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_1 Проверка готовности объектов к работе »

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

Проверка готовности объектов к работе

Фрагмент методического указания.

Создание объектов и построение исходного иерархического дерева объектов.

Система собирается из объектов, принадлежащих определенным классам. В тексте постановки задачи классу соответствует уникальный номер. Относительно номера класса определяются требования (свойства, функциональность).

Первоначальная сборка системы (дерева иерархии объектов, программы) осуществляется исходя из входных данных. Данные вводятся построчно.

Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Корневой объект объявляется в основной программе (main). Исходное состояние корневого объекта соответствует его функционированию.

Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта»«Наименование очередного объекта»«Номер класса принадлежности очередного объекта»«Номер исходного состояния очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно « endtree » (в данной строке ввода больше ничего не указывается).

Готовность объекта характеризуется значением его состояния. Значение состояния - целое число. Определены правила для значения состояния:

0 – объект выключен;

Отрицательное – объект включен, но не функционирует, обнаружена неисправность. Значение классифицирует характер неисправности.

Положительное – объект включен, функционирует в штатном режиме. Значение определяет текущее состояние объекта.

Подчиненные объекты располагаются слева на право относительно голоного, согласно их следованию в исходных данных. Исходные данные подготовлены таким образом, что любой головной объект предварительно добавлен в качестве подчиненного.

Подразумевается, что все объекты имеют уникальные имена.

Для организации исходя из входных данных создания экземпляров объектов и формирования иерархического дерева, необходимо:

1. В базовом классе реализовать метод поиска объекта на дереве объектов по его наименованию и возврата указателя на него. Если объект не найден, то вернуть нулевой указатель.

2. В корневом объекте (объект приложения) реализовать метод чтения исходных данных, создания объектов и построения исходного дерева иерархии.

Пример

Ввод

```
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
```

Построенное дерево

```
app_root

    object_1

    object_7

    object_2

    object_4

    object_5

    object_6

    object_3
```

Вывод списка готовности объектов

```
The object app_root is ready

The object object_1 is ready

The object object_7 is ready

The object object_2 is ready

The object object_4 is not ready
```

```
The object object_5 is ready

The object object_6 is ready

The object object_3 is ready
```

Постановка задачи

Все сложные электронные, технические средства разного назначения в момент включения выполняют опрос готовности к работе составных элементов, индицируя соответствующую информацию на табло, панели или иным образом.

Построить модель иерархической системы. Реализовать задачу опроса готовности каждого объекта из ее состава и вывести соответствующее сообщение на консоль.

Объект считается готовым к работе:

1. Создан и размешен в составе системы (на дереве иерархии объектов) согласно схеме архитектуры;
2. Имеет свое уникальное наименование;
3. Свойство, определяющее его готовность к работе, имеет целочисленное положительное значение.

В результате решения задачи опроса готовности объектов, относительно каждого объекта системы на консоль надо вывести соответствующую информацию:

Если свойство определяющее готовность объекта имеет положительное значение:

The object «наименование объекта» is ready

иначе

The object « наименованиеобъекта » is not ready

Система содержит объекты трех классов, не считая корневого. Номера классов: 2,3,4.

Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Структура данных для ввода согласно изложенному в фрагменте методического указания.

Описание выходных данных

В первой строке вывести Test result Далее, построчно, согласно следованию объектов на дереве иерархии слева на право и сверху вниз, относительно каждого объекта в зависимости от состояния готовности выводиться, если объект готов к работе: The object «наименование объекта» is ready Если не готов, то The object «наименование объекта» is not ready

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);	2
2		set_state(1);	Ø

void cl_application::bild_tree_objects()

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

22	if (selectFamily == 4)		23
	else		28
23	if (this->get_object_name() == nameParent)		24
	else		27
24		ob_4 = new cl_4((cl_base*)this);	25
25		ob_4->set_object_name(nameChild);	26
26		ob_4->set_state(state);	6
27		addNewChild(this, nameParent, nameChild, state, selectFamily);	6
28		break;	Ø

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

№ шага	Предикат	Действе	№ п
1		cl_2* ob_2; cl_3* ob_3; cl_4* ob_4;	2
2	if (selectFamily == 2)		3
	else		8
3	for (size_t i = 0; i < ob_parent->children.size(); i++)		4
	i = ob_parent->children.size()		20
4	if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)	ob_2 = new cl_2((cl_base*)ob_parent->children.at(i));	5
	else		3
5		ob_2->set_object_name(nameChild);	6
6	if (get_state((cl_base*)ob_parent->children.at(i)) == 1)	ob_2->set_state(state);	7
	else	ob_2->set_state(0);	7
7		return;	Ø
8	if (selectFamily == 3)		9
	else		14
9	for (size_t i = 0; i < ob_parent->children.size(); i++)		10
	i = ob_parent->children.size()		20
10	if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)	ob_3 = new cl_3((cl_base*)ob_parent->children.at(i));	11
	else		9
11		ob_3->set_object_name(nameChild);	12
12	if (get_state((cl_base*)ob_parent->children.at(i)) == 1)	ob_3->set_state(state);	13
	else	ob_3->set_state(0);	13

