МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование»

Наименование задачи:

« КЛ_3_2 Вывод иерархического дерева »

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

Вывод иерархического дерева объектов на консоль

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

Построить модель иерархической системы. Реализовать 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).

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в контрольной работе № 1.

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

Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Структура данных для ввода согласно изложенному в фрагменте методического указания в контрольной работе № 1.

Описание выходных данных

Вывести иерархию объектов в следующем виде: Object tree «Наименование корневого объекта» «Наименование объекта 1» «Наименование объекта 2» «Наименование объекта 3» . . . . . Отступ каждого уровня иерархии 4 позиции.

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
1		set_object_name(name);
2		set_state(1);

void cl_application::bild_tree_objects()

№ шага	Предикат	Действе
1		cl_2* ob_2;
2		cl_3* ob_3;
3		cl_4* ob_4; cl_5* ob_5; cl_6* ob_6;
4		string nameParent, nameChild;
5		int selectFamily, state;
6	while (true)	
7		cin >> nameParent;
8	if (nameParent == text_finish)	break;
	else	
9		cin >> nameChild >> selectFamily >> state;
10	if (selectFamily == 2)	
	else	
11	if (this->get_object_name() == nameParent)	
	else	
12		ob_2 = new cl_2((cl_base*)this);
13		ob_2->set_object_name(nameChild);
14		ob_2->set_state(state);
15		addNewChild(this, nameParent, nameChild, state,
16	if (selectFamily == 3)	
	else	
17	if (this->get_object_name() == nameParent)	
	else	
18		ob_3 = new cl_3((cl_base*)this);
19		ob_3->set_object_name(nameChild);
20		ob_3->set_state(state);
21		addNewChild(this, nameParent, nameChild, state,
22	if (selectFamily == 4)	

	else	
23	if (this->get_object_name() == nameParent)	
	else	
24		ob_4 = new cl_4((cl_base*)this);
25		ob_4->set_object_name(nameChild);
26		ob_4->set_state(state);
27		addNewChild(this, nameParent, nameChild, state,
28	if (selectFamily == 5)	
	else	
29	if (this->get_object_name() == nameParent)	
	else	
30		ob_5 = new cl_5((cl_base*)this);
31		ob_5->set_object_name(nameChild);
32		ob_5->set_state(state);
33		addNewChild(this, nameParent, nameChild, state,
34	if (selectFamily == 6)	
	else	
35	if (this->get_object_name() == nameParent)	
36	else	
37		ob_6 = new cl_6((cl_base*)this);
38		ob_6->set_object_name(nameChild);
39		ob_6->set_state(state);
40		addNewChild(this, nameParent, nameChild, state,
41		break;

void cl_application::addNewChild(cl_base* ob_parent, string nameParent, string nameChild, int state, int selectFamily)

№ шага	Предикат	Действе
1		cl_2* ob_2; cl_3* ob_3; cl_4* ob_4;
2	if (selectFamily == 2)	
	else	
3	for (size_t i = 0; i < ob_parent->children.size(); i++)	
	i = ob_parent->children.size()	
4	if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)	ob_2 = new cl_2((cl_base*)ob_parent->
	else	

5		ob_2->set_object_name(nameChild);
6	if (get_state((cl_base*)ob_parent->children.at(i)) >0)	ob_2->set_state(state);
	else	ob_2->set_state(0);
7		return;
8	if (selectFamily == 3)	
	else	
9	for (size_t i = 0; i < ob_parent->children.size(); i++)	
	i = ob_parent->children.size()	
10	if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)	ob_3 = new cl_3((cl_base*)ob_parent->
	else	
11		ob_3->set_object_name(nameChild);
12	if (get_state((cl_base*)ob_parent->children.at(i)) >0)	ob_3->set_state(state);
	else	ob_3->set_state(0);
13		return;
14	if (selectFamily == 4)	
	else	
15	for (size_t i = 0; i < ob_parent->children.size(); i++)	
	i = ob_parent->children.size()	
16	if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)	ob_4 = new cl_4((cl_base*)ob_parent->
	else	
17		ob_4->set_object_name(nameChild);
18	if (get_state((cl_base*)ob_parent->children.at(i)) >0)	ob_4->set_state(state);
	else	ob_4->set_state(0);
19		return;
20	if (selectFamily == 5)	
	else	
21	for (size_t i = 0; i < ob_parent->children.size(); i++)	
	i = ob_parent->children.size()	
22	if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)	ob_5 = new cl_5((cl_base*)ob_parent->
	else	
23		ob_5->set_object_name(nameChild);
24	if (get_state((cl_base*)ob_parent->children.at(i)) >0)	ob_5->set_state(state);
	else	ob_5->set_state(0);
25		return;
26	if (selectFamily == 6)	
	else	
27	for (size_t i = 0; i < ob_parent->children.size(); i++)	
	i = ob_parent->children.size()	

28	if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)	ob_6 = new cl_6((cl_base*)ob_parent->
	else	
29		ob_6->set_object_name(nameChild);
30	if (get_state((cl_base*)ob_parent->children.at(i)) >0)	ob_6->set_state(state);
	else	ob_6->set_state(0);
31		return;
32		ob_parent->it_child = ob_parent->child
33	while (ob_parent->it_child != ob_parent->children.end())	addNewChild((*(ob_parent->it_child)), state, selectFamily);
	ob_parent->it_child == ob_parent->children.end()	
34		ob_parent->it_child++;

int cl_application::exec_app()

№ шага	Предикат	Действе	N
1		show_object_state();	2
2		return 0;	Ø

void cl_application::show_object_state()

№ шага	Предикат	Действе№ перехода
1		show_state_next(this, 0);

void cl_application::show_state_next(cl_base* ob_parent, int i)

№ шага	Предикат	Действе
1	if (i == 0)	cout << endl << get_object_name(ob_parent)
	else	cout << endl << setw(4 * i) << " " << get_object_name(ob_parent);
2	if (ob_parent->children.size() == 0)	return;
	else	
3		ob_parent->it_child = ob_parent->children.be
4	while (ob_parent->it_child != ob_parent->children.end())	show_state_next((*(ob_parent->it_child)), i+
	ob_parent->it_child == ob_parent->children.end()	
5		ob_parent->it_child++;

cl_base(cl_base* p_parent)

№ шага	Предикат	Действе
1		set_object_name(" cl_base ");

2	if(p_parent)	this->p_parent = p_parent;
	else	
3		p_parent->add_child(this);
4		this->p_parent = 0;

void cl_base::set_object_name(string object_name)\

№ шага	Предикат	Действе
1		this->object_name = object_name;

string cl_base::get_object_name(cl_base* p_parent)

№ шага	Предикат	Действе
1		return p_parent->object_name;

void cl_base::set_parent(cl_base* p_parent)

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

void cl_base::add_child(cl_base* p_child)

№ шага	Предикат	Действе
1		children.push_back(p_child);

cl_base* cl_base::get_child(string object_name)

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

void cl_base::set_state(int c_state)

