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

Для решения задачи используется:

- функция erase для удаление элемента из массива;
- функция substr для возвращение подстроки из строки;
- функция find для возвращение индекса символа в строке.

Класс cl_base:

- функционал:
    - метод find_obj_bc — Найти объект по координатам;
    - метод Del_obj — Удалить объект;
    - метод Move_head — Изменить головной объект для производного.

Класс cl_application:

- функционал:
    - метод Tree — Создание дерева иерархии объектов;
    - метод Start — Начало работы приложения.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	cl_base				
		cl_application	public		2
2	cl_application				

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 метода find_obj_bc класса cl_base

Функционал: ищет объект по координате.

Параметры: string coord - координата.

Возвращаемое значение: cl_base* - указатель на базовый класс.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода find_obj_bc класса cl_base

№	Предикат	Действия	№ перехода
1		объявление string s_name; int index	2
2		инициализация cl_base* ob=nullptr	3
3	координата имеет ввид "/"	вернуть метод get_root для this	∅
			4
4	координата имеет ввид "."	вернуть this	∅
			5
5	координата имеет ввид "//obj"	присвоение s_name=coord.substr(2); вернуть метод Find_global для this с параметром s_name	∅
			6
6	координата имеет ввид ".obj"	присвоение s_name=coord.substr(1); вернуть метод Find_current для this с параметром s_name	∅
			7
7		инициализация index = coord.find("/",1)	8

№	Предикат	Действия	№ перехода
8	координата имеет ввид "/obj1/obj2"		11
			9
9	координата имеет ввид "obj1/obj2"		13
			10
10		вернуть nullptr	∅
11	index !=-1	присвоение s_name = coord.substr(1,index-1); присвоение ob = поиск элемента среди подчиненных корню с параметром s_name	12
		присвоение s_name=coord.substr(1); вернуть метод Get_ptr с параметром s_name для корня	∅
12	ob	вернуть метод find_obj_bc с параметром coord.substr(index+1)	∅
		вернуть ob	∅
13	index !=-1	присвоение s_name = coord.substr(0,index); присвоение ob = поиск элемента среди подчиненных this с параметром s_name	14
		вернуть метод Get_ptr для this с параметром coord	∅
14	ob	вернуть метод find_obj_bc с параметром coord.substr(index+1)	∅
		вернуть ob	∅

3.2 Алгоритм метода Del_obj класса cl_base

Функционал: удаляет объект.

Параметры: string name - имя объекта.

Возвращаемое значение: void - не возвращает значений.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода Del_obj класса cl_base

№	Предикат	Действия	№ перехода
1		инициализация cl_base* p = Get_ptr(name); int i =0	2
2	auto sub:subordinate_objects		3
			∅
3	sub==p	вырезать sub из subordinate_objects; освобождение места выделенного под p	∅
		i++	2

3.3 Алгоритм метода Tree класса cl_application

Функционал: Создание дерева иерархии объектов.

Параметры: нет.

Возвращаемое значение: void- не возвращает значений.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода Tree класса cl_application

№	Предикат	Действия	№ перехода
1	данные корректны	создание дерева иерархии по введенным с клавиатуры данным	∅
		вывод на экран уже созданного дерева	∅

3.4 Алгоритм метода Start класса cl_application

Функционал: Начало работы приложения.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности завершения

алгоритма.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода Start класса cl_application

№	Предикат	Действия	№ перехода
1		установить текущий объект корнем	2
2		начать ввод с клавиатуры комманд	3
3	комманда END	закончить ввод комманд	8
			4
4	комманда SET	установить текущий объект - объект по заданной координате	2
			5
5	комманда FIND	Поиск указателя на объект по координате, вывод его имени на экран	2
			6
6	комманда MOVE	изменить головной объект для текущего по координате	2
			7
7	комманда DELETE	удалить объект по имени	2
			2
8		вывести на экран конечное дерево иерархии объектов; вернуть 0	∅

3.5 Алгоритм метода Move_head класса cl_base

Функционал: Изменить головной объект для производного.

Параметры: cl_base* h - указатель на объект h базового класса.

Возвращаемое значение: bool - индикатор выполнения.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода Move_head класса cl_base

№	Предикат	Действия	№ перехода
1	this->Get_baseptr()	инициализация int i=0	2
		вернуть false	∅
2	i< кол-ва подчинненых объектов	инкремент i	3
			4
3	текущий подчинненый объект == this	вырезать sub_obj из массива; break;	4
			2
4		присвоение this->p_head_object=h	5
5		в массив подчиненных объектов нового головного добавить this	6
6		вывод с новой строки "New head object: "<<имя объекта	7
7		вернуть true	∅

3.6 Алгоритм функции main

Функционал: основной алгоритм программы.

Параметры: нет.

Возвращаемое значение: int - индикатор корректности выполнения алгоритма.

Алгоритм функции представлен в таблице 7.

Таблица 7 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1	Дерево создано корректно	вернуть метод Start	∅

№	Предикат	Действия	№ перехода
		вернуть 1	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-10.

