МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**« МИРЭА Российский технологический университет»**

**РТУ МИРЭА**

Институт Информационных технологий

Кафедра Вычислительной техники

# УЧЕБНОЕ ЗАДАНИЕ

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

## « Задание 4_1_1 »

| | | |
|---|---|---|
| С тудент группы | ИКБО-07-19 | Ле Д.. |
| Руководитель практики | Ассистент | Боронников А.С. |
| Работа представлена | «___»_____ 2020 г. | _____ |
| | | *(подпись студента)* |
| Оценка | | _____ |
| | | *(подпись руководителя)* |

Москва 2020

# Постановка задачи

Для организации иерархического построения объектов необходимо разработать базовый класс, который содержит функционал и свойства для построения иерархии объектов. В последующем, в приложениях использовать этот класс как базовый для всех создаваемых классов. Это позволит включать любой объект в состав дерева иерархии объектов.

Создать базовый класс со следующими элементами:

Свойства:

- наименование объекта (строкового типа);
- указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно 0);
- список указателей на объекты подчиненных к текущему объекту в дереве иерархии.

Функционал:

- параметризированный конструктор с параметром указателя на головной объект в дереве иерархии;
- параметризированный конструктор с параметром указателя на головной объект в дереве иерархии и наименованием объекта;
- метод задания имени объекта;
- метод получения имени объекта;
- метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз.

Написать программу, которая последовательно строит дерево иерархии объектов, слева направо и сверху вниз. Переход на новый уровень происходит от правого (последнего) объекта предыдущего уровня. Для построения дерева использовать объекты двух производных классов, наследуемых от базового. Каждый объект имеет уникальное имя.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

## Описание входных данных

**Первая строка:**
«имя корневого объекта»
Создается корневой объект.
**Вторая строка и последующие строки:**
«имя головного объекта»«имя подчиненного объекта»
Создается подчиненный объект и добавляется в иерархическое дерево.

Если «имя головного объекта» равняется «имени подчиненного объекта», то новый объект не создается и построение дерева объектов завершается.

**Примерввода**

Object_root
Object_root Object_1
Object_root Object_2
Object_root Object_3
Object_3 Object_4
Object_3 Object_5
Object_6 Object_6

Дерево объектов, которое будет построено по данному примеру:

Object_root

Object_1
Object_2
Object_3
Object_4

Object_5

## Описание выходных данных

**Первая строка:**
«имя корневого объекта»
**Вторая строка и последующие строки** имена головного и подчиненных объектов

очередного уровня разделенных двумя пробелами.
«имя головного объекта»«имя подчиненного объекта»[[ «имя подчиненного объекта»] …….]

**Пример вывода**

Object_root
Object_root Object_1 Object_2 Object_3
Object_3 Object_4 Object_5

## Метод решения

Используя потоки Ввода/Вывода - cin/cout

Используя void bild_tree_objects() для реализовать построения исходного дерева иерархии.

Используя void show_object_state() для показать состояние объекта.

Используя void show_state_next(cl_base* ob_parent) для показать следующий состояние.

Используя int exec_app() для применять.

## Описание алгоритма

cl_application(string name)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | set_object_name(name); | Ø |

void cl_application::bild_tree_objects()

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | cl_2* ob_2; | 2 |
| 2 | | string nameParent, nameChild; | 3 |

| | | | |
|---|---|---|---|
| 3 | while (true) | | 4 |
| 4 | | cin >> nameParent, nameChild; | 5 |
| 5 | if (nameParent != nameChild) | | 6 |
| | else | break; | Ø |
| 6 | if (get_object_name(this) == nameParent) | ob_2 = new cl_2((cl_base*)this); | 7 |
| | else | add_new_child(this, nameParent, nameChild); | 3 |
| 7 | | ob_2->set_object_name(nameChild); | 3 |

void cl_application::add_new_child(cl_base* ob_parent, string nameParent, string nameChild)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | cl_2* ob_2; | 2 |
| 2 | for (size_t i = 0; i < ob_parent->children.size(); i++) | | 3 |
| | i = ob_parent->children.size() | | 6 |
| 3 | if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent) | ob_2 = new cl_2((cl_base*)ob_parent->children.at(i)); | 4 |
| | else | | 2 |
| 4 | | ob_2->set_object_name(nameChild); | 5 |
| 5 | | return; | Ø |
| 6 | | ob_parent->it_child = ob_parent->children.begin(); | 7 |
| 7 | while (ob_parent->it_child != ob_parent->children.end()) | add_new_child((*(ob_parent->it_child)), nameParent, nameChild, state, selectFamily); | 8 |
| | ob_parent->it_child == ob_parent->children.end() | | Ø |
| 8 | | ob_parent->it_child++; | 7 |

int cl_application::exec_app()