№ шага	Предикат	Действе	№ пе
1		this->c_state = c_state;	Ø

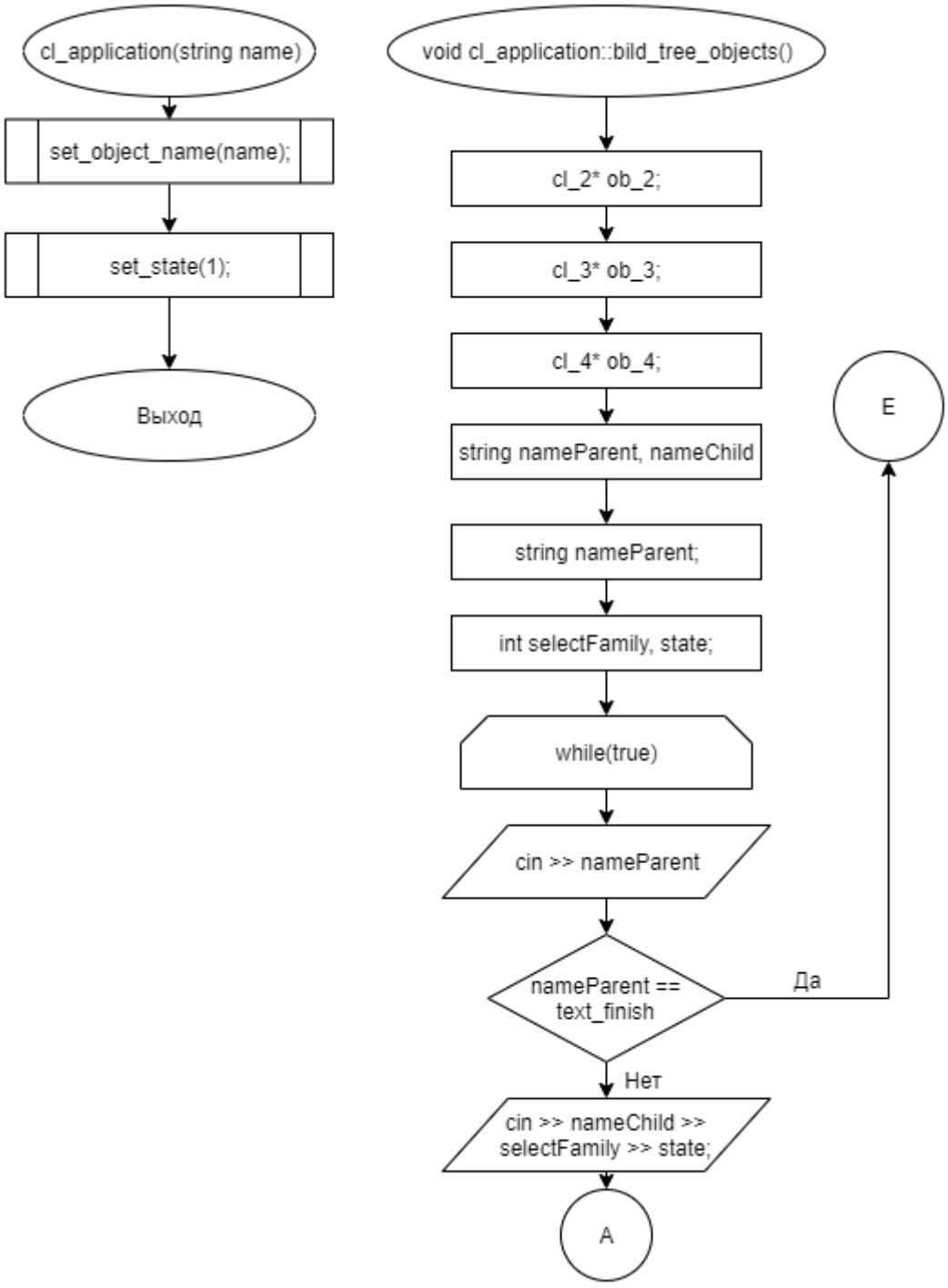
int cl_base::get_state(cl_base* p_parent)

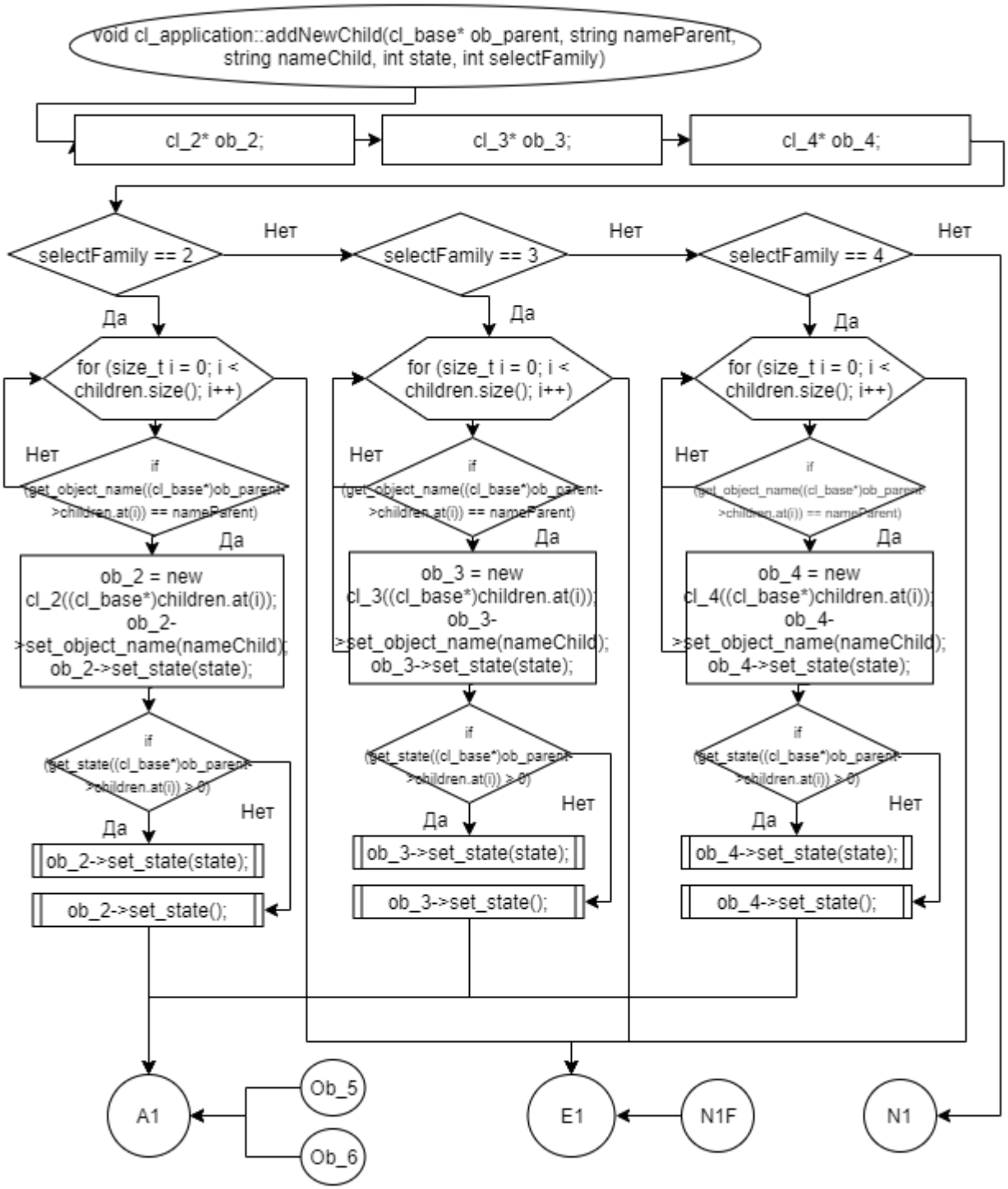
№ шага	Предикат	Действе	№ п
1		return p_parent->c_state;	Ø