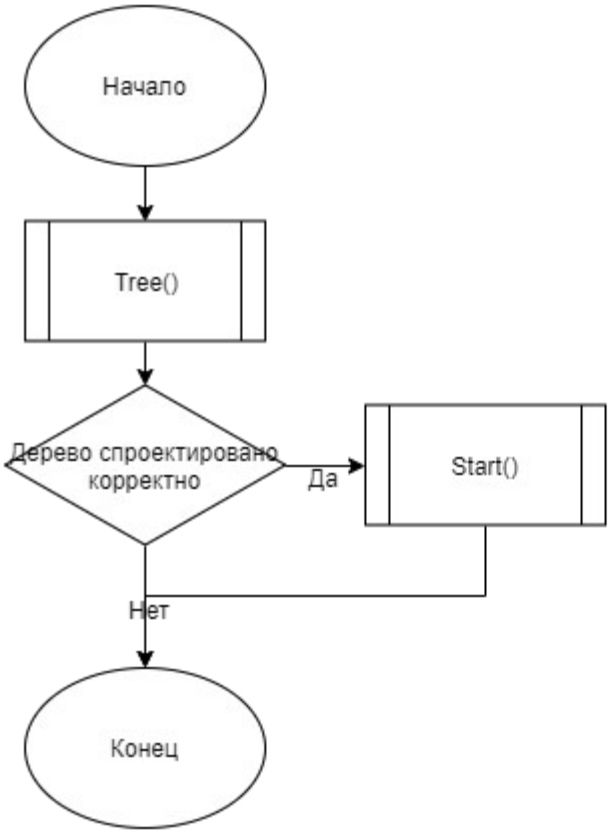

Рисунок 1 – Блок-схема алгоритма

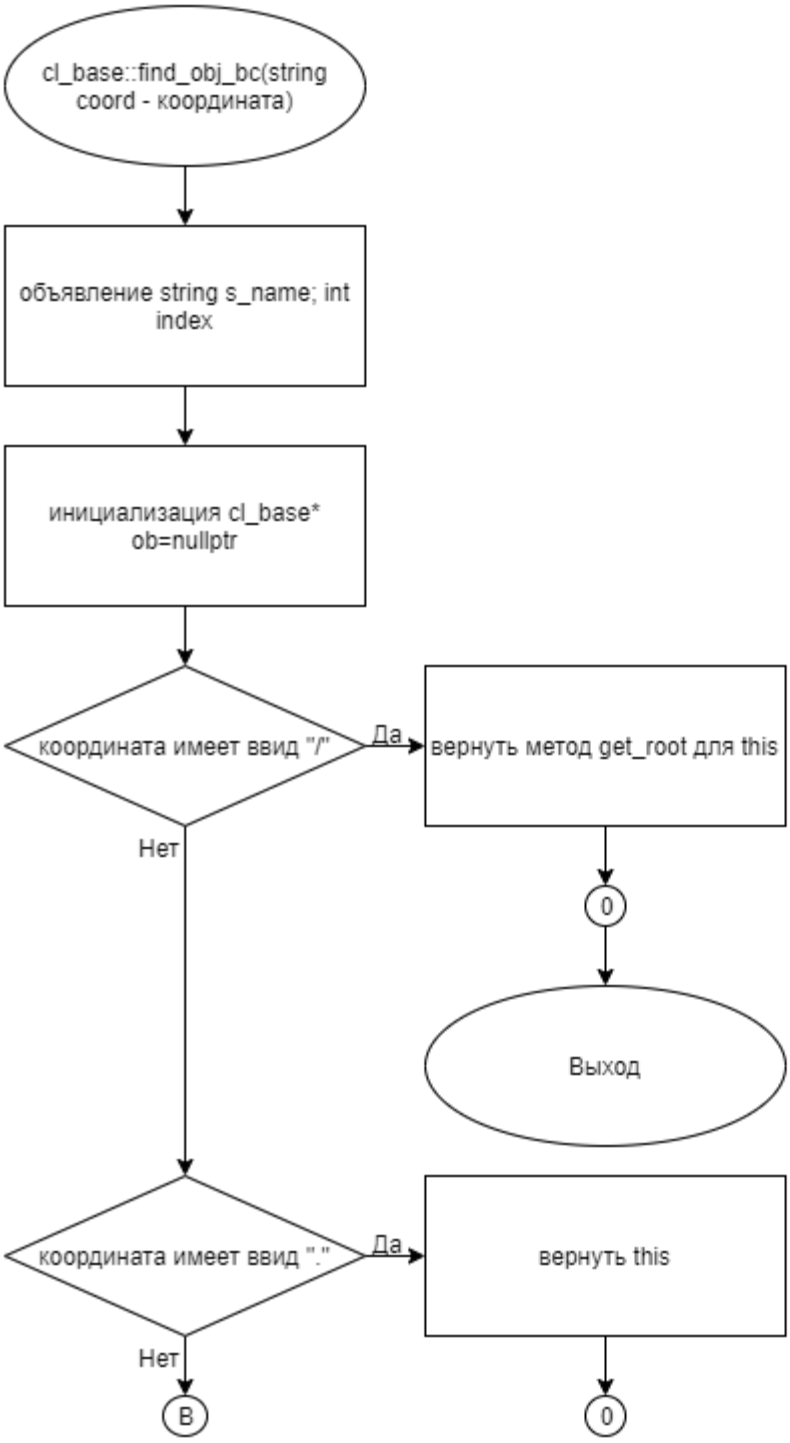

Рисунок 2 – Блок-схема алгоритма

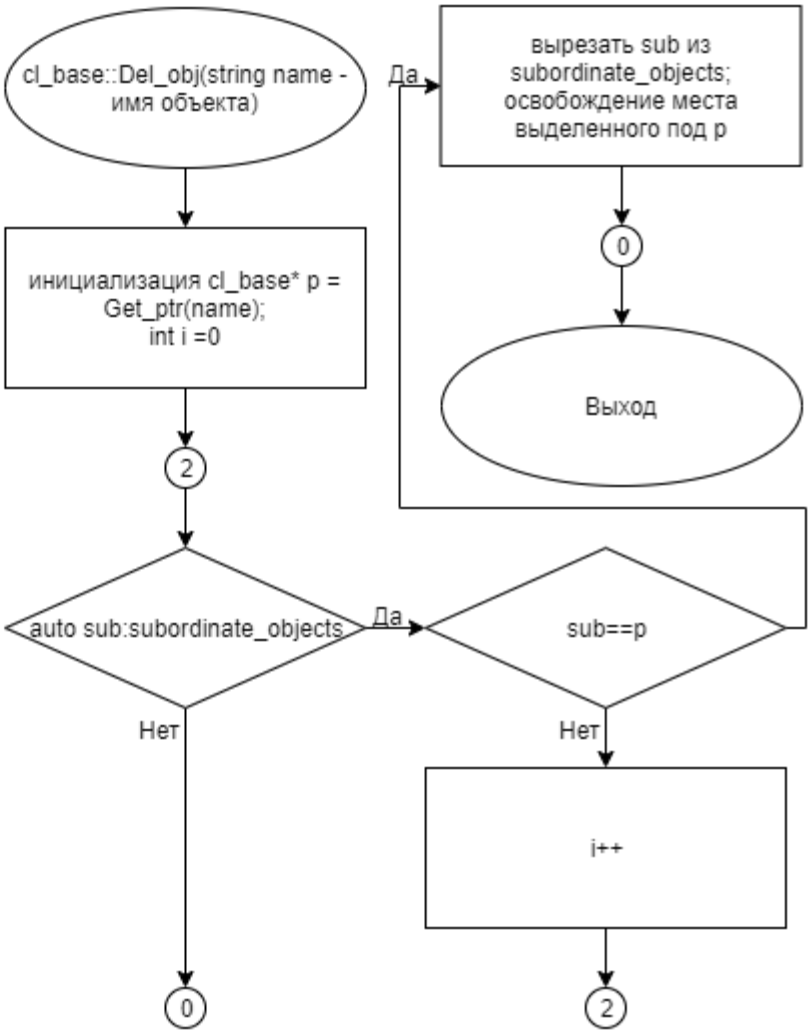

Рисунок 3 – Блок-схема алгоритма

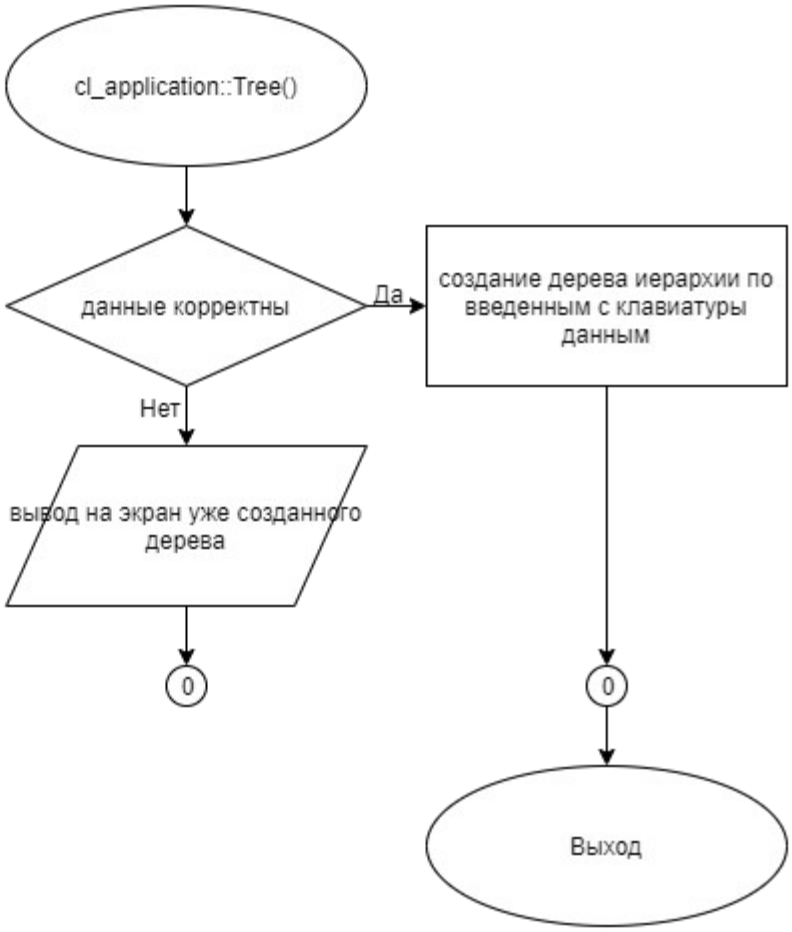

Рисунок 4 – Блок-схема алгоритма

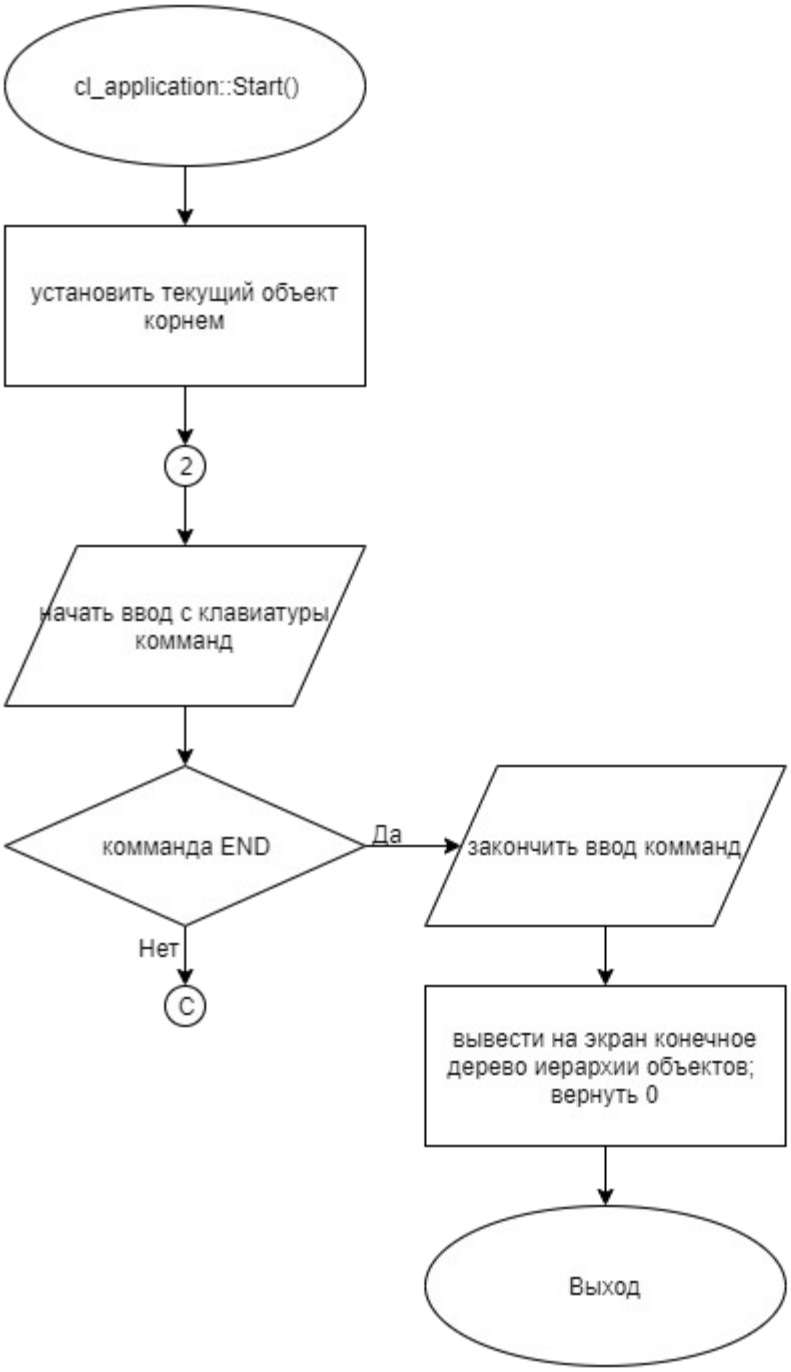

Рисунок 5 – Блок-схема алгоритма

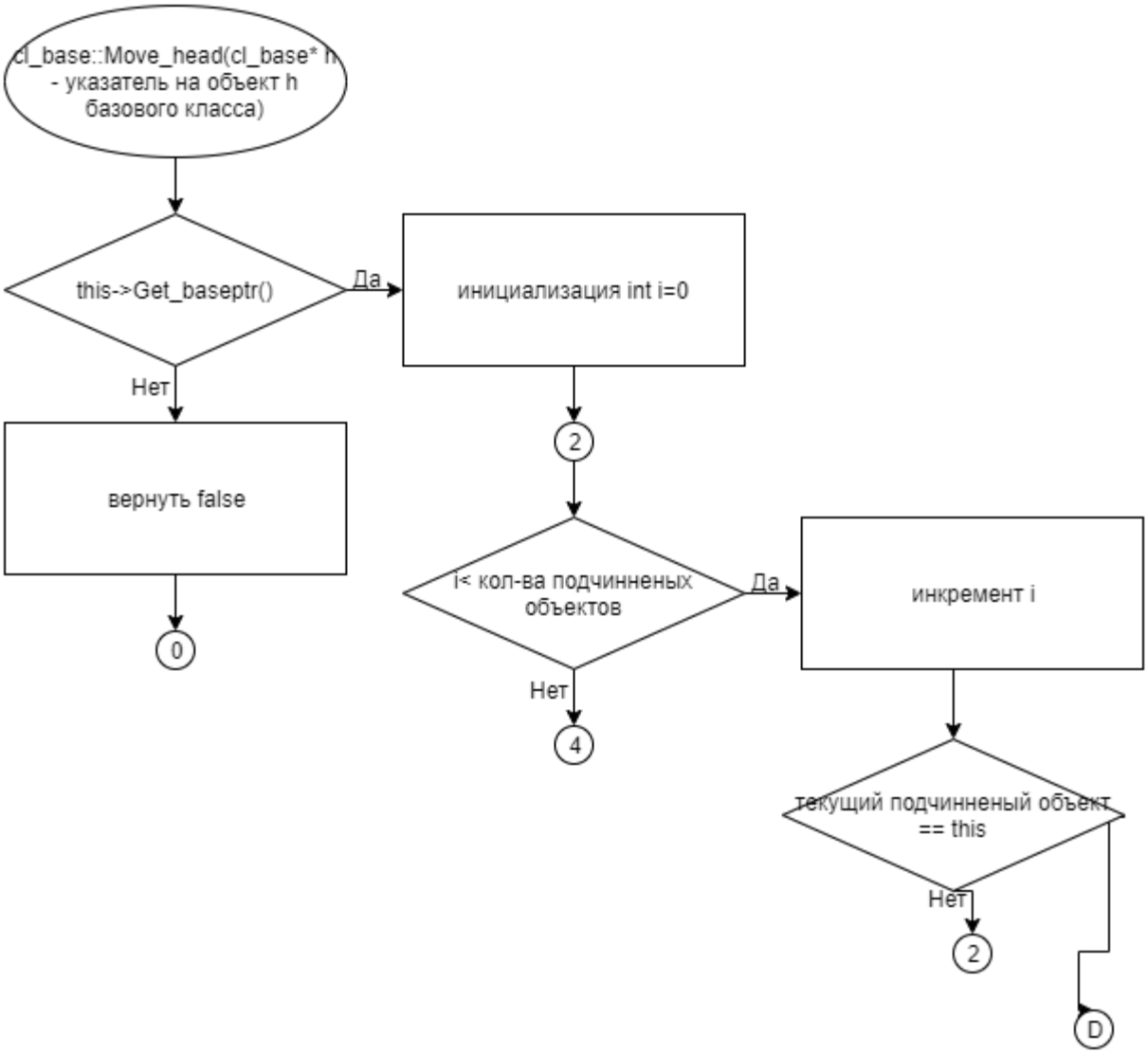

Рисунок 6 – Блок-схема алгоритма

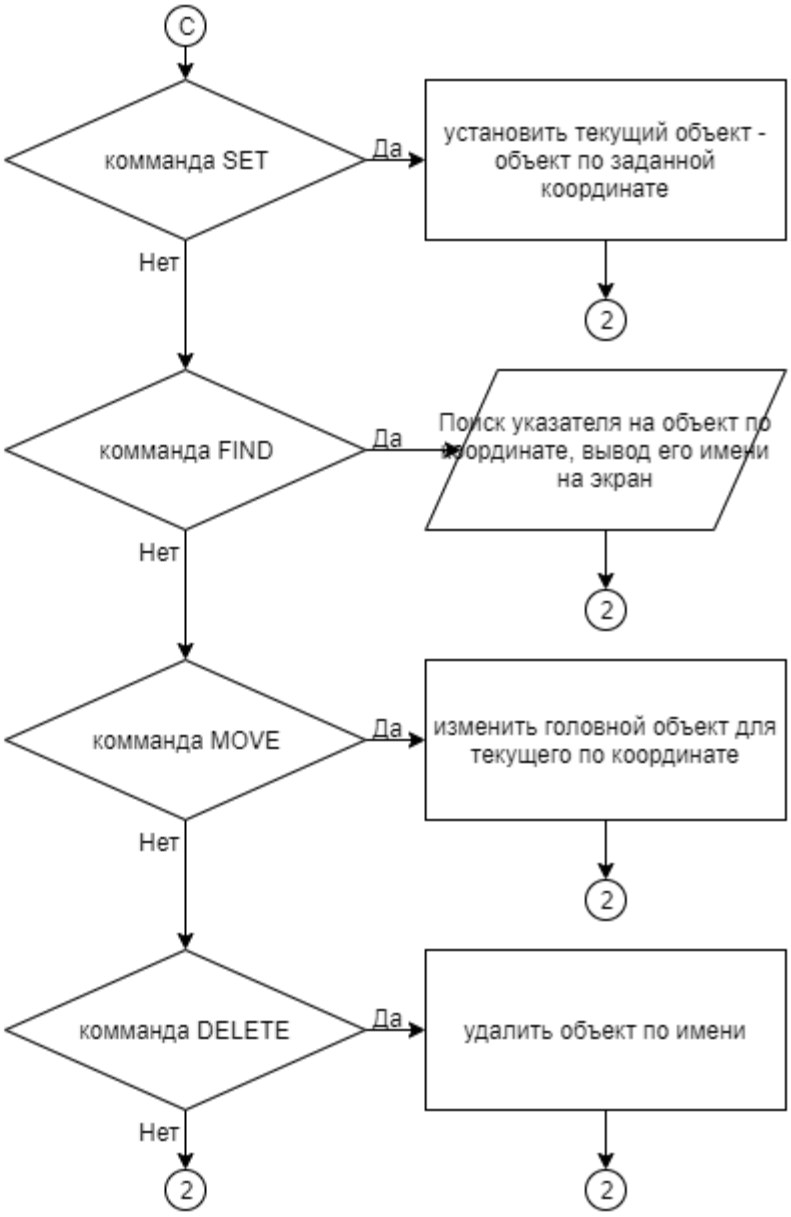

Рисунок 7 – Блок-схема алгоритма

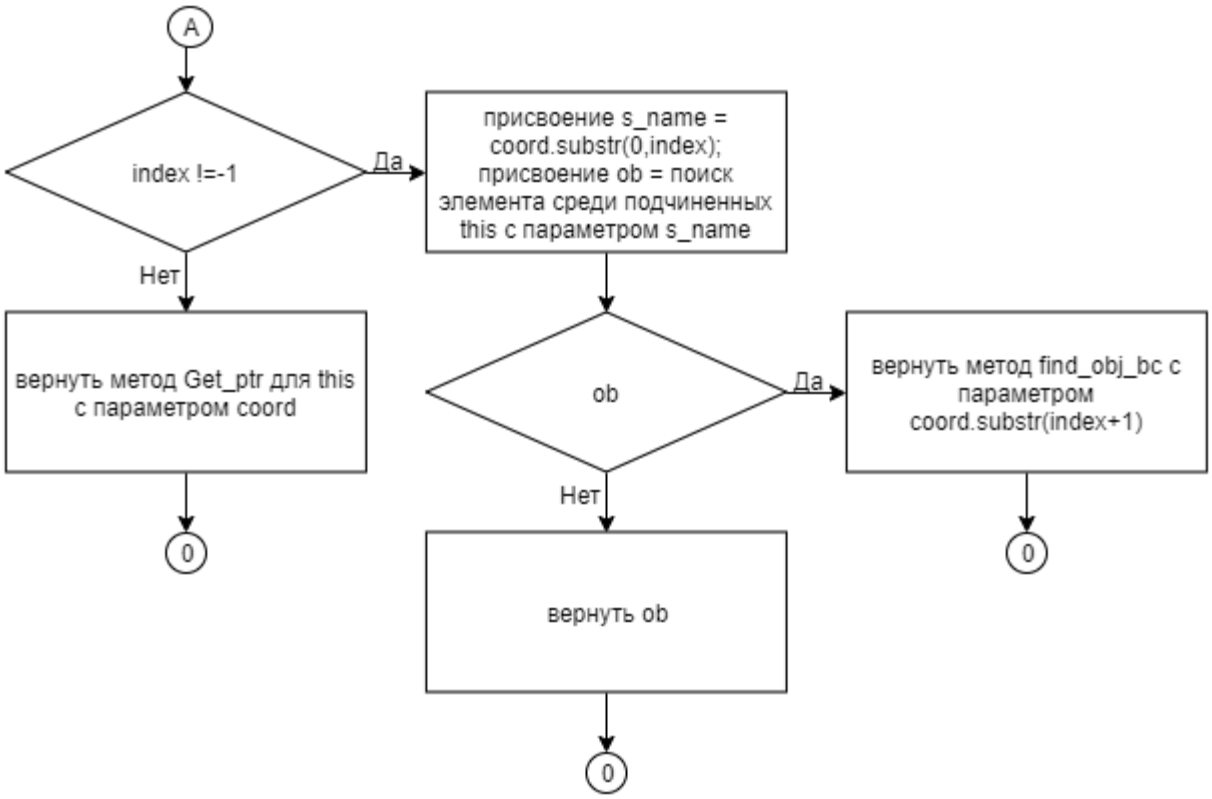

Рисунок 8 – Блок-схема алгоритма

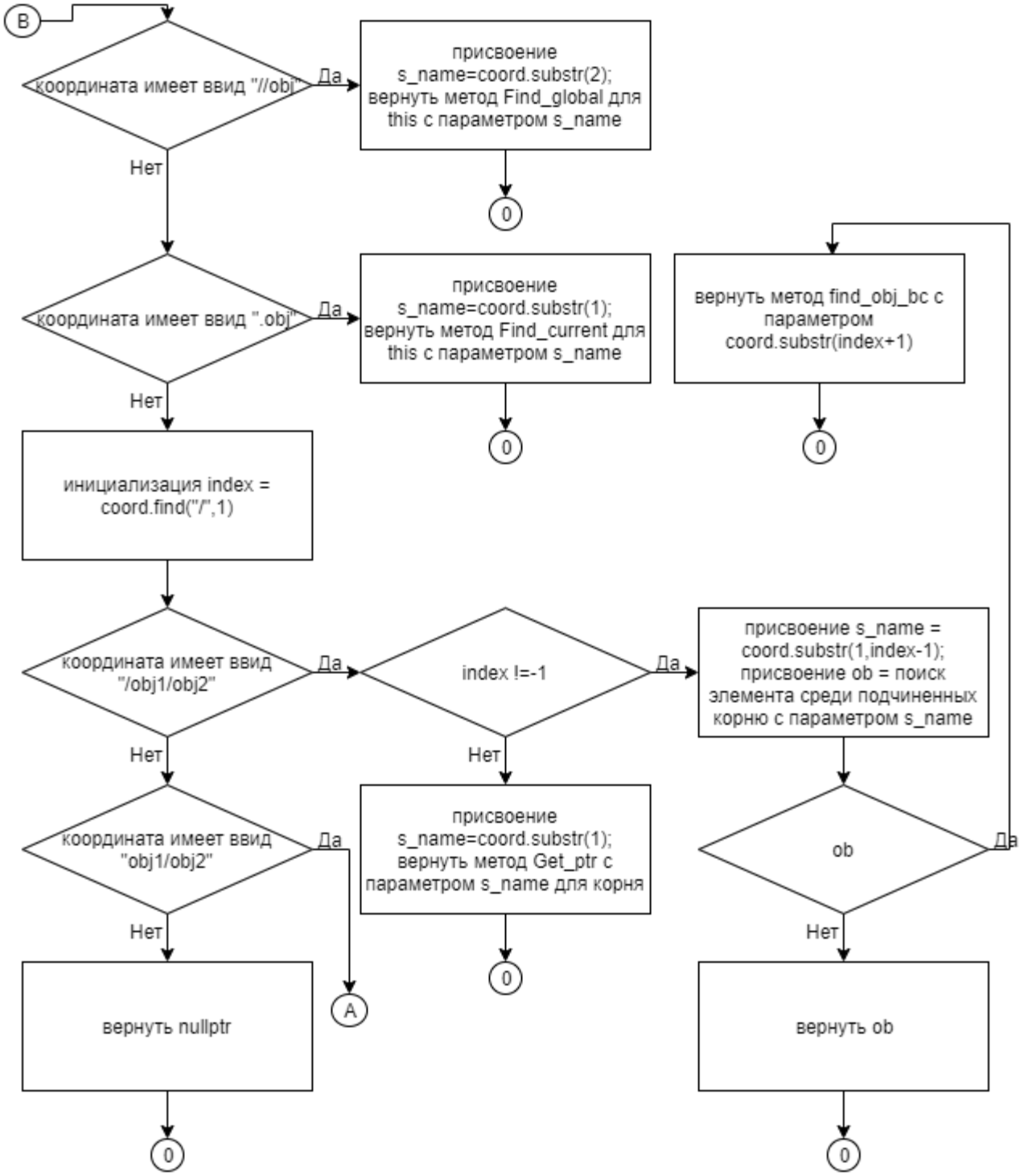

Рисунок 9 – Блок-схема алгоритма

Рисунок 10 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл cl_1.cpp

Листинг 1 – cl_1.cpp

```
#include "cl_1.h"
cl_1::cl_1(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.2 Файл cl_1.h

Листинг 2 – cl_1.h

```
#ifndef __CL_1__H
#define __CL_1__H
#include "cl_base.h"
class cl_1:public cl_base{
   public:
   cl_1(cl_base* p_head, string s_object_name);
};
#endif
```

5.3 Файл cl_2.cpp

Листинг 3 – cl_2.cpp

```
#include "cl_2.h"
cl_2::cl_2(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.4 Файл cl_2.h

Листинг 4 – 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* p_head, string s_object_name);
};
#endif
```

5.5 Файл cl_3.cpp

Листинг 5 – cl_3.cpp

```
#include "cl_3.h"
cl_3::cl_3(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.6 Файл cl_3.h

Листинг 6 – 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* p_head, string s_object_name);
