция программы.

Параметры: none.

Возвращаемое значение: int - код ошибки.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_app класса app с параметром nullptr	2
2		вызов метода build объекта ob_app	3
3		возврат результата работы метода start_app() для объекта ob_app	∅

3.2 Алгоритм метода printnames класса baseclass

Функционал: вывод иерархии объектов.

Параметры: количество пробелов int countspaces.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода printnames класса baseclass

№	Предикат	Действия	№ перехода
1		вывод в консоль переноса на новую строку, пробелов количестовм countspaces и имени	2

№	Предикат	Действия	№ перехода
		объекта	
2	перебор элементов в children указателем child	вызов у child метода printnames	2
			∅

3.3 Алгоритм метода printreadystatus класса baseclass

Функционал: вывод иерархии объектов с указанием состояния готовности каждого объекта.

Параметры: количество пробелов int countspaces.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода printreadystatus класса baseclass

№	Предикат	Действия	№ перехода
1		вывод в консоль переноса на новую строку, пробелов количеством countspaces и имени объекта	2
2	объект активный	вывод в консоль " is ready"	3
		вывод в консоль " is not ready"	3
3	перебор элементов в children указателем child	вызов у child метода printreadystatus	2
			∅

3.4 Алгоритм метода findobjectbycoordinate класса baseclass

Функционал: поиск объекта в иерархии по координате.

Параметры: string coordinate.

Возвращаемое значение: baseclass*.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода findobjectbycoordinate класса baseclass

№	Предикат	Действия	№ перехода
1	coordinate=="/"	инициализация baseclass* temp_root=this	2
	coordinate==".."	возвращение указателя this	∅
	в начале coordinate стоят два "/"	возвращение значения метода getchild от корневого объекта	∅
	в начале coordinate стоит "."	возвращение значения метода gouptotheobject от текущего объекта	∅
	первый символ не "/"	инициализация cutcoordinate значением координаты без первого объекта	3
	есть только / в начале coordinate	возвращение значения getchild, принимающий coordinate без первого /, от корневого объекта	∅
			∅
2	у temp_root есть родитель	присвоение temp_root родителя temp_root	2
			∅
3	в cutcoordinate нет '/'	возвращение значения getchild с подачей cutcoordinate от определенного дочернего объекта текущего	∅
		возвращение значения findobjectbycoordinate с подачей cutcoordinate от определенного дочернего объекта текущего	∅

3.5 Алгоритм метода removechild класса baseclass

Функционал: вычеркивание из списка одного объекта из дочерних.

Параметры: string name.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода removechild класса baseclass

№	Предикат	Действия	№ перехода
1		инициализация i типа int, равной 0	2
2	i<children.size()		3
			∅
3	объект children[i] имеет имя name	вызов children.erase(children.begin()+i)	∅
		прибавление к i 1	2

3.6 Алгоритм метода changenewparent класса baseclass

Функционал: изменение головного объекта на другой.

Параметры: baseclass* newparent.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода changenewparent класса baseclass

№	Предикат	Действия	№ перехода
1	текущий объект является корневым	возврат false	∅
	указатель newparent является пустым	возврат false	∅
	у объекта указателя newparent есть дочерний объект с таким же именем, как и у текущего	возврат false	∅
	объект указателя newparent принадлежит ветке текущего объекта	возврат false	∅
		вызов у текущего родителя метод removechild	2

№	Предикат	Действия	№ перехода
2		присвоить полю parent значение newparent	3
3		у вектора объекта указателя newparent вызвать метод push_back с подачей указателя на нынешний объект	4
4		возврат true	∅

3.7 Алгоритм метода build класса app

Функционал: построение иерархии объектов.

Параметры: none.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода build класса app

№	Предикат	Действия	№ перехода
1		инициализация указателя baseclass* selectedobject=this	2
2		объявление строк parentname, childname, parentcoordinate	3
3		объявление int classnum	4
4		ввод имени корневого объекта в parentname	5
5		вызов метода setname с подачей parentname	6
6		ввод значения parentcoordinate	7
7	parentcoordinate=="endtree"	возвращение пустой строки	∅
		ввод значений childname и classnum	8
8	значение classnum от 2 до 6		6
		присвоение selectedobject = findobjectbycoordinate(parentcoordinate)	9
9	указатель selectedobject не		10

№	Предикат	Действия	№ перехода
	является пустым		
		возвращение значения parentcoordinate	∅
10	у объекта указателя selectedobject нет объекта с таким же именем, как childname		11
		вывод в консоль вводимой координаты и "Dubbing the names of subordinate objects\n"	6
11	classnum==2	создание нового указателя на объект класса cl_2 с подачей selectedobject и childname	6
	classnum==3	создание нового указателя на объект класса cl_3 с подачей selectedobject и childname	6
	classnum==4	создание нового указателя на объект класса cl_4 с подачей selectedobject и childname	6
	classnum==5	создание нового указателя на объект класса cl_5 с подачей selectedobject и childname	6
	classnum==6	создание нового указателя на объект класса cl_6 с подачей selectedobject и childname	6

3.8 Алгоритм метода start_app класса app

Функционал: вывод иерархии объектов в консоль.

Параметры: failcoordinate.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода start_app класса app

№	Предикат	Действия	№ перехода
1		вывод "Object tree"	2

№	Предикат	Действия	№ перехода
2		вызов метода printnames	3
3		вывод в консоль перенос на новую строку	4
4	строка failcoordinate не пустая	вывод в консоль "The head object {failcoordinate} is not foound"	5
		объявление строк commandcore, secondparametr и coordobject	6
5		возврат 1	∅
6		инициализация baseclass* pointer=this	7
7		объявление baseclass* temp	8
8		ввод значения commandcode	9
9	commandcode=="END"	вывод "Current object hierarchy tree"	10
		ввод значения secondparameter	11
10		вызов метода printnames	23
11	commandcode=="SET"	присвоение temp значение findobjectbycoordinate относительно pointer	12
	commandcode=="FIND"	присвоение temp значение findobjectbycoordinate относительно pointer	14
	commandcode=="MOVE"	присвоение temp значение findobjectbycoordinate относительно pointer	15
	commandcode=="DELETE"	присвоение temp указателя на дочернего объекта с именем secondparameter	16
12	указатель temp не является пустым	pointer=temp	13
		вывод в консоль "The object was not found at specified coordinate: {secondparameter}\n"	8
13		вывод в консоль "Object is set: {имя объекта указателя pointer}\n"	8
14	указатель temp не является пустым	вывод "{secondparameter} Object name: {имя объекта указателя temp}\n"	8

№	Предикат	Действия	№ перехода
		вывод "{secondparameter} Object is not found\n"	8
15	результат выполнения метода changenewparent относительно pointer положительный	вывод в консоль "New head object: {имя объекта указателя temp}\n"	8
	указатель temp не является пустым	вывод в консоль "{secondparametr} Head object is not found\n"	8
	у temp нет дочерних объектов схожих по имени с pointer	вывод в консоль "{secondparametr} Dubbing the names of subordinate objects\n"	8
		вывод в консоль "{secondparametr} Refining the head object failed\n"	8
16	указатель temp не является пустым		17
			8
17	объект temp не является корневым	coordobject="/"	21
		вызов у родителя объекта temp метода removechild с подачей имени объекта temp	18
18		coordobject="/"+{имя объекта temp}	19
19	temp не является дочерним объектом корневого	присвоение temp указателя на родителя относительно temp	20
			21
20		coordobject="/"+{имя объекта temp}+coordobject	19
21		удаление объекта по координате coordobject	22
22		вывод "The object {coordobject} has been deleted\n"	8
23		возврат 0	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-9.