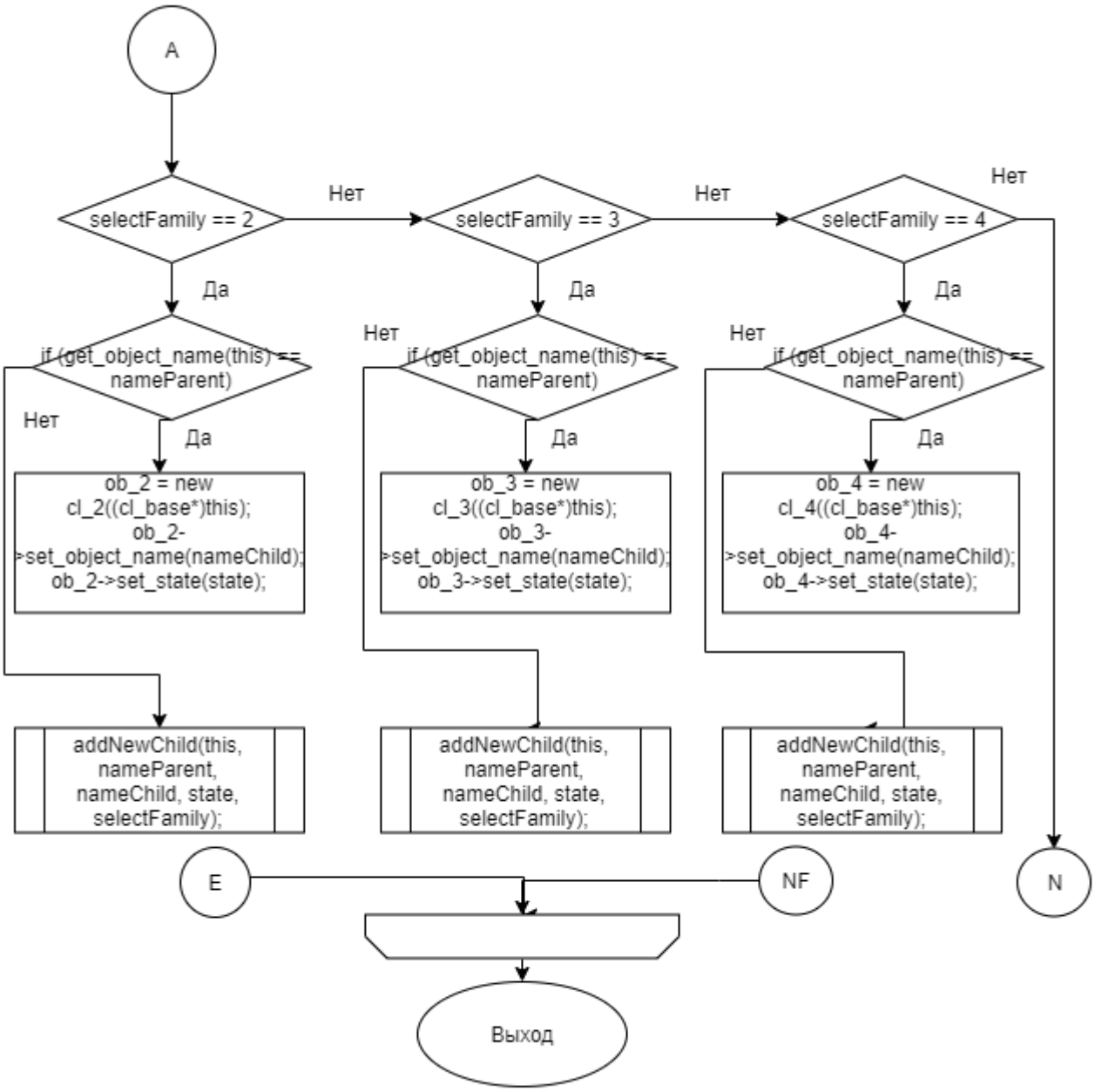
int main()

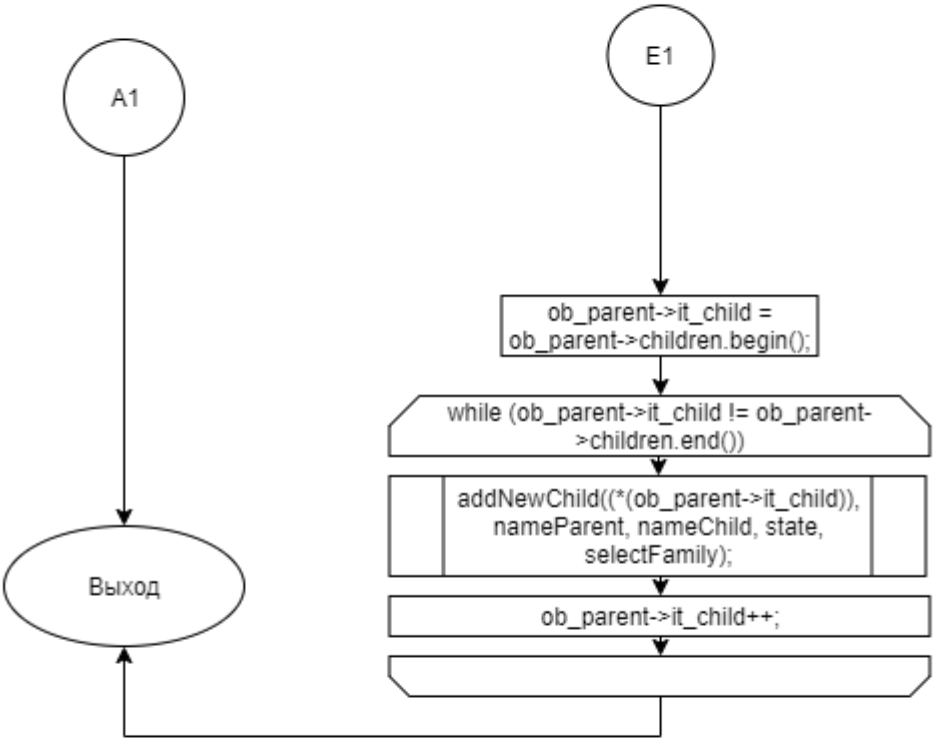
№ шага	Предикат	Действе	N
1		string name;	2
2		cin >> name;	3
3		cl_application ob_application(name);	4
4		ob_application.bild_tree_objects();	5
5		cout << "Object tree";	6
6		return ob_application.exec_app();	Ø