};


#endif
```

5.7 Файл cl_4.cpp

Листинг 7 – cl_4.cpp

```
#include "cl_4.h"
cl_4::cl_4(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.8 Файл cl_4.h

Листинг 8 – 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* p_head, string s_object_name);
};
#endif
```

5.9 Файл cl_5.cpp

Листинг 9 – cl_5.cpp

```
#include "cl_5.h"
cl_5::cl_5(cl_base*     p_head,     string     s_object_name):cl_base(p_head,
s_object_name){
}
```

5.10 Файл cl_5.h

Листинг 10 – 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* p_head, string s_object_name);
};
#endif
```

5.11 Файл cl_application.cpp

Листинг 11 – cl_application.cpp

```
#include "cl_application.h"
cl_application::cl_application(cl_base*
p_head_object):cl_base(p_head_object){}
void cl_application::Tree(){
   string head, coord;
   cl_base* p_head=this;
   cl_base* p_sub=nullptr;
   int i_class, i_state;
   cin>>head;
   Change_name(head);
   //Ввод иерархии
   while(true){
      cin>>head;
      if (head=="endtree"){
         break;
      }
      cin>>coord>> i_class;
      p_head = this->find_obj_bc(head);
      if(!p_head){
         cout<<"Object tree"<<endl;
         Out_fc();
         cout<<endl<<"The head object "<<head<<" is not found";
         cl_base* p_root=this;
         while(p_root->Get_baseptr()){
            p_root=p_root->Get_baseptr();
         }
         p_root->Change_name("EEERROR");
         return;
      }
         /*
         if(this->Find_global(head.substr(1))){