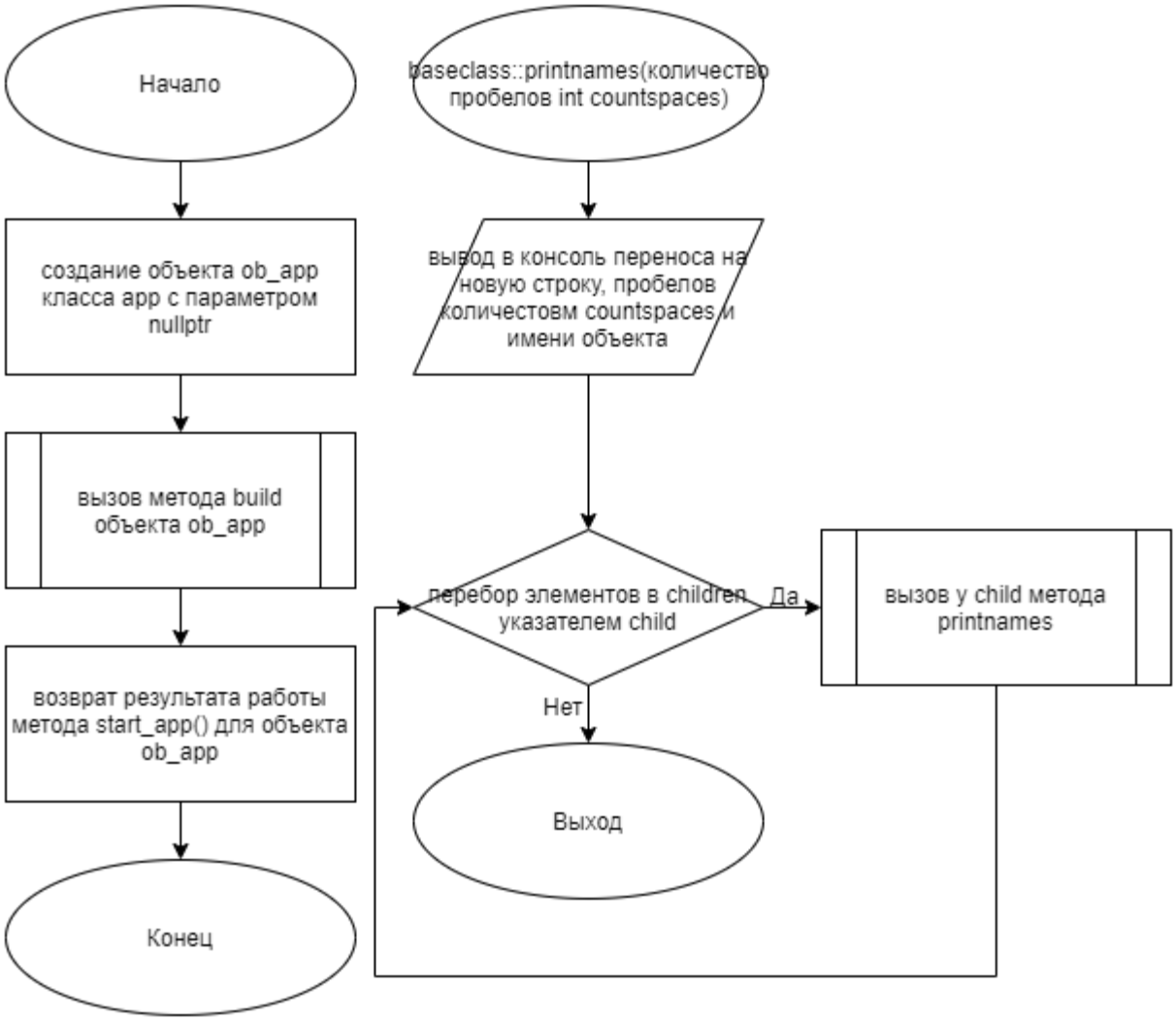

Рисунок 1 – Блок-схема алгоритма

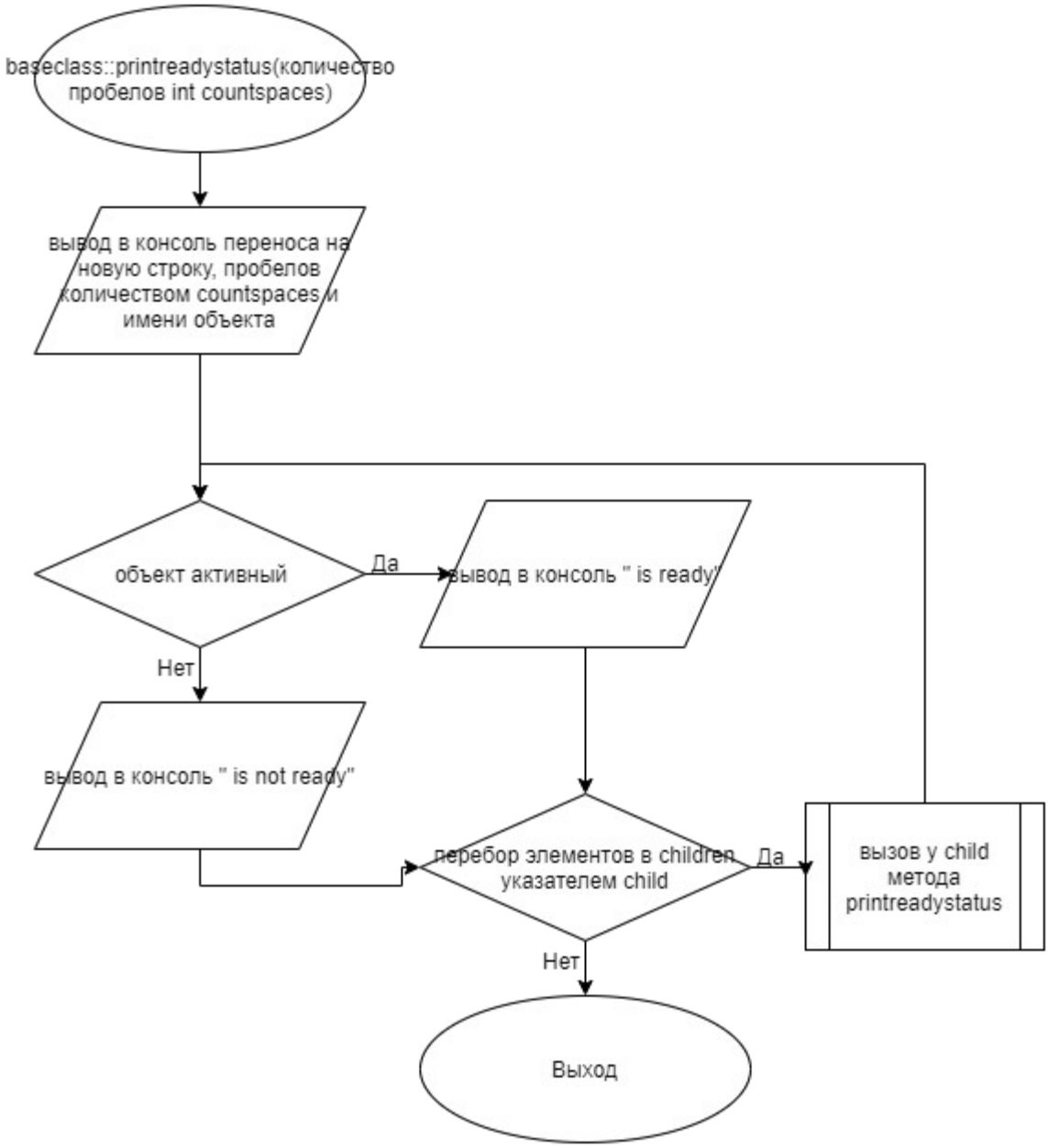

Рисунок 2 – Блок-схема алгоритма

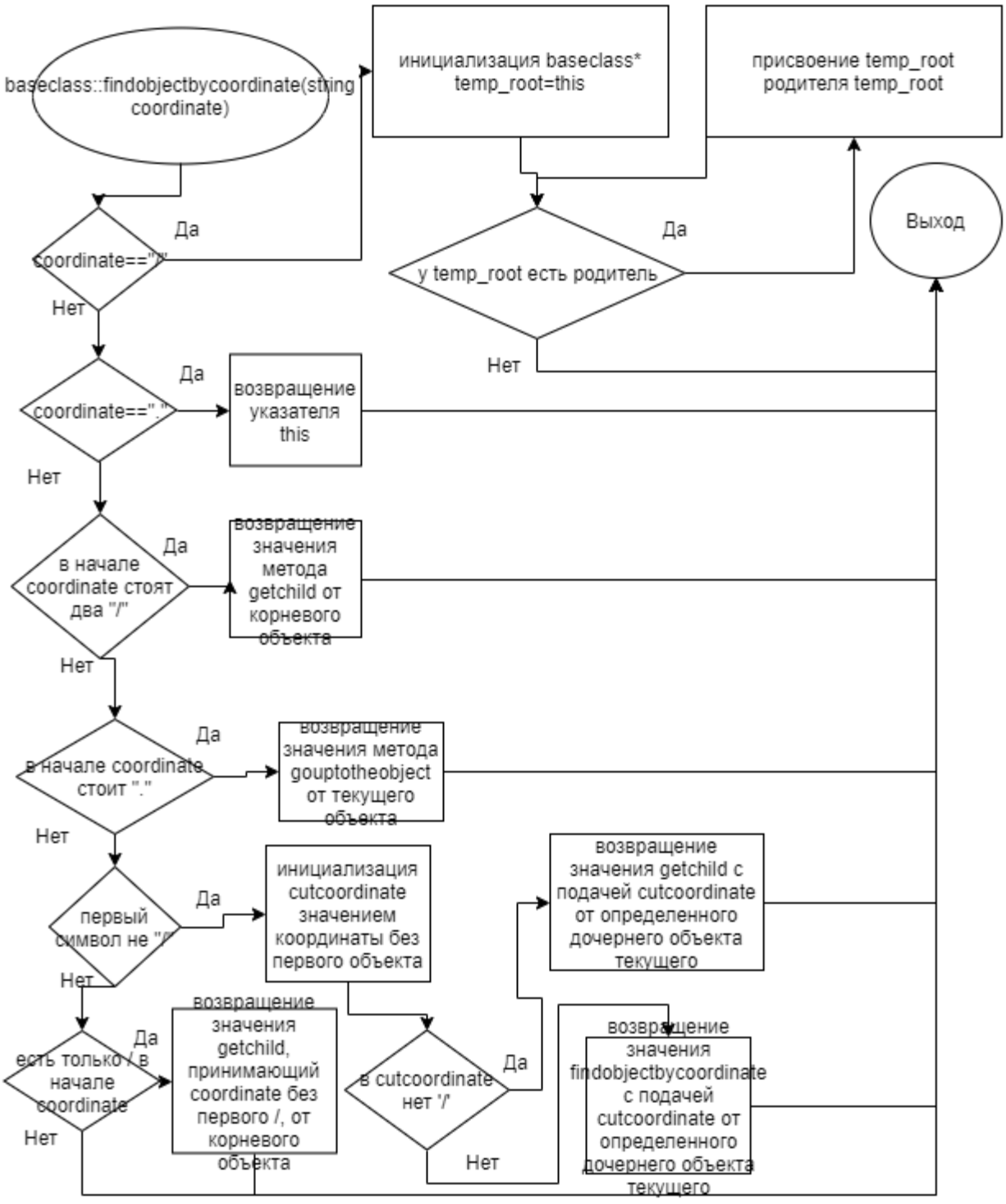

Рисунок 3 – Блок-схема алгоритма

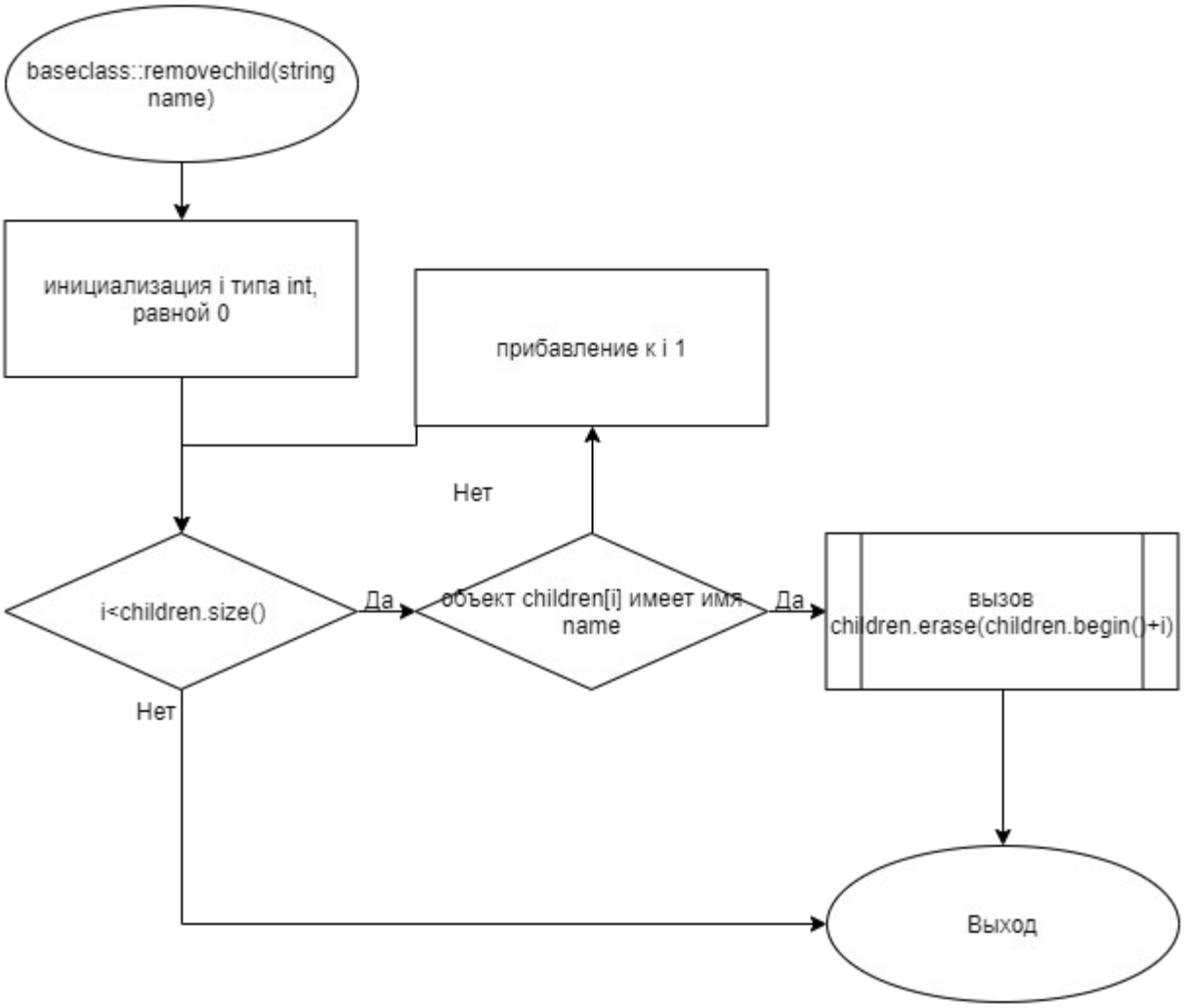

Рисунок 4 – Блок-схема алгоритма

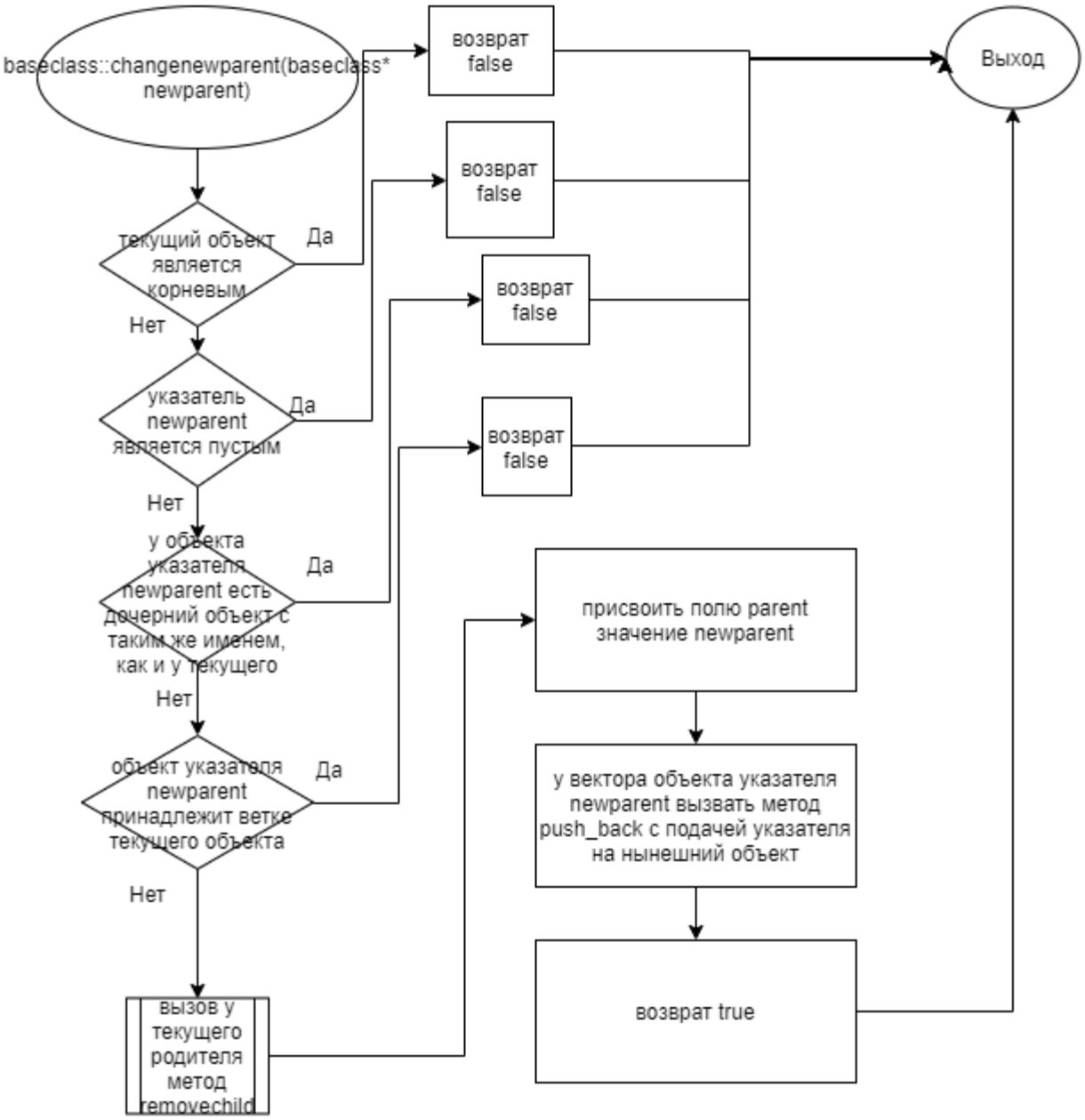

Рисунок 5 – Блок-схема алгоритма

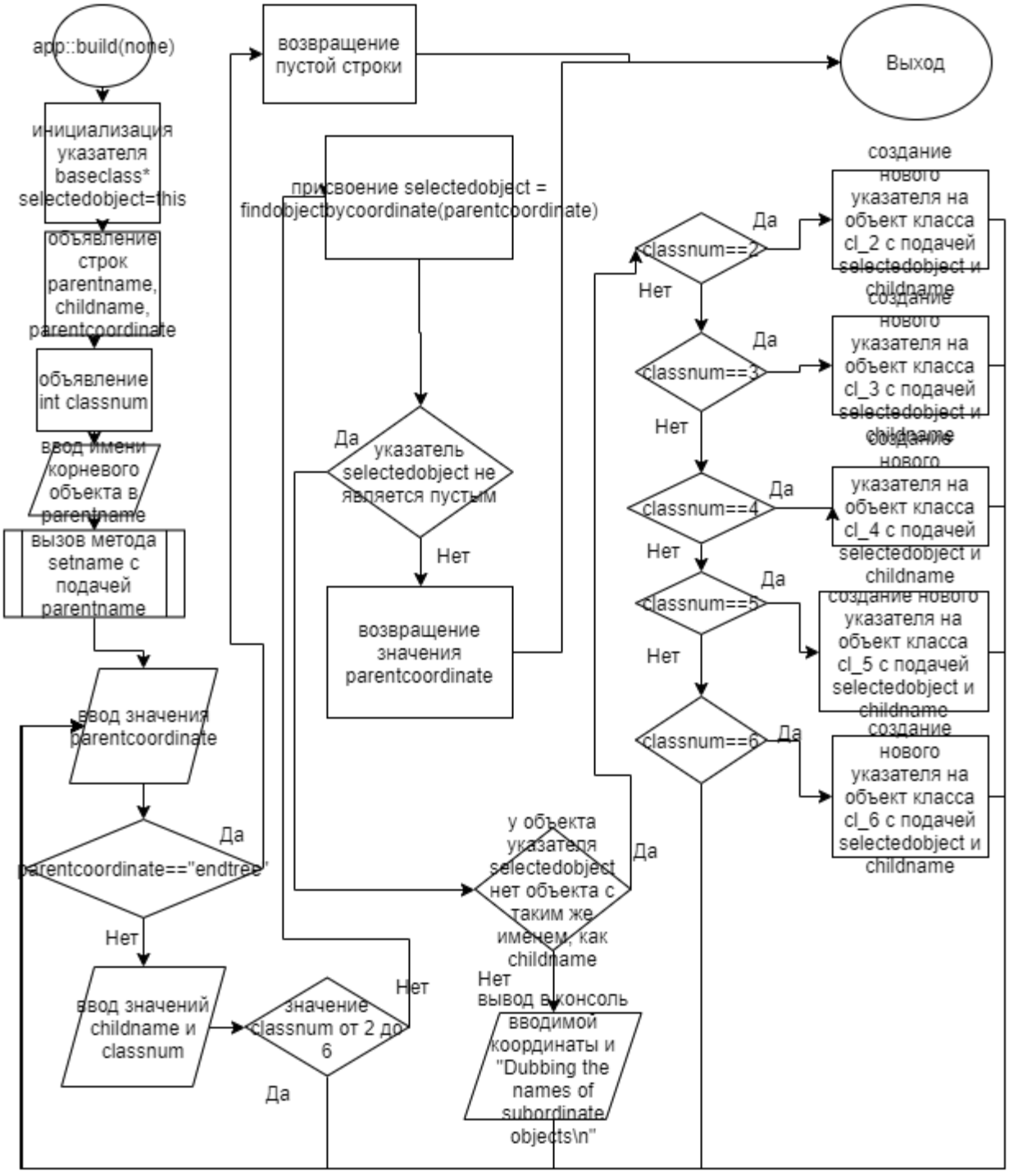

Рисунок 6 – Блок-схема алгоритма

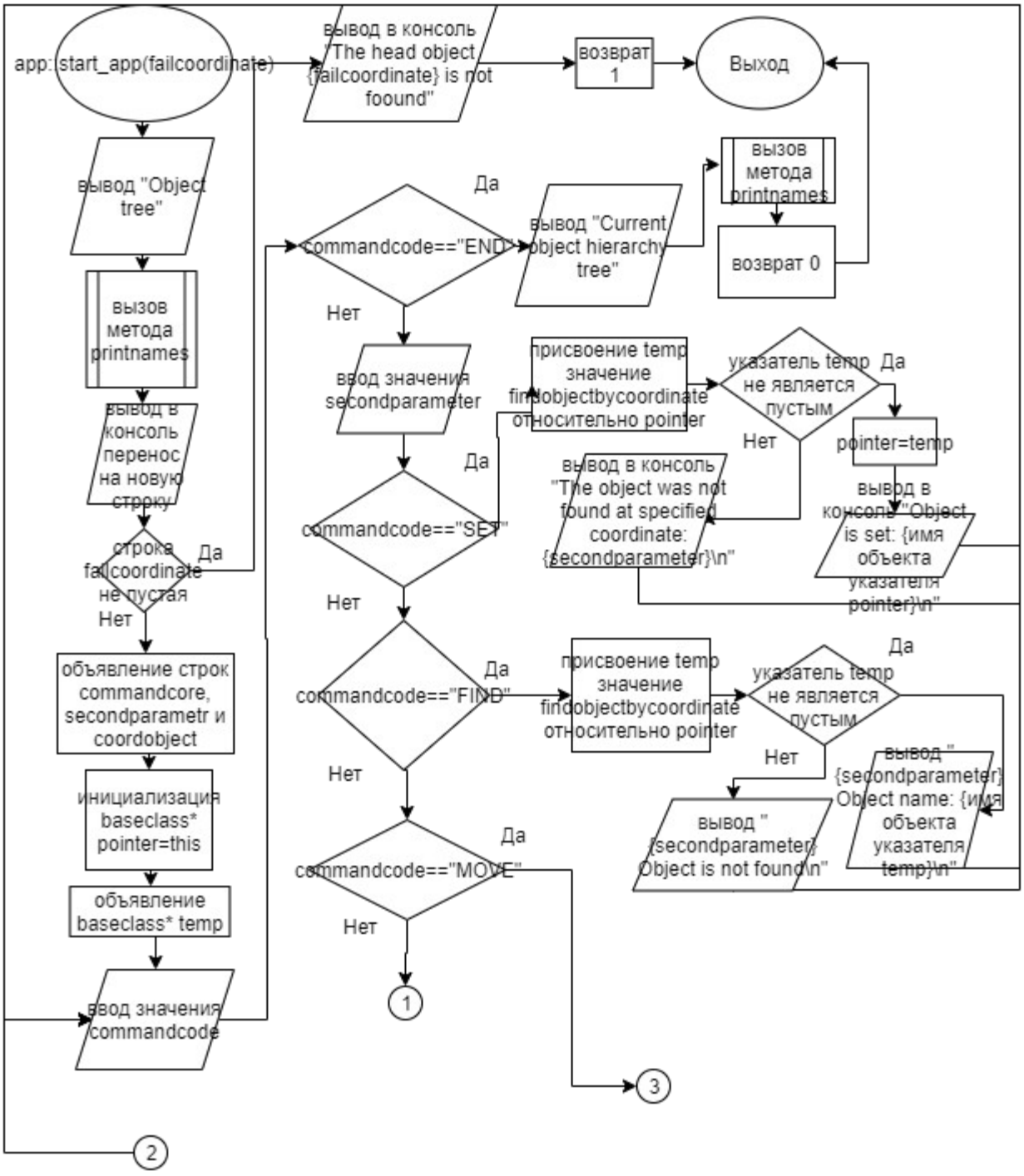

Рисунок 7 – Блок-схема алгоритма

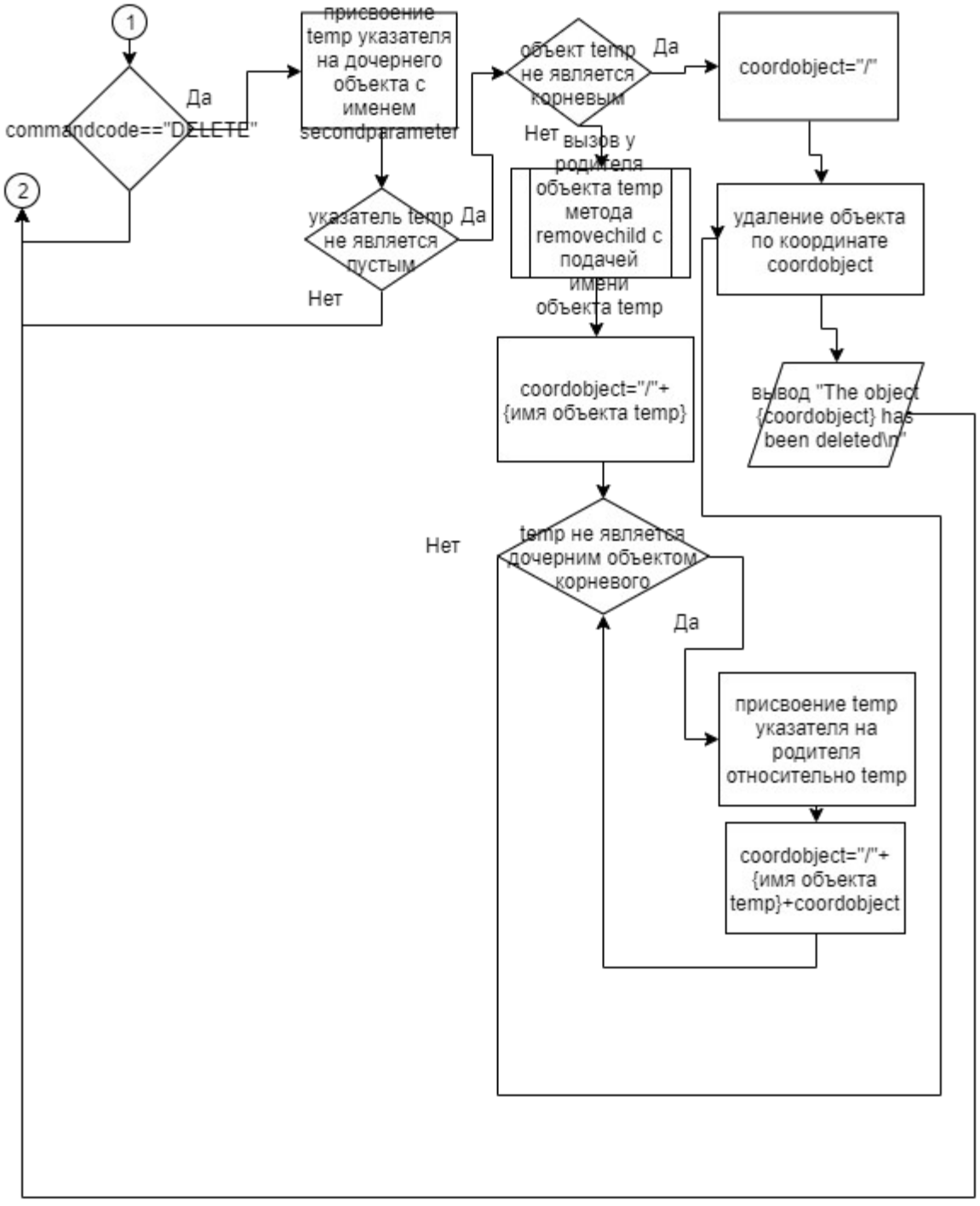

Рисунок 8 – Блок-схема алгоритма

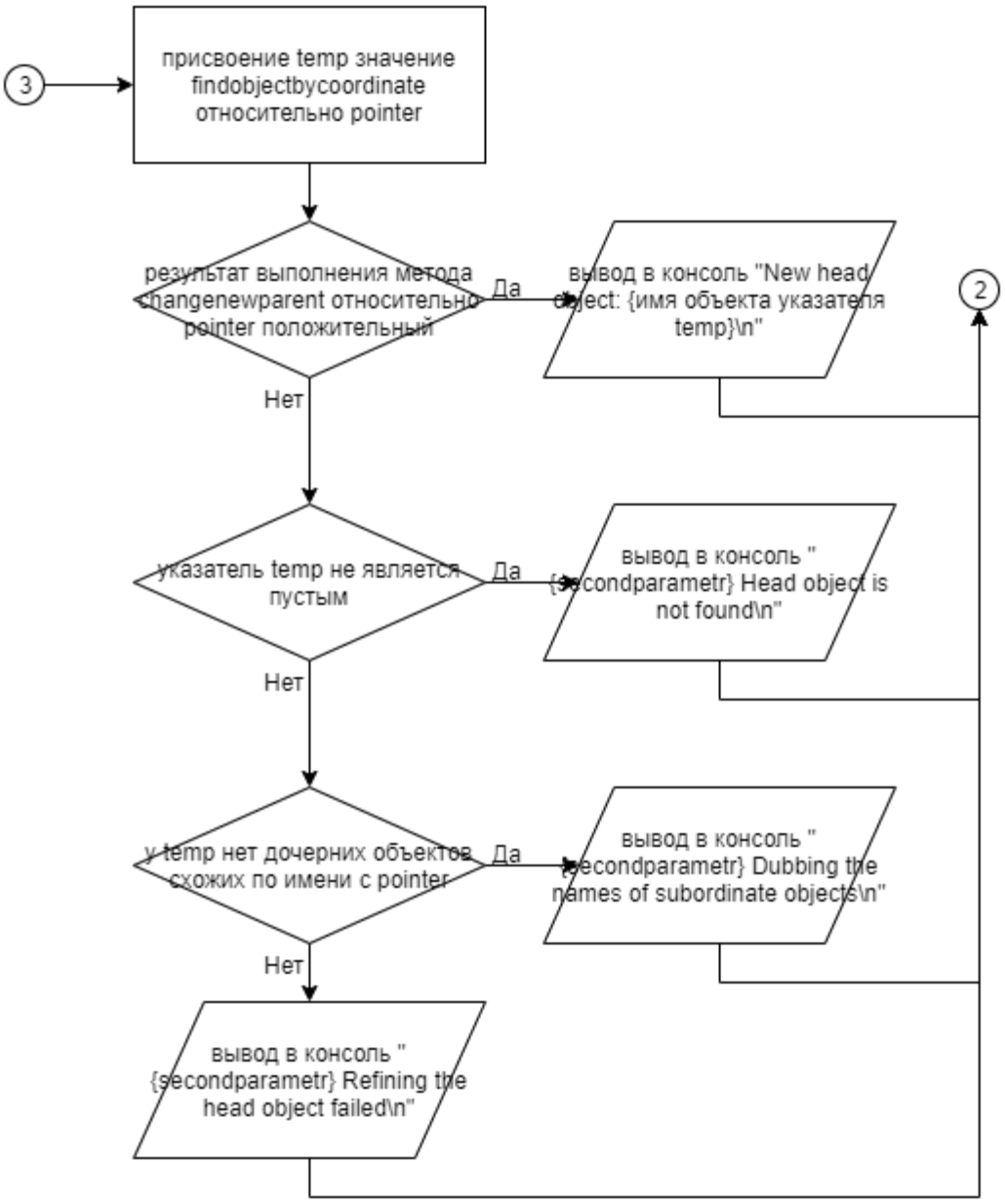

Рисунок 9 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл app.cpp

Листинг 1 – app.cpp

```
#include "app.h"
#include <iostream>
#include <string>

using namespace std;

app::app(baseclass* parent): baseclass(parent) {}

int app::start_app() { /* Edited */
   cout << "Object tree";
   printNames();
   cout << endl;
   if (failcoord.size() != 0)
   {
      cout << "The head object "<< failcoord << " is not found";
      return 1;
   }
   string commandCode, secondParameter, coordObject;
   baseclass* pointer = this;
   baseclass* temp;
   while (true)
   {
      cin >> commandCode;
      if (commandCode == "END")
      {
         cout << "Current object hierarchy tree";
         printNames();
         break;
      }
      cin >> secondParameter;
      if (commandCode == "SET")
      {
         temp = pointer->findObjectByCoordinate(secondParameter);
         if (temp != nullptr)
         {