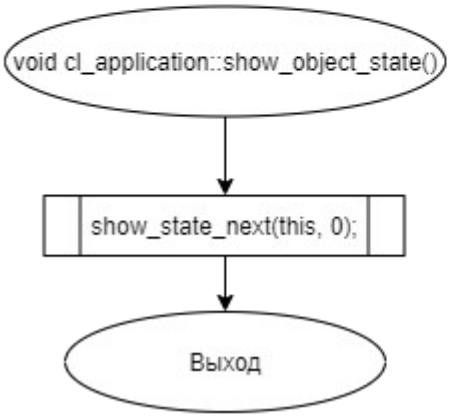
Блок-схема алгоритма

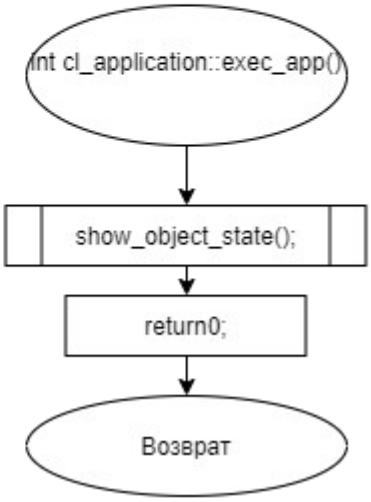

cl_application(string name)
set_object_name(name);
set_state(1);
Выход
void cl_application::bild_tree_objects()
cl_2* ob_2;
cl_3* ob_3;
cl_4* ob_4;
string nameParent, nameChild
string nameParent;
int selectFamily, state;
while(true)
cin >> nameParent
nameParent == text_finish
Да
E
Нет
cin >> nameChild >> selectFamily >> state;
A

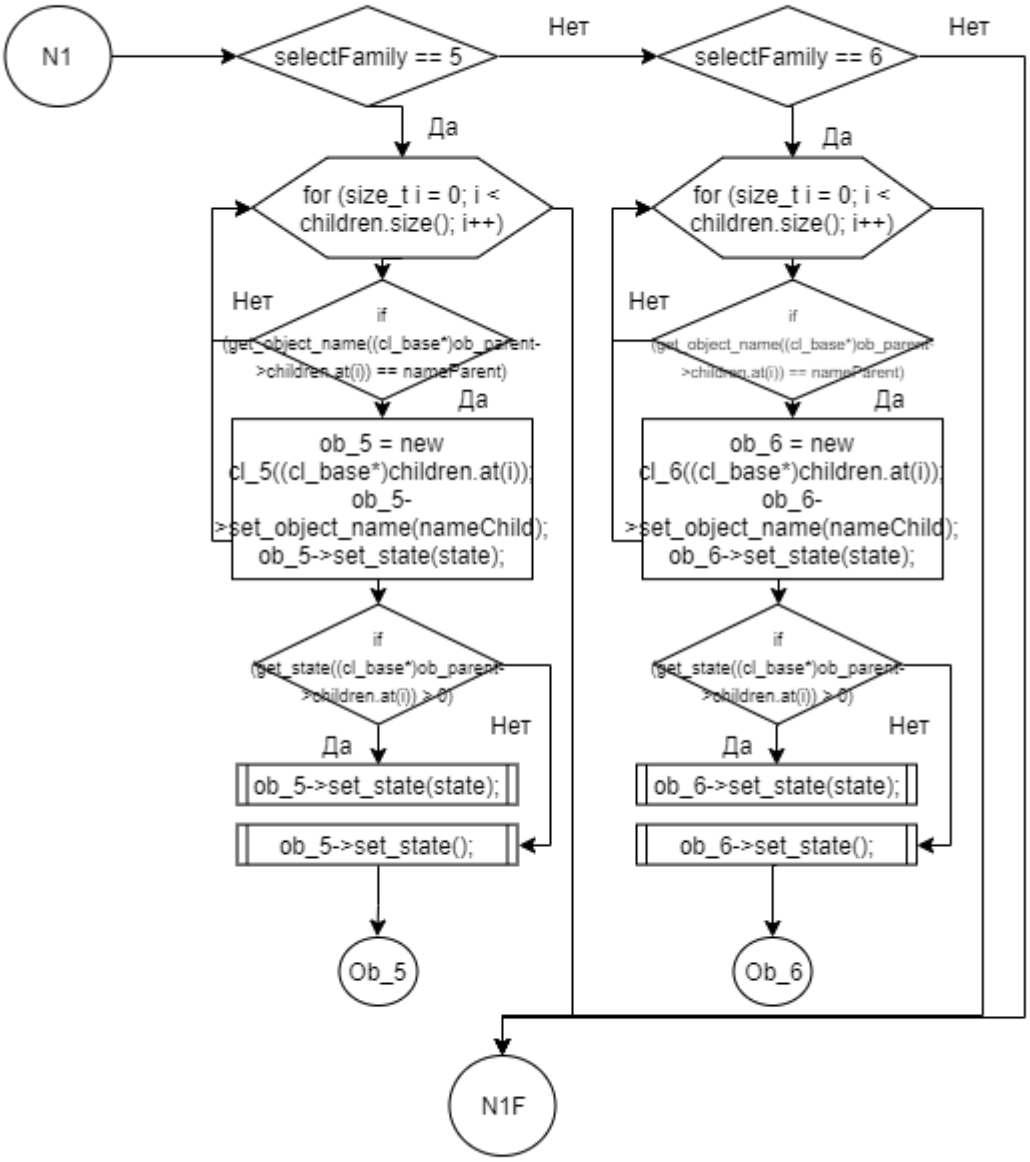

void cl_application::addNewChild(cl_base* ob_parent, string nameParent, string nameChild, int state, int selectFamily)
cl_2* ob_2;
cl_3* ob_3;
cl_4* ob_4;
selectFamily == 2
Нет
selectFamily == 3
Нет
selectFamily == 4
Нет
Да
Да
Да
for (size_t i = 0; i < children.size(); i++)
for (size_t i = 0; i < children.size(); i++)
for (size_t i = 0; i < children.size(); i++)
Нет
Нет
Нет
if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)
if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)
if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)
Да
Да
Да
ob_2 = new cl_2((cl_base*)children.at(i)); ob_2->set_object_name(nameChild); ob_2->set_state(state);
ob_3 = new cl_3((cl_base*)children.at(i)); ob_3->set_object_name(nameChild); ob_3->set_state(state);
ob_4 = new cl_4((cl_base*)children.at(i)); ob_4->set_object_name(nameChild); ob_4->set_state(state);
if (get_state((cl_base*)ob_parent->children.at(i)) > 0)
if (get_state((cl_base*)ob_parent->children.at(i)) > 0)
if (get_state((cl_base*)ob_parent->children.at(i)) > 0)
Нет
Нет
Нет
Да
Да
Да
ob_2->set_state(state);
ob_3->set_state(state);
ob_4->set_state(state);
ob_2->set_state();
ob_3->set_state();
ob_4->set_state();
A1
Ob_5
Ob_6
E1
N1F
N1