            p_head=this->Find_global(head.substr(1));
         }
         else {
            Out_fc();
            cout<<endl<<"The head object "<<coord<<" is not found";
            return 1;
         }
         */
      /*
```

```
            if(p_sub!=nullptr && head==p_sub->Get_name()){
               p_head=p_sub;
            }
            if(head==p_head->Get_name() && p_head->Get_ptr(sub)==nullptr){
               p_sub=new cl_1 (p_head,sub);
            }
            */
            if (p_head && !p_head->Get_ptr(coord)){
               switch (i_class)
               {
                  case 2:
                     p_sub=new cl_1(p_head,coord);
                     break;
                  case 3:
                     p_sub=new cl_2(p_head,coord);
                     break;
                  case 4:
                     p_sub=new cl_3(p_head,coord);
                     break;
                  case 5:
                     p_sub=new cl_4(p_head,coord);
                     break;
                  case 6:
                     p_sub=new cl_5(p_head,coord);
                     break;
                  default:
                     break;
               }
            }
            else cout<<endl<<coord<<"    Dubbing the name of subordinate objects";
         }

         /*
         while(cin>>head>>i_state){
            p_head = Find_global(head);
            if(p_head!=nullptr) {p_head->can_off(i_state);}
         }
         */
         //cout<<"Статусы заданы\n";

      }
      /*
      void cl_application::Tree(){