            pointer = temp;
            cout << "Object is set: " << pointer->getName() << endl;
         }
         else
```

```
            {
               cout << "The object was not found at the specified coordinate: "
<< secondParameter << endl;
            }
         }
         if (commandCode == "FIND")
         {
            temp = pointer->findObjectByCoordinate(secondParameter);
            if (temp != nullptr)
               cout << secondParameter << "        Object name: " << temp-
>getName() << endl;
            else
               cout << secondParameter << "     Object is not found\n";
         }
         if (commandCode == "MOVE")
         {
            temp = pointer->findObjectByCoordinate(secondParameter);
            if (pointer->changeNewParent(temp))
               cout << "New head object: " << temp->getName() << endl;
            else if (temp == nullptr) cout << secondParameter << "       Head
object is not found\n";
            else if (temp->getChild(pointer->getName()) != nullptr) cout <<
secondParameter << "     Dubbing the names of subordinate objects\n";
            else cout << secondParameter << "       Redefining the head object
failed\n";
         }
         if (commandCode == "DELETE")
         {
            temp = pointer->getChild(secondParameter);
            if (temp != nullptr)
            {
               if (temp->getParent() == nullptr) coordObject = "/";
               else
               {
                  temp->getParent()->removeChild(temp->getName());
                  coordObject = "/" + temp->getName();