13		return;	Ø
14	if (selectFamily == 4)		15
	else		20
15	for (size_t i = 0; i < ob_parent->children.size(); i++)		16
	i = ob_parent->children.size()		20
16	if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)	ob_4 = new cl_4((cl_base*)ob_parent->children.at(i));	17
	else		15
17		ob_4->set_object_name(nameChild);	18
18	if (get_state((cl_base*)ob_parent->children.at(i)) == 1)	ob_4->set_state(state);	19
	else	ob_4->set_state(0);	19
19		return;	Ø
20		ob_parent->it_child = ob_parent->children.begin();	21
21	while (ob_parent->it_child != ob_parent->children.end())	addNewChild((*(ob_parent->it_child)), nameParent, nameChild, state, selectFamily);	22
	ob_parent->it_child == ob_parent->children.end()		Ø
22		ob_parent->it_child++;	21

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		show_state_next(this);	Ø

void cl_application::show_state_next(cl_base* ob_parent)

№ шага	Предикат	Действе	№ перехода
1	if (get_state(ob_parent) == 0)	return;	Ø
	else		2
2	if (get_state(ob_parent) == 1)	cout << "The object " << ob_parent->get_object_name() << " is ready" << endl;	4
	else		3

3	if (get_state(ob_parent) == -1)	cout << "The object " << ob_parent->get_object_name() << " is not ready" << endl;	4
	else		4
4	if (ob_parent->children.size() == 0)	return;	Ø
	else		5
5		ob_parent->it_child = ob_parent->children.begin();	6
6	while (ob_parent->it_child != ob_parent->children.end())	show_state_next((*(ob_parent->it_child)));	7
	ob_parent->it_child == ob_parent->children.end()		Ø
7		ob_parent->it_child++;	6

cl_base(cl_base* p_parent)

№ шага	Предикат	Действе	№ перехода
1		set_object_name(" cl_base ");	2
2	if(p_parent)	this->p_parent = 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	№ переход
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	№ п
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

void cl_base::set_state(int c_state)

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

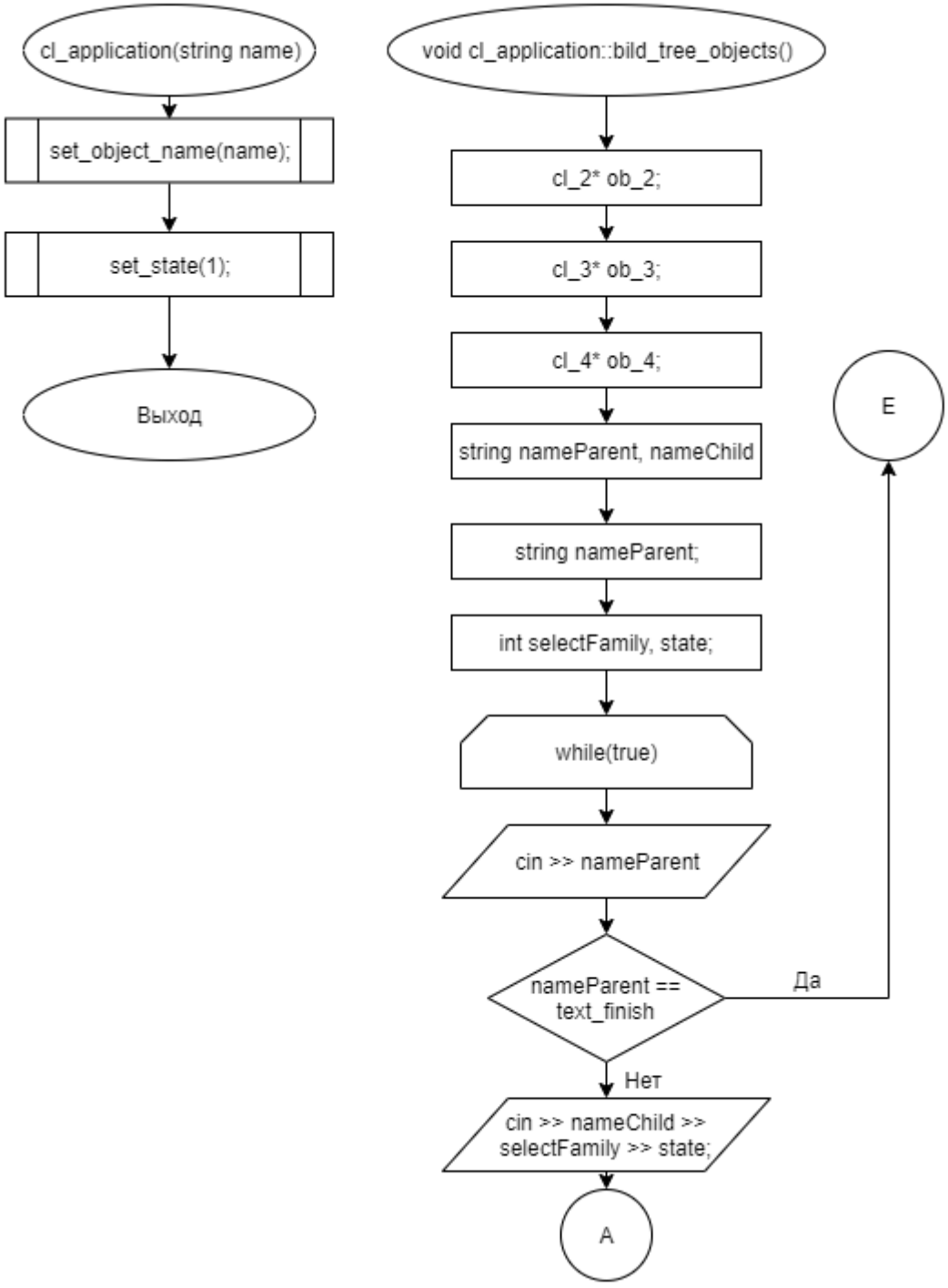
int cl_base::get_state(cl_base* p_parent)