         string head, sub;
         cl_base* p_head=this;
         cl_base* p_sub=nullptr;
         int i_class, i_state;
         cin>>head;
         Change_name(head);
         //Ввод иерархии
         while(true){
            cin>>head;
            if (head=="endtree"){
               break;
            }
```

```
            cin>> sub >> i_class;
            p_head=Find_global(head);

            if(p_sub!=nullptr && head==p_sub->Get_name()){
               p_head=p_sub;
            }
            if(head==p_head->Get_name() && p_head->Get_ptr(sub)==nullptr){
               p_sub=new cl_1 (p_head,sub);
            }

            if (p_head&&!p_head->Get_ptr(sub)){
               switch (i_class)
               {
                  case 1:
                     p_sub=new cl_1(p_head,sub);
                     break;
                  case 2:
                     p_sub=new cl_2(p_head,sub);
                     break;
                  case 3:
                     p_sub=new cl_3(p_head,sub);
                     break;
                  case 4:
                     p_sub=new cl_4(p_head,sub);
                     break;
                  case 5:
                     p_sub=new cl_5(p_head,sub);
                     break;
                  default:
                     break;
               }
            }
         }
         //Установка состояний объекта
         while(cin>>head>>i_state){
            p_head = Find_global(head);
            if(p_head!=nullptr) {p_head->can_off(i_state);}