                  while (temp->getParent()->getParent() != nullptr)
                  {
                     temp = temp->getParent();
                     coordObject = "/" + temp->getName() + coordObject;
                  }
               }
               delete this->findObjectByCoordinate(coordObject);
               cout << "The object " << coordObject << " has been deleted\n";
            }
         }
      }
      return 0;
   }

   void app::build() { /* Edited */
      baseclass* selectedObject = this;
      string parentName, childName, parentCoordinate;
      int classNum;
      cin >> parentName;
```

```
        setName(parentName);
        while (true) {
           cin >> parentCoordinate;
           if (parentCoordinate == "endtree") break;
           cin >> childName >> classNum;
           if (classNum < 2 || classNum > 6) continue;
           selectedObject = findObjectByCoordinate(parentCoordinate);
           if (selectedObject != nullptr)
           {
              if (selectedObject->getChild(childName) == nullptr)
              {
                 switch (classNum)
                 {
                    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;
                 }
              }
              else cout << parentCoordinate << "       Dubbing the names of
     subordinate objects\n";
           }
           else
           {
              failcoord = parentCoordinate;
              break;
           }
        }
     }
```

5.2 Файл app.h

Листинг 2 – app.h

```
     #ifndef __APP__H
     #define __APP__H
     #include "baseclass.h"
```

```
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include "cl_6.h"

class app : public baseclass {
   private:
      string failcoord = "";
   public:
      app(baseclass* parent);
      void build();
      int start_app();

};
#endif
```

5.3 Файл baseclass.cpp

Листинг 3 – baseclass.cpp

```
#include "baseclass.h"

baseclass::baseclass(baseclass*   parent,   string   name):   parent(parent),
name(name) {
   if (parent != nullptr) {
      parent->children.push_back(this);
   }
}

baseclass::~baseclass() {
   for  (int  ChildIndex  =  0;  ChildIndex  <  children.size();  ChildIndex++)
delete children[ChildIndex];
}

string baseclass::getName() {return name;}
baseclass* baseclass::getParent() {return parent;}

bool baseclass::setName(string newName) {
   if  (getParent()  !=  nullptr  &&  getParent()->getChild(newName))  return
false;
   name = newName;
   return true;
}

void baseclass::printNames(int countSpaces) /* Edited */
{

   cout << endl << string(countSpaces, ' ') << this->getName();
   for (baseclass* child : this->children)
```

```
        child->printNames(countSpaces+4);
    }

    baseclass* baseclass::getChild(string name) {
       for (baseclass* child : children) {
          if (child->name == name) return child;
       }
       return nullptr;
    }

    baseclass* baseclass::goUpToTheObject(string name)
    {
       for (int childIndex = 0; childIndex < children.size(); childIndex++)
       {
          if (name == children[childIndex]->getName())