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | show_object_state(); | 2 |
| 2 | | return 0; | Ø |

void cl_application::show_object_state()

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1 | | cout << get_object_name(this); | 2 |
| 2 | | show_state_next(this); | Ø |

void cl_application::show_child(cl_base* ob_parent)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | cout << " " << get_object_name(ob_parent); | Ø |

void cl_application::show_state_next(cl_base * ob_parent)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | if (ob_parent->children.size() == 0) | return; | Ø |
| | else | | 2 |
| 2 | | cout << endl; | 3 |
| 3 | | cout << get_object_name(ob_parent); | 4 |
| 4 | | ob_parent->it_child = ob_parent->children.begin(); | 5 |
| 5 | while (ob_parent->it_child != ob_parent->children.end()) | show_child((*(ob_parent->it_child))); | 6 |
| | ob_parent->it_child == ob_parent->children.end() | | 7 |
| 6 | | ob_parent->it_child++; | 5 |
| 7 | | ob_parent->it_child = ob_parent->children.begin(); | 8 |
| 8 | while (ob_parent->it_child != ob_parent->children.end()) | show_state_next((*(ob_parent->it_child))); | 9 |
| | ob_parent->it_child == ob_parent->children.end() | | Ø |
| 9 | | ob_parent->it_child++; | 8 |

cl_base(cl_base* p_parent)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | set_object_name ( "cl_base" ); | 2 |
| 2 | if(p_parent) | | 3 |
| | else | | 4 |
| 3 | | p_parent->add_child(this); | Ø |
| 4 | | this->p_parent = 0; | Ø |

void cl_base::set_object_name(string object_name)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | this -> object_name = object_name; | Ø |

string cl_base::get_object_name(cl_base* p_parent)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | return p_parent->object_name; | Ø |

void cl_base::set_parent(cl_base* p_parent)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | if (p_parent) | this ->p_parent = p_parent; | 2 |
| | else | | Ø |
| 2 | | p_parent->add_child(this); | Ø |

void cl_base::add_child(cl_base* p_child)

| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | children.push_back ( p_child ); | Ø |

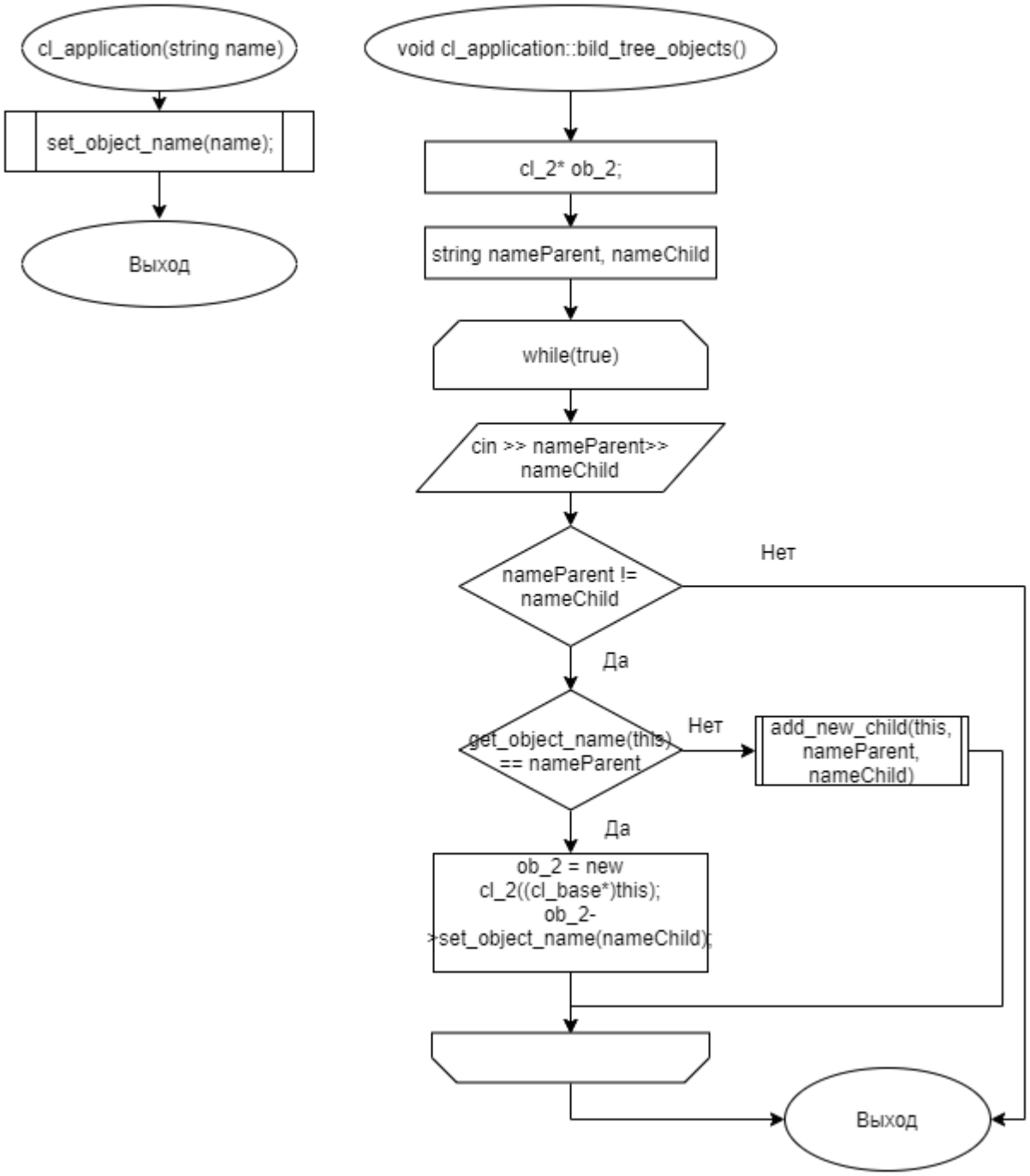
cl_base * cl_base :: get_child ( string object_name )