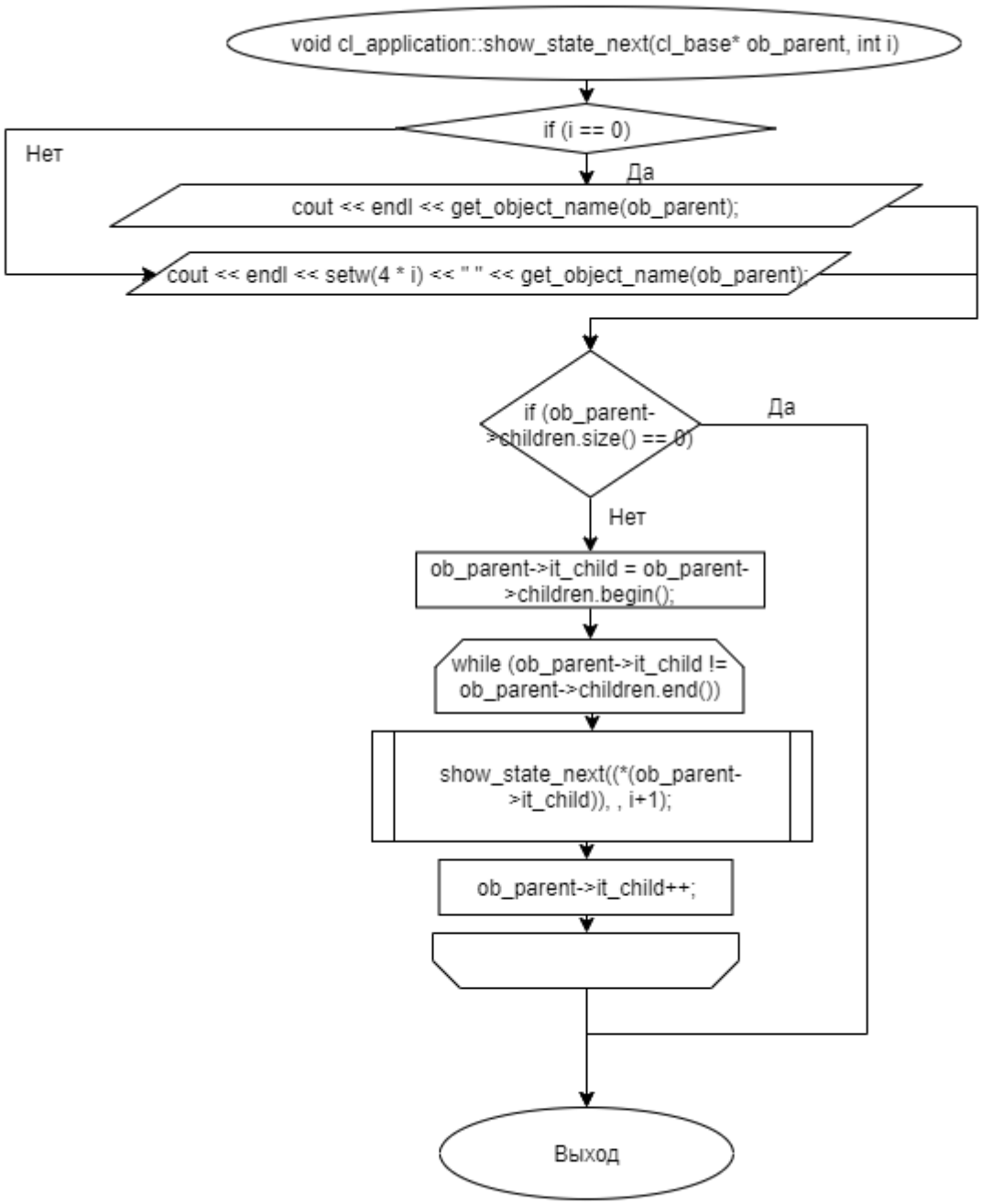

A
selectFamily == 2
Нет
selectFamily == 3
Нет
selectFamily == 4
Нет
Да
Да
Да
Нет
Нет
if (get_object_name(this) == nameParent)
if (get_object_name(this) == nameParent)
if (get_object_name(this) == nameParent)
Нет
Да
Да
Да
ob_2 = new cl_2((cl_base*)this); ob_2->set_object_name(nameChild); ob_2->set_state(state);
ob_3 = new cl_3((cl_base*)this); ob_3->set_object_name(nameChild); ob_3->set_state(state);
ob_4 = new cl_4((cl_base*)this); ob_4->set_object_name(nameChild); ob_4->set_state(state);
addNewChild(this, nameParent, nameChild, state, selectFamily);
addNewChild(this, nameParent, nameChild, state, selectFamily);
addNewChild(this, nameParent, nameChild, state, selectFamily);
E
NF
N
Выход

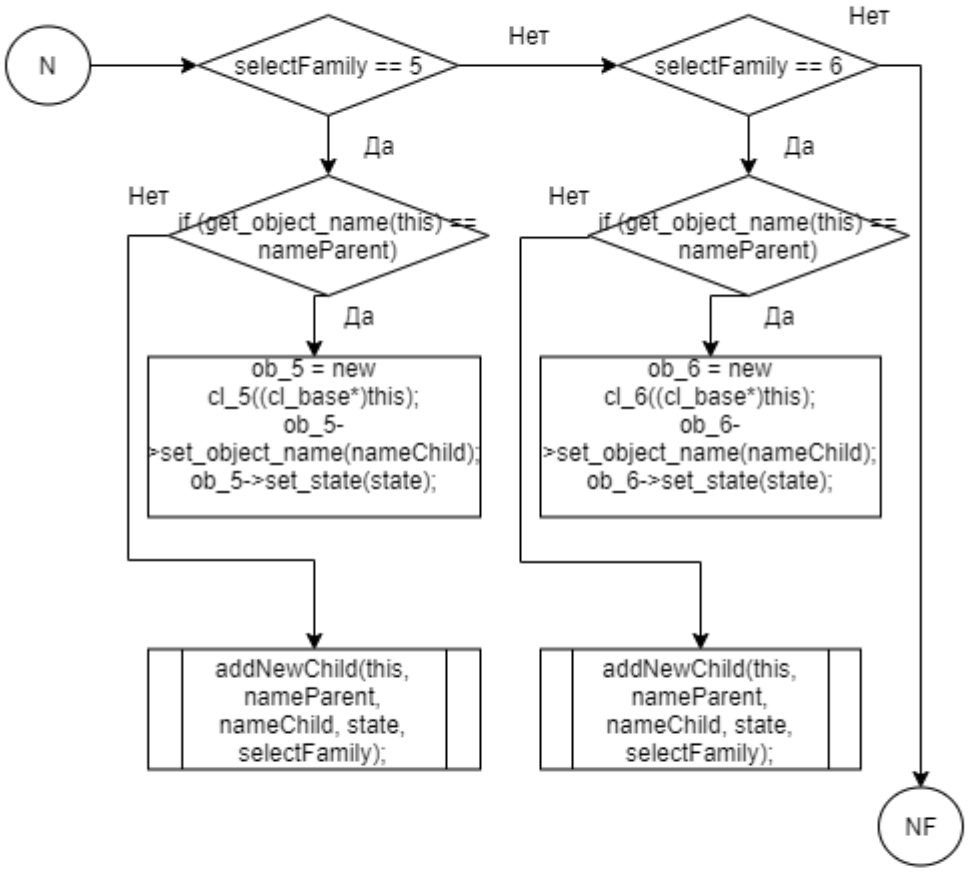

E1
A1
ob_parent->it_child = ob_parent->children.begin();
while (ob_parent->it_child != ob_parent->children.end())
addNewChild((*(ob_parent->it_child)), nameParent, nameChild, state, selectFamily);
ob_parent->it_child++;
Выход