             return children[childIndex];
          if (children[childIndex]->children.size() > 0)
          {
             baseclass*        resultFromUp        =        children[childIndex]-
    >goUpToTheObject(name);
             if (resultFromUp != nullptr) return resultFromUp;
          }
       }
       return nullptr;
    }

    baseclass* baseclass::goDownToTheObject(string name)
    {
       if (name == getName()) return this;
       if (parent == nullptr) return goUpToTheObject(name);
       return parent->goDownToTheObject(name);
    }

    void baseclass::setReadyCode(int code)
    {
       if (parent == nullptr || parent->readyCode != 0)
          this->readyCode = code;
       if (!code)
       {
          this->readyCode = code;
          for (int childIndex = 0; childIndex < children.size(); childIndex++)
             children[childIndex]->setReadyCode(code);
       }
    }

    void baseclass::printReadyStatus(int countSpaces) /* Edited */
    {
       cout << endl << string(countSpaces, ' ') << this->getName();
       if (this->readyCode) cout << " is ready";
       else cout << " is not ready";
       for (baseclass* child : this->children)
          child->printReadyStatus(countSpaces+4);
    }

    baseclass* baseclass::findObjectByCoordinate(string coordinate) /* Added */
```

```
{
   if (coordinate == "/")
   {
      baseclass* temp_root = this;
      while (temp_root->getParent())
         temp_root = temp_root->getParent();
      return temp_root;
   }

   if (coordinate == ".") return this;