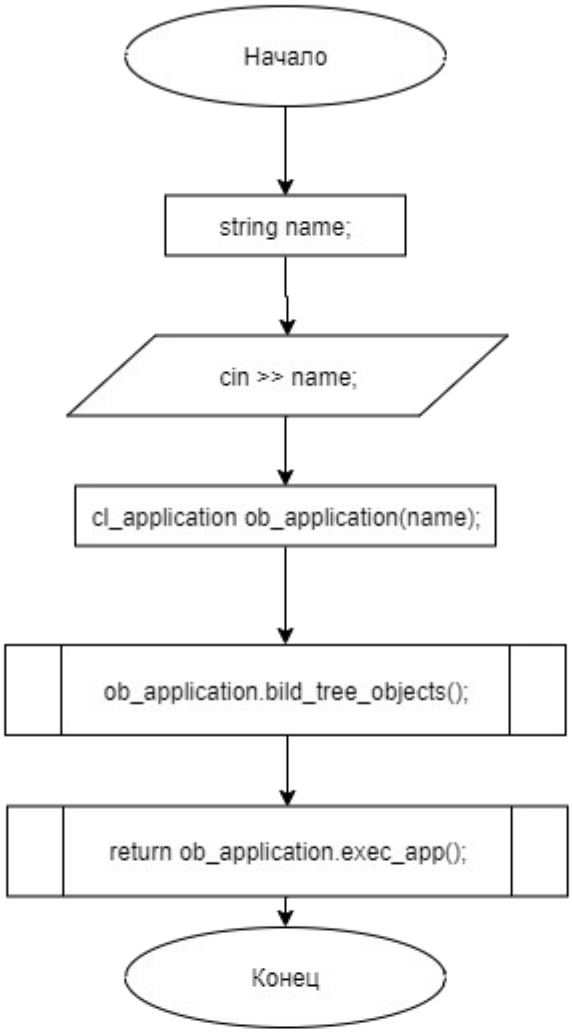
| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | if (children.size() == 0) | return 0; | Ø |
| | else | | 2 |
| 2 | | it_child = children.begin(); | 3 |
| 3 | while (it_child != children.end()) | | 4 |
| | it_child == children.end() | | 6 |
| 4 | if (get_object_name((*it_child)) == object_name) | return (*it_child); | 5 |
| | else | | 5 |
| 5 | | it_child++; | 4 |
| 6 | | return 0; | Ø |