         }
         //cout<<"Статусы заданы\n";
      }
      */
      /*
      void cl_application::Tree_1(){
         string head, sub;
         cl_base* p_head=this;
         cl_base* p_sub=nullptr;
         cin>>head;
         Change_name(head);
         while(true){
            cin>>head>>sub;
            if (head==sub){
               break;
            }
            if(p_sub!=nullptr && head==p_sub->Get_name()){
               p_head=p_sub;
```

```
         }
         if(head==p_head->Get_name() && p_head->Get_ptr(sub)==nullptr){
            p_sub=new cl_1 (p_head,sub);
         }
      }
   }
   */
   int cl_application::Start(){
      cl_base* p_root=this;
      while(p_root->Get_baseptr()){
         p_root=p_root->Get_baseptr();
      }
      if(p_root->Get_name()=="EEERROR") return 1;
      cout<<"Object tree"<<endl;
      Out_fc();
      //cout<<endl<<"The tree of objects and their readiness"<<endl;
      //Out();
      cl_base* cur_ob=p_root;
      //Команды
      while(true){
         string command, coord;
         cin>>command;
         if(command=="END") break;
         cin>>coord;
         if (command=="SET"){
            if(cur_ob->find_obj_bc(coord)){
               cur_ob=cur_ob->find_obj_bc(coord);
               cout<<endl<<"Object is set: "<<cur_ob->Get_name();
            }
            else cout<<endl<<"The object was not found in specified coordinate:
   "<<coord;
         }
         if (command=="FIND"){
            cout<<endl<<coord<<"     ";
            if(cur_ob->find_obj_bc(coord)){