   if (coordinate[0] == '/' && coordinate[1] == '/')
      return findObjectByCoordinate("/")->getChild(coordinate.substr(2));

   if (coordinate[0] == '.')
      return this->goUpToTheObject(coordinate.substr(1));

   if (coordinate[0] != '/')
   {
      string cutCoordinate = coordinate.substr(coordinate.find('/')+1);
      if (cutCoordinate.find('/') == -1)
         return  this->getChild(coordinate.substr(0,  coordinate.find('/')))-
>getChild(cutCoordinate);
      else
         return  this->getChild(coordinate.substr(0,  coordinate.find('/')))-
>findObjectByCoordinate(cutCoordinate);
   }
   else
   {
      if (coordinate.substr(1).find('/') == -1)
         return findObjectByCoordinate("/")->getChild(coordinate.substr(1));
      return                                    findObjectByCoordinate("/")-
>findObjectByCoordinate(coordinate.substr(1));
   }
}

void baseclass::removeChild(string name) /* Added */
{
   for (int i = 0; i < children.size(); i++)
   {
      if (children[i]->getName() == name)
      {
         children.erase(children.begin() + i);
         break;
      }
   }
}

bool baseclass::changeNewParent(baseclass* newParent) /* Added */
{
   if (this->getParent() == nullptr) return false;
   if (newParent == nullptr) return false;
   if (newParent->getChild(this->getName()) != nullptr) return false;
   if  (this->goUpToTheObject(newParent->getName())  ==  newParent)  return
false;
```

```
   this->getParent()->removeChild(this->getName());
   this->parent = newParent;
   newParent->children.push_back(this);
   return true;
}
```