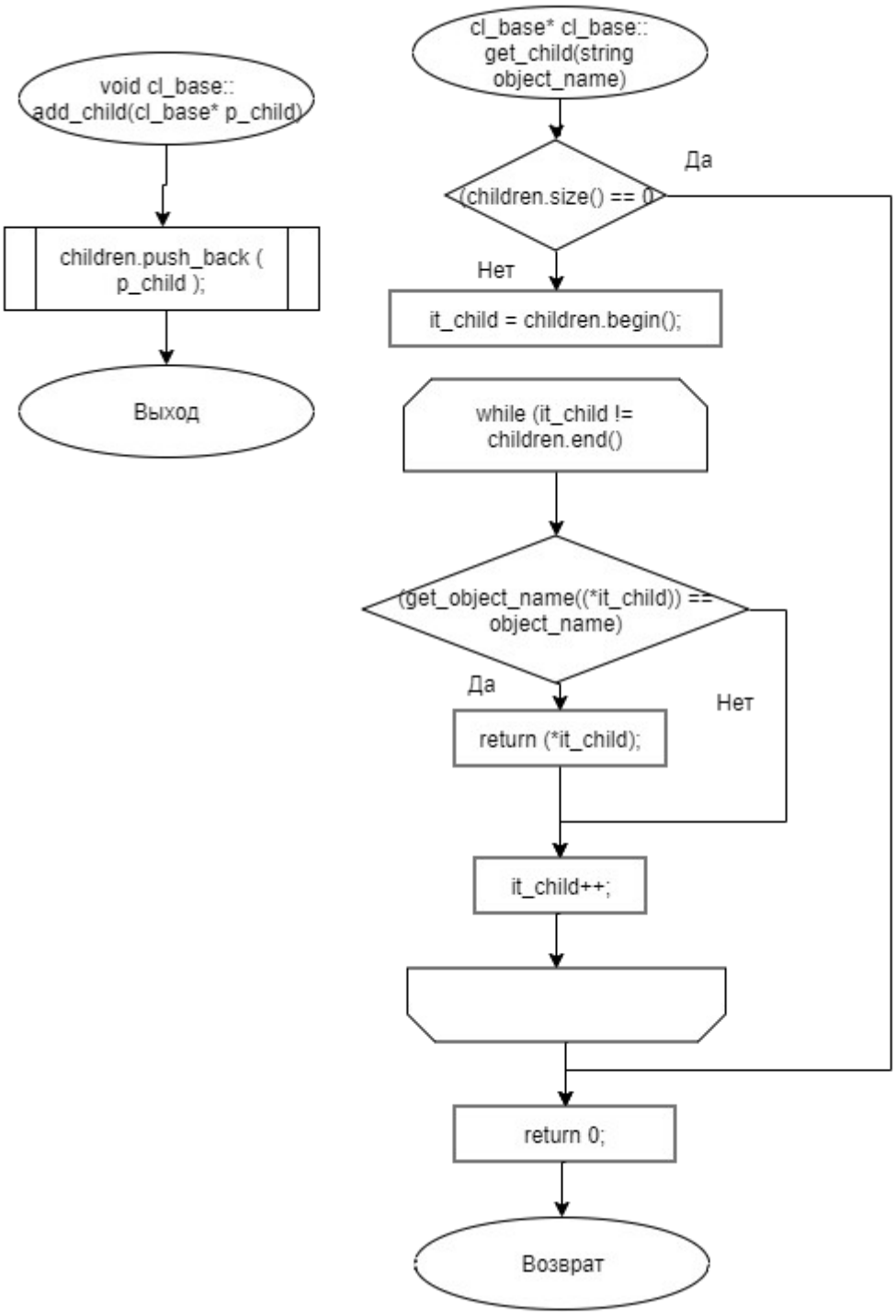
int main()

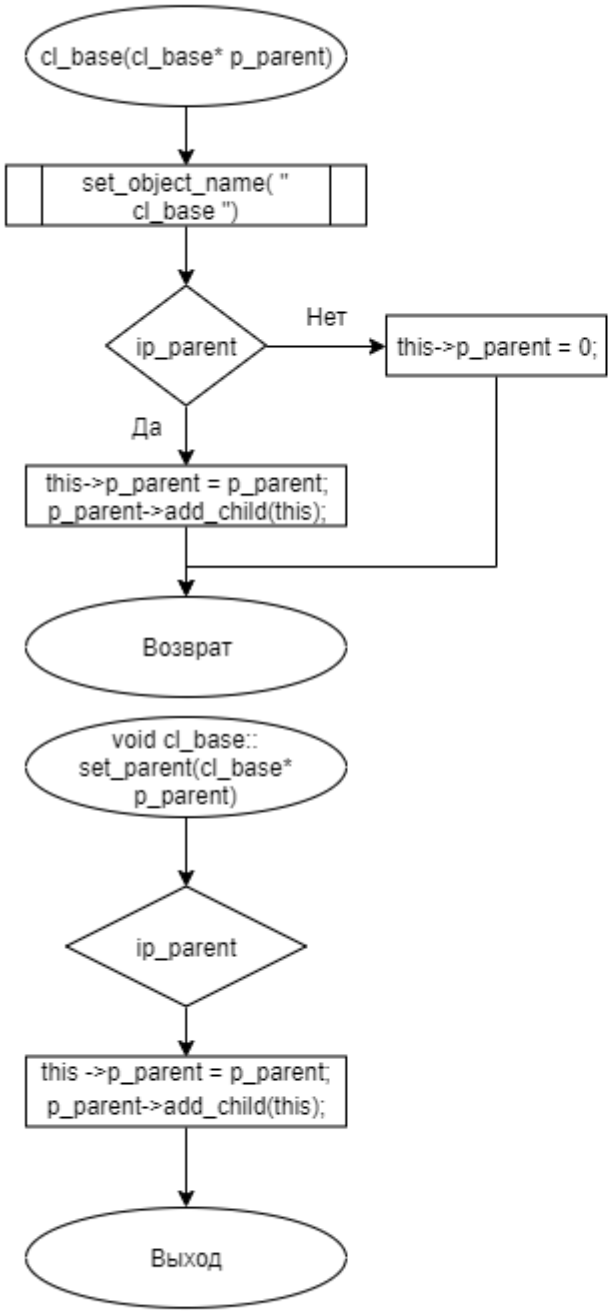
| № шага | Предикат | Действе | № перехода |
|---|---|---|---|
| 1 | | string name; | 2 |
| 2 | | cin >> name; | 3 |
| 3 | | cl_application ob_application(name); | 4 |
| 4 | | ob_application.bild_tree_objects(); | 5 |
| 5 | | return ob_application.exec_app(); | Ø |