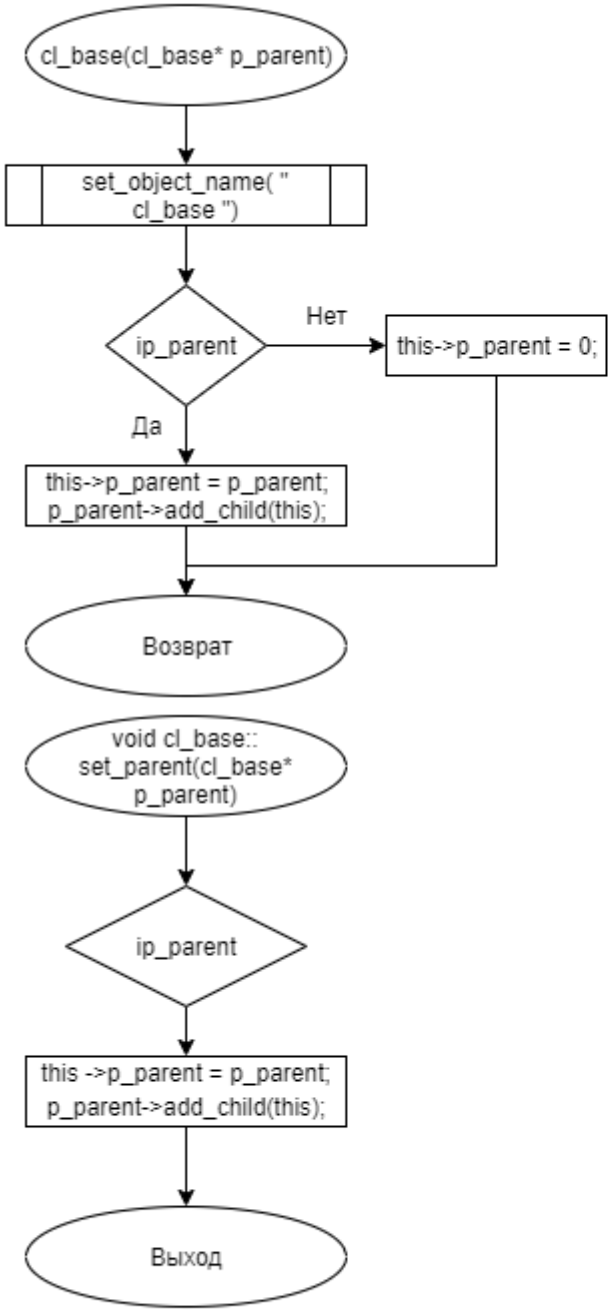

void cl_application::show_object_state()
show_state_next(this, 0);
Выход

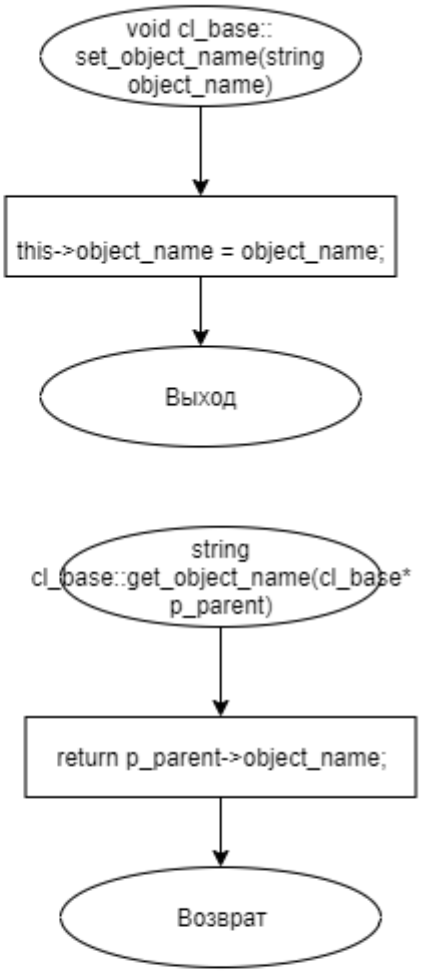

int cl_application::exec_app()
show_object_state();
return0;
Возврат

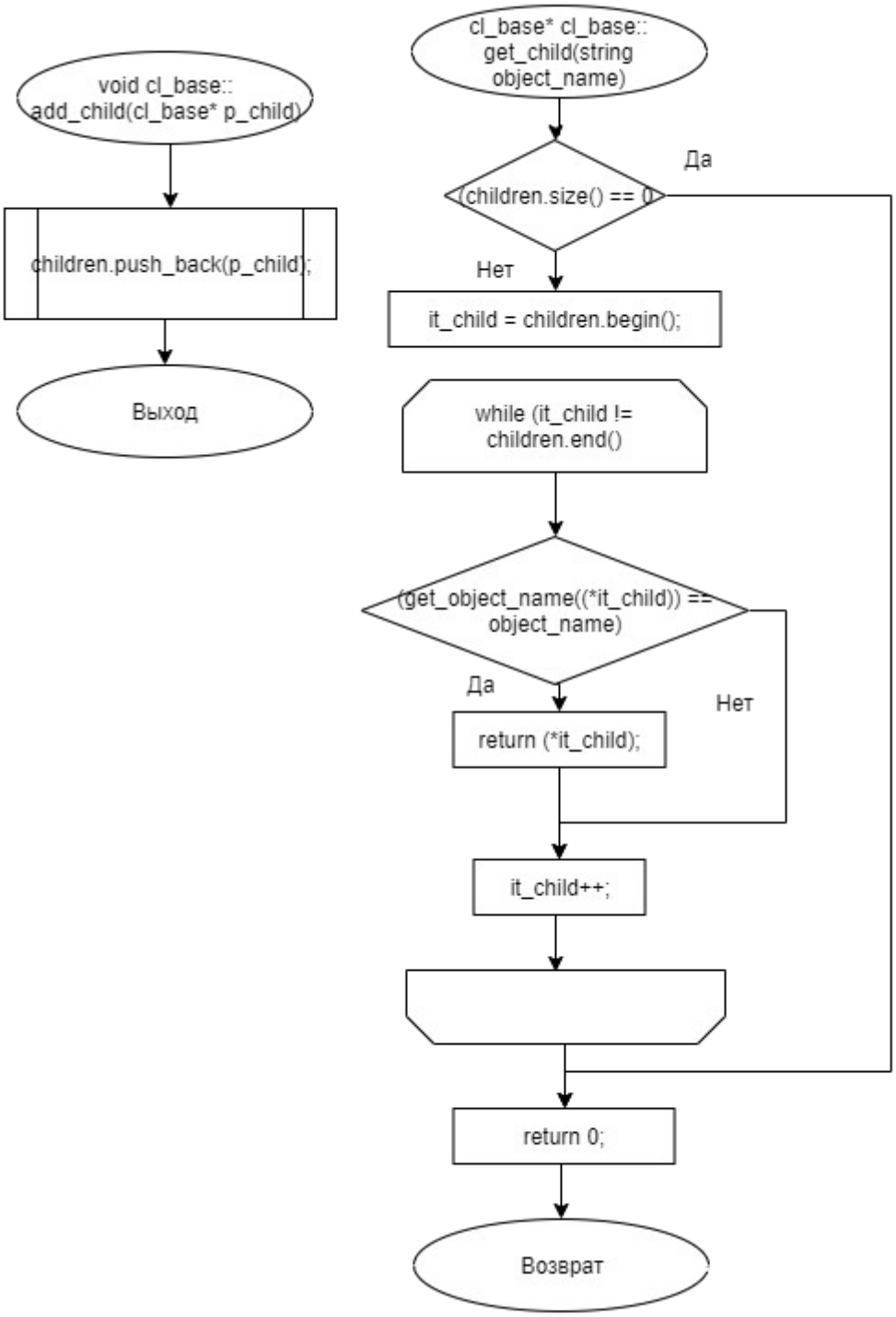

N1
selectFamily == 5
Нет
selectFamily == 6
Нет
Да
Да
for (size_t i = 0; i < children.size(); i++)
for (size_t i = 0; i < children.size(); i++)
Нет
if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)
Нет
if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)
Да
Да
ob_5 = new cl_5((cl_base*)children.at(i)); ob_5->set_object_name(nameChild); ob_5->set_state(state);
ob_6 = new cl_6((cl_base*)children.at(i)); ob_6->set_object_name(nameChild); ob_6->set_state(state);
if (get_state((cl_base*)ob_parent->children.at(i)) > 0)
if (get_state((cl_base*)ob_parent->children.at(i)) > 0)
Нет
Нет
Да
Да
ob_5->set_state(state);
ob_6->set_state(state);
ob_5->set_state();
ob_6->set_state();
Ob_5
Ob_6
N1F