5.4 Файл baseclass.h

Листинг 4 – baseclass.h

```
#ifndef __BASECLASS__H
#define __BASECLASS__H
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class baseclass {
   private:
      string name;
      baseclass* parent = nullptr;
      vector <baseclass*> children;
      int readyCode = 0;

   public:
      baseclass(baseclass* parent, string name = "Object_Root");
      ~baseclass();
      bool setName(string name);
      baseclass* getParent();
      string getName();
      void printNames(int countSpaces = 0);
      baseclass* getChild(string name);
      baseclass* goUpToTheObject(string name);
      baseclass* goDownToTheObject(string name);
      void setReadyCode(int code);
      void printReadyStatus(int countSpaces = 0);
      baseclass* findObjectByCoordinate(string coordinate);
      void removeChild(string name);
      bool changeNewParent(baseclass* newParent);
};
#endif
```

5.5 Файл cl_2.cpp

Листинг 5 – cl_2.cpp

```
#include "cl_2.h"

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

5.6 Файл cl_2.h

Листинг 6 – cl_2.h

```
#ifndef __CL_2__H
#define __CL_2__H
#include "baseclass.h"

class cl_2:public baseclass
{
   public:
      cl_2(baseclass* parent, string name);
};
#endif
```

5.7 Файл cl_3.cpp

Листинг 7 – cl_3.cpp

```
#include "cl_3.h"

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

5.8 Файл cl_3.h

Листинг 8 – cl_3.h

```
#ifndef __CL_3__H
#define __CL_3__H
#include "baseclass.h"
```

```
class cl_3:public baseclass
{
   public:
      cl_3(baseclass* parent, string name);
};
#endif
```

5.9 Файл cl_4.cpp

Листинг 9 – cl_4.cpp

```
#include "cl_4.h"

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

5.10 Файл cl_4.h

Листинг 10 – cl_4.h

```
#ifndef __CL_4__H
#define __CL_4__H
#include "baseclass.h"

class cl_4:public baseclass
{
   public:
      cl_4(baseclass* parent, string name);
};
#endif
```

5.11 Файл cl_5.cpp

Листинг 11 – cl_5.cpp

```
#include "cl_5.h"

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

5.12 Файл cl_5.h

Листинг 12 – cl_5.h

```
#ifndef __CL_5__H
#define __CL_5__H
#include "baseclass.h"

class cl_5:public baseclass
{
   public:
      cl_5(baseclass* parent, string name);
};
#endif
```

5.13 Файл cl_6.cpp

Листинг 13 – cl_6.cpp

```
#include "cl_6.h"

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

5.14 Файл cl_6.h

Листинг 14 – cl_6.h

```
#ifndef __CL_6__H
#define __CL_6__H
#include "baseclass.h"

class cl_6:public baseclass
{
   public:
      cl_6(baseclass* parent, string name);
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include "app.h"

int main()
{
   app ob_app(nullptr);
   ob_app.build();
   return (ob_app.start_app());
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 10.

Таблица 10 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4
object_7 3
endtree
FIND
object_2/object_4
SET /object_2
FIND //object_7
FIND
object_4/object_7
FIND .
FIND .object_7
FIND
object_4/object_7
MOVE .object_7
SET
object_4/object_7
MOVE //object_1
MOVE /object_3
END
	Object tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_7
	object_5
	object_3
	object_3
	object_2/object_4
	Object name:
	object_4
	Object is set:
	object_2
	//object_7
	Object is not found
	object_4/object_7
	Object name:
	object_7
	. Object name:
	object_2
	.object_7 Object
	name: object_7
	object_4/object_7
	Object name:
	object_7
	.object_7
	Redefining the head
	object failed
	Object is set:
	object_7
	//object_1
	Dubbing the names of
	subordinate objects
	New head object:
	object_3
	Current object
	hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_5
		Object tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_7
		object_5
		object_3
		object_3
		object_2/object_4
		Object name:
		object_4
		Object is set:
		object_2
		//object_7
		Object is not found
		object_4/object_7
		Object name:
		object_7
		. Object name:
		object_2
		.object_7 Object
		name: object_7
		object_4/object_7
		Object name:
		object_7
		.object_7
		Redefining the head
		object failed
		Object is set:
		object_7
		//object_1
		Dubbing the names of
		subordinate objects
		New head object:
		object_3
		Current object
		hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_5

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_3 object_3 object_7	object_3 object_3 object_7
rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree SET /object_1/object_7 MOVE /object_4 END	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 Object is set: object_7 /object_4 Head object is not found Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 Object is set: object_7 /object_4 Head object is not found Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3
rootela / object_1 3 / object_2 2 /object_3 object_fail 3	Object tree rootela object_1 object_2 The head object /object_3 is not found	Object tree rootela object_1 object_2 The head object /object_3 is not found
rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree SET object_2/object_7 END	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 The object was not found at the specified coordinate: object_2/object_7	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 The object was not found at the specified coordinate:

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3	object_2/object_7 Current object hierarchy tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3
rootela / object_1 3 / object_2 2 /object_2 object_4 3 /object_2 object_5 4 / object_3 3 /object_2 object_3 6 /object_1 object_7 5 /object_2/object_4 object_7 3 endtree FIND /object_1/object_7 SET /object_1 DELETE object_7 FIND . SET / END	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 /object_1/object_7 Object name: object_7 Object is set: object_1 The object /object_1/object_7 has been deleted . Object name: object_1 Object is set: rootela Current object hierarchy tree rootela object_1 object_2 object_4 object_7 object_5 object_3 object_3	Object tree rootela object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 /object_1/object_7 Object name: object_7 Object is set: object_1 The object /object_1/object_7 has been deleted . Object name: object_1 Object is set: rootela Current object hierarchy tree rootela object_1 object_2 object_4 object_7 object_5 object_3 object_3

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).