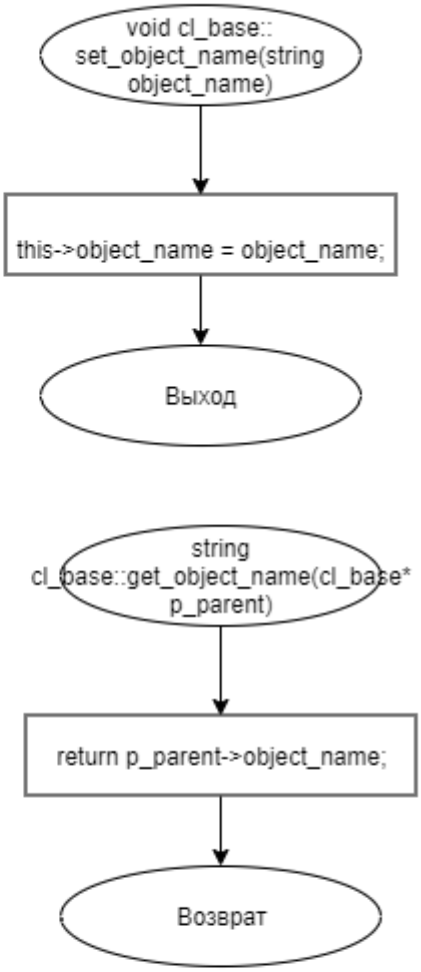
## Блок-схема алгоритма

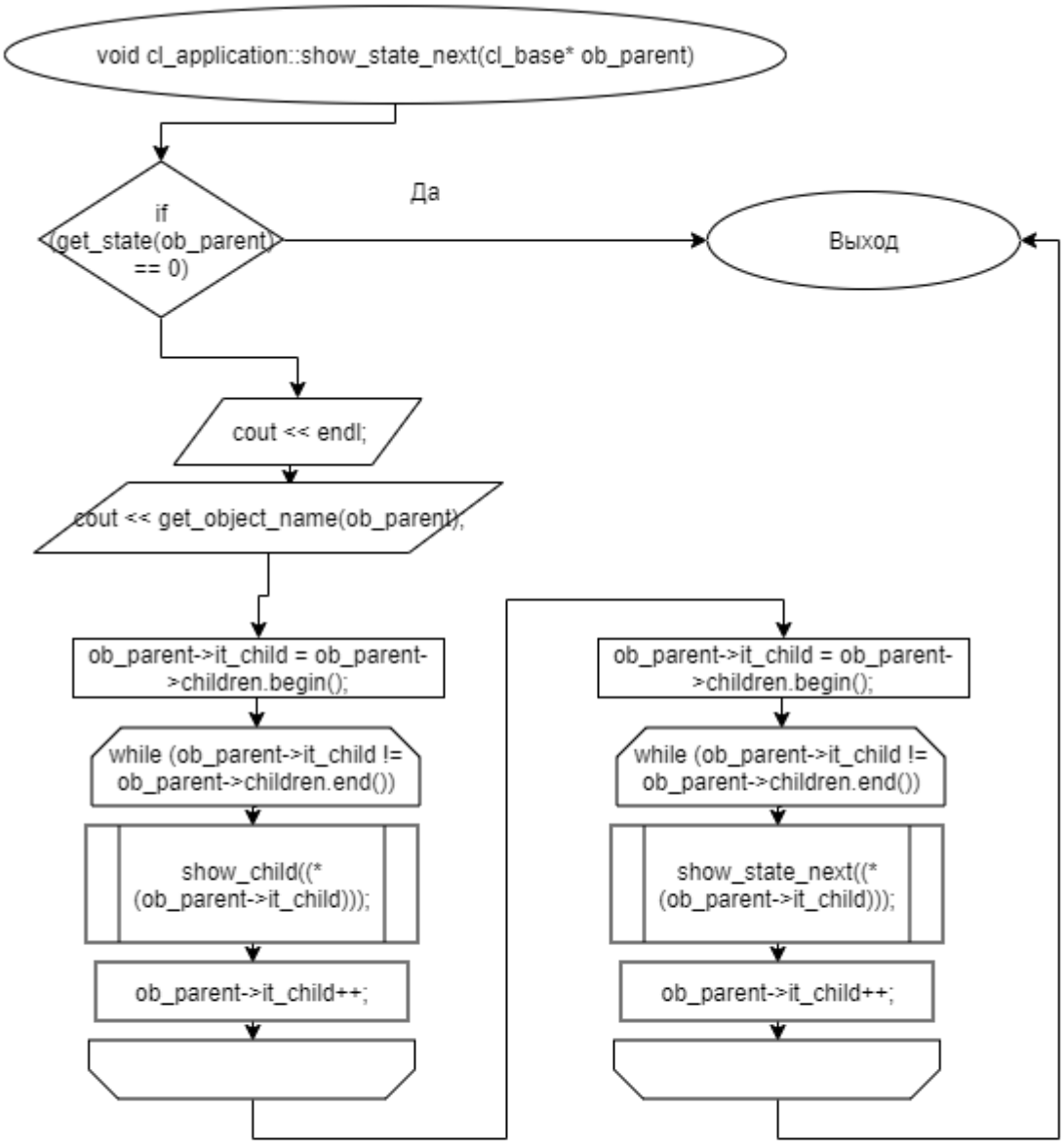


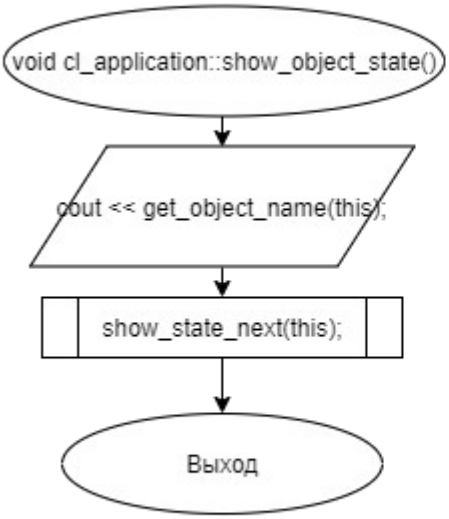

Начало
string name;
cin >> name;
cl_application ob_application(name);
ob_application.bild_tree_objects();
return ob_application.exec_app();
Конец

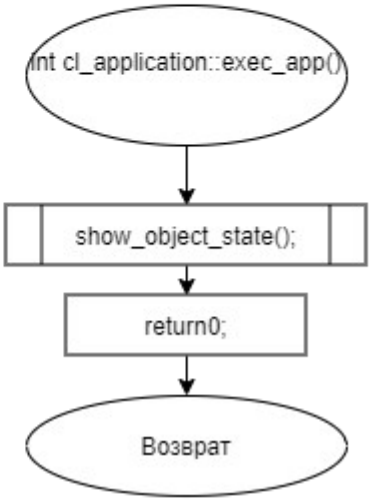

void cl_base::
add_child(cl_base* p_child)
children.push_back (
p_child );
Выход
cl_base* cl_base::
get_child(string
object_name)
(children.size() == 0)
Да
Нет
it_child = children.begin();
while (it_child !=
children.end()
(get_object_name((*it_child)) ==
object_name)
Да
Нет
return (*it_child);
it_child++;
return 0;
Возврат

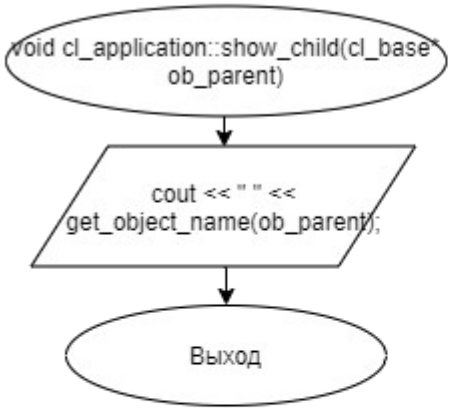

cl_base(cl_base* p_parent)
set_object_name( " cl_base ")
ip_parent
Нет
this->p_parent = 0;
Да
this->p_parent = p_parent;
p_parent->add_child(this);
Возврат
void cl_base:: set_parent(cl_base* p_parent)
ip_parent
this ->p_parent = p_parent;
p_parent->add_child(this);
Выход