               cout<<"Object name: "<<cur_ob->find_obj_bc(coord)->Get_name();
            }
            else cout<<"Object is not found";
         }
         if (command=="MOVE"){
            cl_base* ob = cur_ob->find_obj_bc(coord);
            if(!ob) cout<<endl<<coord<<"     Head object is not found";
            else  if(ob->Get_baseptr()  !=  cur_ob->Get_baseptr()  &&  cur_ob-
   >Find_current(ob->Get_name())) cout<<endl<<coord<<"      Redefining the head
   object failed";
            else   if   (ob->Get_ptr(cur_ob->Get_name()))   cout<<endl<<coord<<"
   Dubbing the names of subordinate objects";
            else      if(cur_ob->Move_head(ob)==false)      cout<<endl<<coord<<"
   Redefining the head object failed";
         }
         if (command=="DELETE"){
            if(cur_ob->Get_ptr(coord)){
               vector <string> text;
               string text1;
               cl_base* h = cur_ob->Get_ptr(coord)->Get_baseptr();
```

```
                cur_ob->Del_obj(coord);
                while(h){
                   if(h->Get_name()!=p_root->Get_name()){
                      text.push_back(h->Get_name());
                   }
                   h=h->Get_baseptr();
                }
                for (int i = text.size()-1;i>=0;i--) text1+="/"+text[i];
                cout<<endl<<"The object "<<text1+"/"+coord<<" has been deleted";
             }
          }
       }
       cout<<endl<<"Current object hierarchy tree"<<endl;
       Out_fc();
       return 0;


    }
```