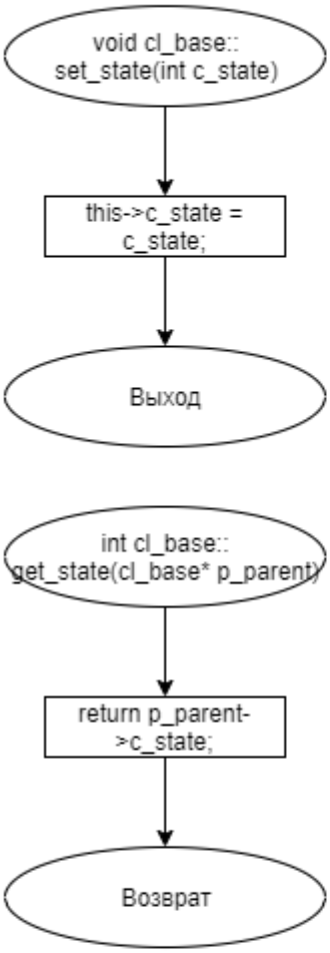

void cl_application::show_state_next(cl_base* ob_parent, int i)
if (i == 0)
Нет
Да
cout << endl << get_object_name(ob_parent);
cout << endl << setw(4 * i) << " " << get_object_name(ob_parent);
if (ob_parent->children.size() == 0)
Да
Нет
ob_parent->it_child = ob_parent->children.begin();
while (ob_parent->it_child != ob_parent->children.end())
show_state_next((*(ob_parent->it_child)), , i+1);
ob_parent->it_child++;
Выход

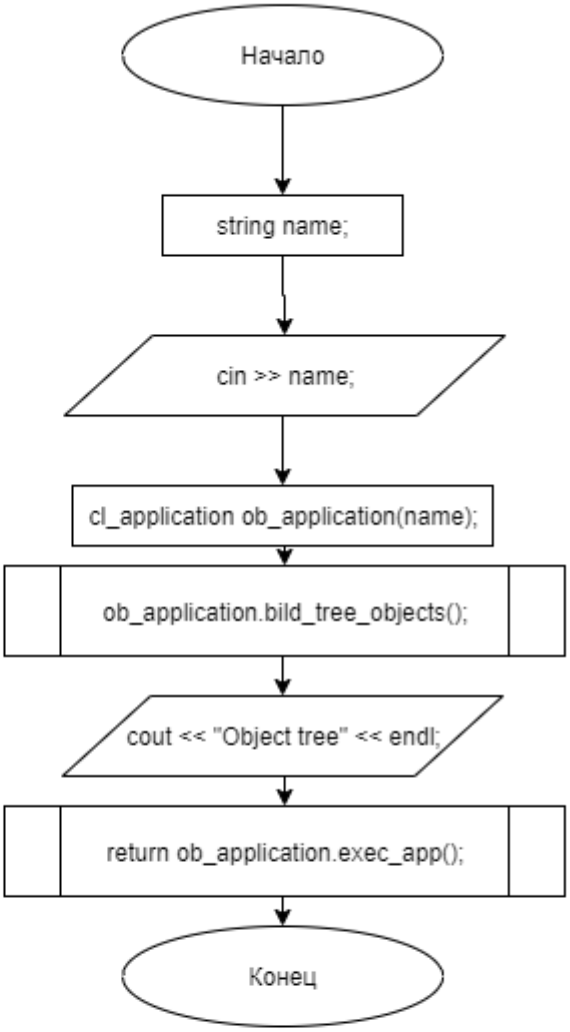

N
selectFamily == 5
Нет
selectFamily == 6
Нет
Да
Да
Нет
if (get_object_name(this) == nameParent)
Нет
if (get_object_name(this) == nameParent)
Да
Да
ob_5 = new cl_5((cl_base*)this);
ob_5->set_object_name(nameChild);
ob_5->set_state(state);
ob_6 = new cl_6((cl_base*)this);
ob_6->set_object_name(nameChild);
ob_6->set_state(state);
addNewChild(this, nameParent, nameChild, state, selectFamily);
addNewChild(this, nameParent, nameChild, state, selectFamily);
NF


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_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


void cl_base::
set_state(int c_state)
this->c_state =
c_state;
Выход
int cl_base::
get_state(cl_base* p_parent)
return p_parent-
>c_state;
Возврат


Начало
string name;
cin >> name;
cl_application ob_application(name);
ob_application.bild_tree_objects();
cout << "Object tree" << endl;
return ob_application.exec_app();
Конец

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_3.cpp

```
#include "cl_3.h"

cl_3::cl_3(cl_base* p_parent) : cl_base(p_parent) {}
```

Файл 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_parent = 0);
};

#endif // CL_3_H
```

Файл cl_4.cpp

```
#include "cl_4.h"

cl_4::cl_4(cl_base* p_parent) : cl_base(p_parent) {}
```

Файл 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_parent = 0);
};

#endif // CL_4_H
```

Файл cl_5.cpp

```
#include "cl_5.h"

cl_5::cl_5(cl_base* p_parent) : cl_base(p_parent) {}
```

Файл 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_parent = 0);
};

#endif // CL_5_H
```

Файл cl_6.cpp

```
#include "cl_6.h"

cl_6::cl_6(cl_base* p_parent) : cl_base(p_parent) {}
```

Файл 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* p_parent = 0);
};

#endif // CL_6_H
```