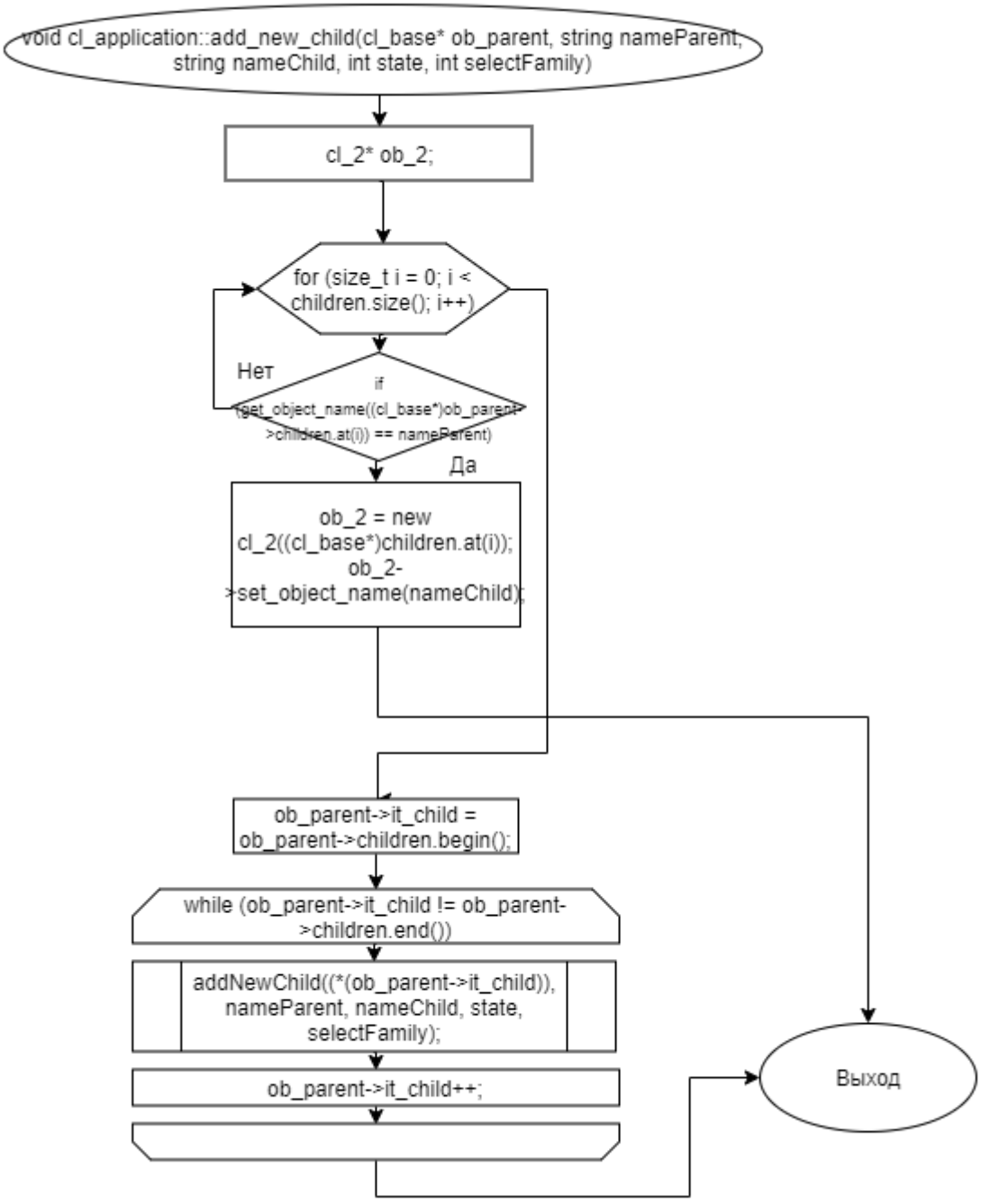

void cl_base:: set_object_name(string object_name)
this->object_name = object_name;
Выход
string cl_base::get_object_name(cl_base* p_parent)
return p_parent->object_name;
Возврат


void cl_application::show_state_next(cl_base* ob_parent)
if (get_state(ob_parent) == 0)
Да
Выход
cout << endl;
cout << get_object_name(ob_parent);
ob_parent->it_child = ob_parent->children.begin();
while (ob_parent->it_child != ob_parent->children.end())
show_child((*(ob_parent->it_child)));
ob_parent->it_child++;
ob_parent->it_child = ob_parent->children.begin();
while (ob_parent->it_child != ob_parent->children.end())
show_state_next((*(ob_parent->it_child)));
ob_parent->it_child++;


void cl_application::show_object_state()
cout << get_object_name(this);
show_state_next(this);
Выход


int cl_application::exec_app()
show_object_state();
return0;
Возврат


void cl_application::show_child(cl_base*
ob_parent)
cout << " " <<
get_object_name(ob_parent);
Выход




## Код программы

**Файл cl_2.cpp**

```
#include "cl_2.h"

cl_2::cl_2(cl_base* p_parent) : cl_base(p_parent){}
```

## Файл 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_parent = 0);
};

#endif // CL_2_H
```

## Файл cl_application.cpp

```
#include "cl_application.h"

using namespace std;

cl_application :: cl_application (string name ){
        set_object_name (name);
}

void cl_application :: bild_tree_objects (){

        cl_2* ob_2;
        string nameParent, nameChild;

        while (true) {
                cin >> nameParent;
                cin >> nameChild;
                if (nameParent != nameChild) {
                        if (get_object_name(this) == nameParent) {
```