5.12 Файл cl_application.h

Листинг 12 – cl_application.h

```
    #ifndef __CL_APPLICATION__H
    #define __CL_APPLICATION__H
    #include "cl_base.h"
    #include "cl_1.h"
    #include "cl_2.h"
    #include "cl_3.h"
    #include "cl_4.h"
    #include "cl_5.h"
    class cl_application:public cl_base{
       public:
       cl_application(cl_base* p_head_object);
       void Tree();
       //void Tree_1();
       int Start();
    };
    #endif
```

5.13 Файл cl_base.cpp

Листинг 13 – cl_base.cpp

```
    #include "cl_base.h"
```

```
#include <stack>
using namespace std;
cl_base::cl_base(cl_base* p_head_object, string s_object_name){
   this -> s_object_name = s_object_name;
   this -> p_head_object = p_head_object;
   if (p_head_object!=nullptr){
      p_head_object->subordinate_objects.push_back(this);
   }
}
string cl_base::Get_name(){
   return s_object_name;
}//Получение имени объекта
bool cl_base::Change_name(string name){
   if(Get_baseptr()!=nullptr){
      for (int i=0; i<p_head_object->subordinate_objects.size();i++){
         if (p_head_object->subordinate_objects[i]->Get_name()==name){
            return false;
         }
      }
   }
   this->s_object_name=name;
   return true;
}//Поменять имя объекта
cl_base* cl_base::Get_ptr(string s_object_name){
   for (int i=0; i<subordinate_objects.size();i++){
      if (subordinate_objects[i]->Get_name()==s_object_name){
         return subordinate_objects[i];
      }
   }
   return nullptr;
}//Получение указателя на объект
cl_base* cl_base::Get_baseptr(){
   return p_head_object;
}//Получение головного указателя
//Статус
/*
int cl_base::Get_status(){
   return status;
}
*/
/*Сет статус
void cl_base::Set_status(int i){
   this->status=i;
}
*/
//Статус
void cl_base::can_off(int status1){
   if (status1==0){
      status=0;
      for (auto sub:subordinate_objects){
         sub->can_off(0);
      }
      //cout<<"Поставил фальш\n";
   }
   if (Get_baseptr()&&(Get_baseptr()->status==0)){
```

```
            return;
        }
        else if (status1!=0){
            status=1;
        }
    }

    void cl_base::Out_fc(int probel){
        cout<<Get_name();
        if(subordinate_objects.size()!=0){
            for (auto sub:subordinate_objects){
                cout<<endl;
                for(int i =0; i<probel;i++) cout<<" ";
                sub->Out_fc(probel+4);
            }
        }
    }//Вывод дерева
    /*
    void cl_base::Out(int probel){
        string ready="is ready";
        string nready="is not ready";
        cout<<"    "<<this->Get_name();
        if(this->Get_status()!=0){
            cout<<ready<<endl;
        }
        else{cout<<nready<<endl;}
    }*/
    //Вывод дерева
    void cl_base::Out(int probel){
        cout<< Get_name();
        if (status==0)cout<<" is not ready";
        else if (status!=0){cout<<" is ready";}
        if (subordinate_objects.size()!=0){
            for(int i=0; i<subordinate_objects.size();i++){
                cout<<endl;
                for(int i =0; i<probel;i++) cout<<" ";
                subordinate_objects[i]->Out(probel+4);
            }
        }
    }//Вывод дерева

    //Поиск не на основе очереди, был на ней, но не судьба
    cl_base* cl_base::Find_current(string name){
        if (Get_name() == name)
        {
            //cout<<"Нашел\n";
            return this;
        }
        for (auto sub : subordinate_objects)
        {
            cl_base* p_sub = sub->Find_current(name);
            if (p_sub)
            {
                return p_sub;
            }
```

```
   }
   return nullptr;
}
/*
int cl_base::Count(string name){
   int count=0;
   if (Get_name()==name) count++;
   for(auto sub:subordinate_objects)count+=sub->Count(name);
   return count;
}
*/

cl_base* cl_base::Find_global(string name){
   cl_base* p_root=this;
   while(p_root->Get_baseptr()){
      p_root=p_root->Get_baseptr();
   }
   cl_base* p_found = p_root;
   int count =0;
   stack <cl_base*> stack;
   stack.push(p_found);
   while(!stack.empty()){
      cl_base * cur = stack.top();
      stack.pop();
      if (cur->Get_name()==name) count++;
      for (auto sub:cur->subordinate_objects) stack.push(sub);
   }
   if (count!=1) return nullptr;
   //cout<<"Нащел родича \n";
   //cout<<p_found->Find_current(name)->Get_name();
   return p_found->Find_current(name);
}
/*
cl_base* cl_base::get_root(){
   cl_base* p_root=this;
   while(p_root->Get_baseptr()){
      p_root=p_root->Get_baseptr();
   }
   return p_root;
}
*/
//Метод поиска обхекта по координате
cl_base* cl_base::find_obj_bc(string coord){
   string s_name;
   int index;
   cl_base* ob = nullptr;
   cl_base* p_root=this;
   while(p_root->Get_baseptr()){
      p_root=p_root->Get_baseptr();
   }
   //головной объект
   if(coord=="/"){
      return p_root;
   }
   //текущий
```

```
        if(coord=="."){
           return this;
        }
        //Поиск от корня по имени
        if(coord[0]=='/' && coord[1]=='/'){
           s_name=coord.substr(2);
           return this->Find_global(s_name);
        }
        //Поиск от текущего по имени
        if(coord[0]=='.'){
           s_name=coord.substr(1);
           return this->Find_current(s_name);
        }
        index=coord.find("/",1);
        //Абсолютный путь
        if(coord[0]=='/'){
           if (index!=-1){
              s_name= coord.substr(1,index-1);
              ob=p_root->Get_ptr(s_name);
              if(ob!=nullptr){
                 return ob->find_obj_bc(coord.substr(index+1));
              }
              else return ob;
           }
           else {
              //cout<<"\n ИНДЕКС: "<<index<<endl;
              s_name= coord.substr(1);
              return p_root->Get_ptr(s_name);
           }
        }
        //Относительный путь
        if(coord[0]!='/'){
           if (index!=-1){
              s_name= coord.substr(0,index);
              ob=this->Get_ptr(s_name);
              if(ob!=nullptr){
                 return ob->find_obj_bc(coord.substr(index+1));
              }
              else return ob;
           }
           else{
              s_name=coord;
              return this->Get_ptr(s_name);
           }
        }
        return nullptr;
     }
     //Метод переопределения головного объекта
     bool cl_base::Move_head(cl_base* h){
        if (this->Get_baseptr()){
           /*
           for (auto sub:h->subordinate_objects){
              if (sub==h){
                 cout<<"    Dubbing the names of subordinate objects";
                 return false;
```

```
            }
         }
         */
         for (int i=0;i<this->Get_baseptr()->subordinate_objects.size();i++){
            if(this->Get_baseptr()->subordinate_objects[i]==this){
               Get_baseptr()->subordinate_objects.erase(this->Get_baseptr()-
>subordinate_objects.begin()+i);
               break;
            }
         }
         this->p_head_object=h;
         h->subordinate_objects.push_back(this);
         cout<<endl<<"New head object: "<<h->Get_name();
         return true;
      }
      return false;
   }
   //Метод удаления подчиненного объекта
   void cl_base::Del_obj(string name){
      cl_base* p = Get_ptr(name);
      /*
      if (p->subordinate_objects.size()!=0){
         return false;
      }
      */
      int i = 0;
      for(auto sub:subordinate_objects){
         if (sub==p){
            subordinate_objects.erase(subordinate_objects.begin()+i);
            delete p;
            return;
         }
         i++;
      }
      /*
      string text;
      cl_base* h = p->Get_baseptr();
      while(h){
         text+="/"+h->Get_name();
         h=h->Get_baseptr();
      }
      cout<<endl<<"The object "<< text+"/"+name<<" has been deleted";
      */
   }

   cl_base::~cl_base(){
      for (int i=0; i<subordinate_objects.size();i++){
         delete subordinate_objects[i];
      }
   }
```

5.14 Файл cl_base.h

Листинг 14 – cl_base.h

```
#ifndef __CL_BASE__H
#define __CL_BASE__H
#include <iostream>
#include <string>
#include <vector>
#include <queue>
using namespace std;
class cl_base{
   private:
   int status=0;
   string s_object_name;
   cl_base * p_head_object;
   vector <cl_base*> subordinate_objects;
   public:

   cl_base(cl_base* p_head_object,string s_object_name="Base_object");
   bool Change_name(string name);
   string Get_name();
   cl_base* Get_baseptr();
   void Out(int probel=4);
   void Out_fc(int probel=4);
   void can_off(int status);
   //void Out1();
   //int Get_status();
   //void Set_status(int i);
   //int Count(string name);
   //cl_base* get_root();
   bool Move_head(cl_base*);
   void Del_obj(string name);
   cl_base* Find_current(string name);
   cl_base* Find_global(string name);
   cl_base* Get_ptr(string s_object_name);
   cl_base* find_obj_bc(string coord);
   //cl_base* Search(string name);
   ~cl_base();
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

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

```
#include "cl_base.h"
#include "cl_application.h"
int main()
{
   cl_application  ob_cl_application ( nullptr );
   ob_cl_application.Tree();
   return ob_cl_application.Start();
   return 1;
   /*
   string a,b;
   int c;
   cin>>a;
   cin>>b>>c;
   cout<<a<<" "<<b<<" "<<c<<endl;
   cout<<a[0]<<endl;
   */
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 8.

Таблица 8 – Результат тестирования программы

Входные данные	**Ожидаемые выходные данные**	**Фактические выходные данные**
root endtree END	Object tree root Current object hierarchy tree root	Object tree root Current object hierarchy tree root

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).