Файл cl_application.cpp

```
#include "cl_application.h"
#include <iomanip>
using namespace std;

cl_application::cl_application(string name) {
```

```
        set_object_name(name);
        set_state(1);
}

void cl_application::bild_tree_objects() {

        cl_2* ob_2;
        cl_3* ob_3;
        cl_4* ob_4;
        cl_5* ob_5;
        cl_6* ob_6;
        string nameParent, nameChild;
        int selectFamily;
        int state;

        while (true) {
                cin >> nameParent;
                if (nameParent == text_finish)
                        break;
                cin >> nameChild >> selectFamily >> state;
                if (selectFamily == 2) {
                        if (get_object_name(this) == nameParent) {
                                ob_2 = new cl_2((cl_base*)this);
                                ob_2->set_object_name(nameChild);
                                ob_2->set_state(state);
                        }
                        else {
                                addNewChild(this, nameParent, nameChild,
state, selectFamily);
                        }

                }
                else if (selectFamily == 3) {

                        if (get_object_name(this) == nameParent) {
                                ob_3 = new cl_3((cl_base*)this);
                                ob_3->set_object_name(nameChild);
                                ob_3->set_state(state);

                        }
                        else {
                                addNewChild(this, nameParent, nameChild,
state, selectFamily);
                        }

                }
                else if (selectFamily == 4) {

                        if (get_object_name(this) == nameParent) {
                                ob_4 = new cl_4((cl_base*)this);
                                ob_4->set_object_name(nameChild);
                                ob_4->set_state(state);
                        }
                        else {
                                addNewChild(this, nameParent, nameChild,
state, selectFamily);
```

```
                        }
                }
                else if (selectFamily == 5) {

                        if (get_object_name(this) == nameParent) {
                                ob_5 = new cl_5((cl_base*)this);
                                ob_5->set_object_name(nameChild);
                                ob_5->set_state(state);
                        }
                        else {
                                addNewChild(this, nameParent, nameChild,
state, selectFamily);
                        }
                }
                else if (selectFamily == 6) {

                        if (get_object_name(this) == nameParent) {
                                ob_6 = new cl_6((cl_base*)this);
                                ob_6->set_object_name(nameChild);
                                ob_6->set_state(state);
                        }
                        else {
                                addNewChild(this, nameParent, nameChild,
state, selectFamily);
                        }
                }
                else
                        break;
        };
}
void cl_application::addNewChild(cl_base* ob_parent, string nameParent, string
nameChild, int state, int selectFamily) {
        cl_2* ob_2;
        cl_3* ob_3;
        cl_4* ob_4;
        cl_5* ob_5;
        cl_6* ob_6;
        if (selectFamily == 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);
                        if (get_state((cl_base*)ob_parent->children.at(i)) >
0) {
                                ob_2->set_state(state);
                        }
                        else {
                                ob_2->set_state(0);
                        }
                        return;
                }
        }
        }
        else if (selectFamily == 3) {
                for (size_t i = 0; i < ob_parent->children.size(); i++) {
                        if (get_object_name((cl_base*)ob_parent-
>children.at(i)) == nameParent) {
                                ob_3 = new cl_3((cl_base*)ob_parent-
>children.at(i));
```

```
                                ob_3->set_object_name(nameChild);
                                if (get_state((cl_base*)ob_parent-
>children.at(i)) > 0) {
                                        ob_3->set_state(state);
                                }
                                else {
                                        ob_3->set_state(0);
                                }
                                return;
                        }
                }
        }
        else if (selectFamily == 4) {
                for (size_t i = 0; i < ob_parent->children.size(); i++) {
                        if (get_object_name((cl_base*)ob_parent-
>children.at(i)) == nameParent) {
                                ob_4 = new cl_4((cl_base*)ob_parent-
>children.at(i));
                                ob_4->set_object_name(nameChild);
                                if (get_state((cl_base*)ob_parent-
>children.at(i)) > 0) {
                                        ob_4->set_state(state);
                                }
                                else {
                                        ob_4->set_state(0);
                                }
                                return;
                        }
                }
        }
        else if (selectFamily == 5) {
                for (size_t i = 0; i < ob_parent->children.size(); i++) {
                        if (get_object_name((cl_base*)ob_parent-
>children.at(i)) == nameParent) {
                                ob_5 = new cl_5((cl_base*)ob_parent-
>children.at(i));
                                ob_5->set_object_name(nameChild);
                                if (get_state((cl_base*)ob_parent-
>children.at(i)) > 0) {
                                        ob_5->set_state(state);
                                }
                                else {
                                        ob_5->set_state(0);
                                }
                                return;
                        }
                }
        }
        else if (selectFamily == 6) {
                for (size_t i = 0; i < ob_parent->children.size(); i++) {
                        if (get_object_name((cl_base*)ob_parent-
>children.at(i)) == nameParent) {
                                ob_6 = new cl_6((cl_base*)ob_parent-
>children.at(i));
                                ob_6->set_object_name(nameChild);
                                if (get_state((cl_base*)ob_parent-
```

```
>children.at(i)) > 0) {
                                        ob_6->set_state(state);
                                }
                                else {
                                        ob_6->set_state(0);
                                }
                                return;
                        }
                }
        }
        ob_parent->it_child = ob_parent->children.begin();
        while (ob_parent->it_child != ob_parent->children.end()) {
                addNewChild((*(ob_parent->it_child)), nameParent, nameChild,
state, selectFamily);
                ob_parent->it_child++;
        }
}
int cl_application::exec_app() {
        show_object_state();
        return 0;
}

void cl_application::show_object_state() {
        show_state_next(this, 0);
}


void cl_application::show_state_next(cl_base* ob_parent, int i) {
        if (i == 0) {
                cout << endl << get_object_name(ob_parent);
        }
        else {
                cout << endl << setw(4 * i) << " " <<
get_object_name(ob_parent);
        }
        if (ob_parent->children.size() == 0)
                return;
        ob_parent->it_child = ob_parent->children.begin();
        while (ob_parent->it_child != ob_parent->children.end()) {
                show_state_next((*(ob_parent->it_child)), i+1 ); ob_parent-
>it_child++;
        }
}
```

Файл cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H

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

```
#include "cl_base.h"

class cl_application : public cl_base {

public:

cl_application(string name);
void bild_tree_objects();
int exec_app();
void show_object_state();
void addNewChild(cl_base* ob_parent, string nameParent, string nameChild, int
state, int selectFamily);

private:

void show_state_next(cl_base* ob_parent, int i);


};

#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;
}



void cl_base::set_state(int c_state) {
        this->c_state = c_state;
}

int cl_base::get_state(cl_base* p_parent) {
        return p_parent->c_state;
}
```

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);
```

```
void set_state(int c_state);
int get_state(cl_base* p_parent);

vector < cl_base* > children;
vector < cl_base* > ::iterator it_child;
string text_finish = "endtree";

private:

string  object_name;

cl_base* p_parent;

int c_state;


};

#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();
cout << "Object tree";
return ob_application.exec_app();
}
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 3 -1 object_2 object_5 3 1 app_root object_3 3 1 object_2 object_6 2 1 object_1 object_7 2 1 object_7 object_8 2 1 endtree	Object tree app_root object_1 object_7 object_8 object_2 object_4 object_5 object_6 object_3	Object tree app_root object_1 object_7 object_8 object_2 object_4 object_5 object_6 object_3
app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 3 -1 object_4 object_10 3 1 object_10 object_11 3 1 object_2 object_5 3 1 app_root object_3 3 1 object_2 object_6 2 1 object_1 object_7 2 1 object_7 object_8 2 1 endtree	Object tree app_root object_1 object_7 object_8 object_2 object_4 object_10 object_11 object_5 object_6 object_3	Object tree app_root object_1 object_7 object_8 object_2 object_4 object_10 object_11 object_5 object_6 object_3
app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 3 -1 object_2 object_5 3 1 app_root object_3 3 1 object_2 object_6 2 1 object_1 object_7 2 1 endtree	Object tree app_root object_1 object_7 object_2 object_4 object_5 object_6 object_3	Object tree app_root object_1 object_7 object_2 object_4 object_5 object_6 object_3
app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 4 -1 object_4 object_10 3 1 object_10 object_11 3 1 object_2 object_5 3 1 app_root object_3 5 1 object_2 object_6 2 1 object_1 object_7 6 1 object_7 object_8 2 1 endtree	Object tree app_root object_1 object_7 object_8 object_2 object_4 object_10 object_11 object_5 object_6 object_3	Object tree app_root object_1 object_7 object_8 object_2 object_4 object_10 object_11 object_5 object_6 object_3