```
                                ob_2 = new cl_2((cl_base*)this);
                                ob_2->set_object_name(nameChild);

                        }
                        else {
                                add_new_child(this, nameParent, nameChild);
                        }
                }
                else {
                        break;
                }
        }
}

void cl_application::add_new_child(cl_base* ob_parent, string nameParent,
string nameChild) {
        cl_2* ob_2;
        for (size_t i = 0; i < ob_parent->children.size(); i++) {
                if (get_object_name((cl_base*)ob_parent->children.at(i)) ==
nameParent) {
                        ob_2 = new cl_2((cl_base*)ob_parent->children.at(i));
                        ob_2->set_object_name(nameChild);
                        return;
                }
        }
        ob_parent->it_child = ob_parent->children.begin();
        while (ob_parent->it_child != ob_parent->children.end()) {
                        add_new_child((*(ob_parent->it_child)), nameParent,
nameChild);
                        ob_parent->it_child++;
        }
}


        int cl_application::exec_app() {
                show_object_state();
                return 0;
        }

        void cl_application::show_object_state() {
                cout << get_object_name(this);
                show_state_next(this);
        }

        void cl_application::show_child(cl_base* ob_parent) {
                cout << " " << get_object_name(ob_parent);
        }

        void cl_application::show_state_next(cl_base * ob_parent) {

                if (ob_parent->children.size() == 0)
                        return;
                cout << endl;
                cout << get_object_name(ob_parent);

                ob_parent->it_child = ob_parent->children.begin();
                while (ob_parent->it_child != ob_parent->children.end()) {
                        show_child((*(ob_parent->it_child)));
                        ob_parent->it_child++;
                }
```

```
                ob_parent->it_child = ob_parent->children.begin();
                while (ob_parent->it_child != ob_parent->children.end()) {
                        show_state_next((*(ob_parent->it_child)));
                        ob_parent->it_child++;
                }
        }
```

## Файл cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H

#include "cl_2.h"

class cl_application : public cl_base {

public:

cl_application(string name);

void bild_tree_objects();
int exec_app();
void show_child(cl_base* ob_parent);
void show_object_state();
void add_new_child(cl_base* ob_parent, string nameParent, string nameChild);

private:

void show_state_next(cl_base* ob_parent);
};

#endif // CL_APPLICATION_H
```

## Файл cl_base.cpp

```
#include "cl_base.h"
cl_base :: cl_base ( cl_base * p_parent )
{

        set_object_name ( "cl_base" );
        if ( p_parent ) {
                this -> p_parent = p_parent;
                p_parent -> add_child ( this );
```

```
        }else {
                this ->p_parent = 0;
        }
}
void cl_base :: set_object_name ( string object_name ){

        this -> object_name = object_name;
}

string cl_base::get_object_name(cl_base* p_parent) {
        return p_parent->object_name;
}

void cl_base :: set_parent ( cl_base * p_parent ){
        if ( p_parent ) {
                this -> p_parent = p_parent;
                p_parent -> add_child ( this );
        }
}
void cl_base :: add_child ( cl_base * p_child ){
        children.push_back ( p_child );
}


cl_base * cl_base :: get_child ( string object_name ){

        if ( children.size ( ) == 0 ) return 0;
        it_child = children.begin ( );
        while ( it_child != children.end ( ) ) {
                if (get_object_name (( * it_child ) ) == object_name ) {
                        return ( * it_child );
                }
                it_child ++;
        }
        return 0;
}
```

## Файл cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H


#include <iostream>
#include <string>
#include <vector>

using namespace std;

class cl_base {

public:

cl_base(cl_base* p_parent = 0);
```

```
void set_object_name(string object_name);
string get_object_name(cl_base* p_parent);
void set_parent(cl_base* p_parent);
void add_child(cl_base* p_child);
cl_base* get_child(string object_name);

vector < cl_base* > children;
vector < cl_base* > ::iterator it_child;

private:

string object_name;
cl_base* p_parent;
};

#endif // CL_BASE_H
```

## Файл main.cpp

```
#include <iostream>
using namespace std;

#include "cl_application.h"

int main()
{
string name;
cin >> name;

cl_application ob_application(name);

ob_application.bild_tree_objects();

return ob_application.exec_app();
}
```

## Тестирование

| Входные данные | Ожидаемые выходные данные | Фактические выходные данные |
| --- | --- | --- |
| Object_root Object_root Object_1 Object_root Object_2 Object_root Object_3 Object_3 Object_4 Object_3 Object_5 Object_6 Object_6 | Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5 | Object_root Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5 |