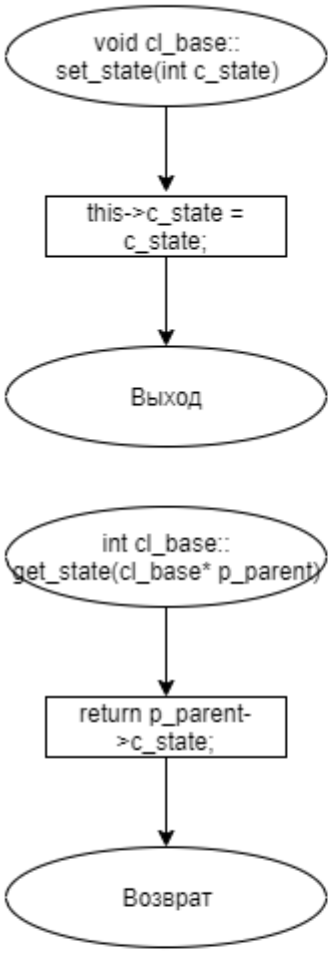
№ шага	Предикат	Действе	№ перехода
1		return p_parent->c_state;	Ø

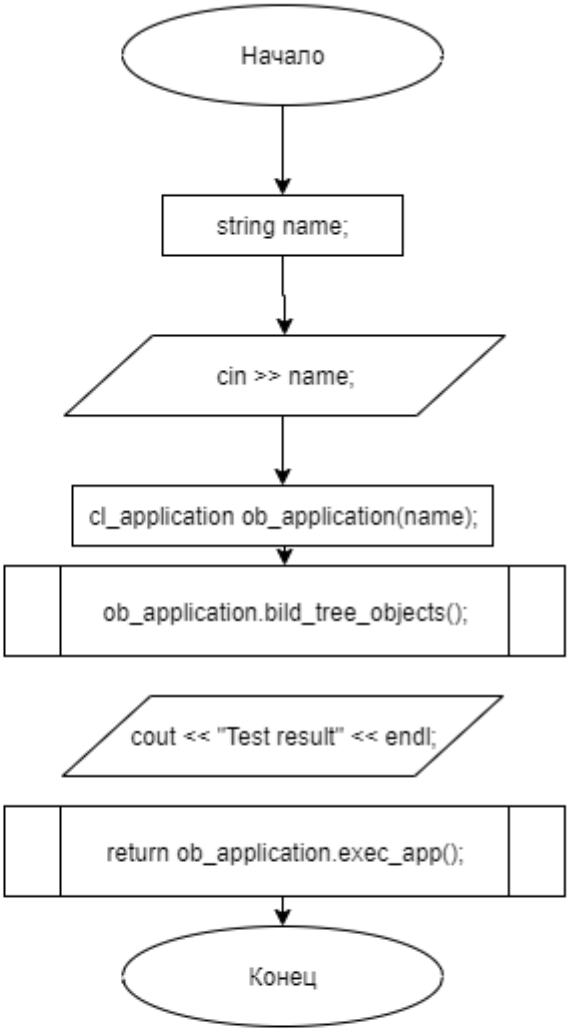
int main()

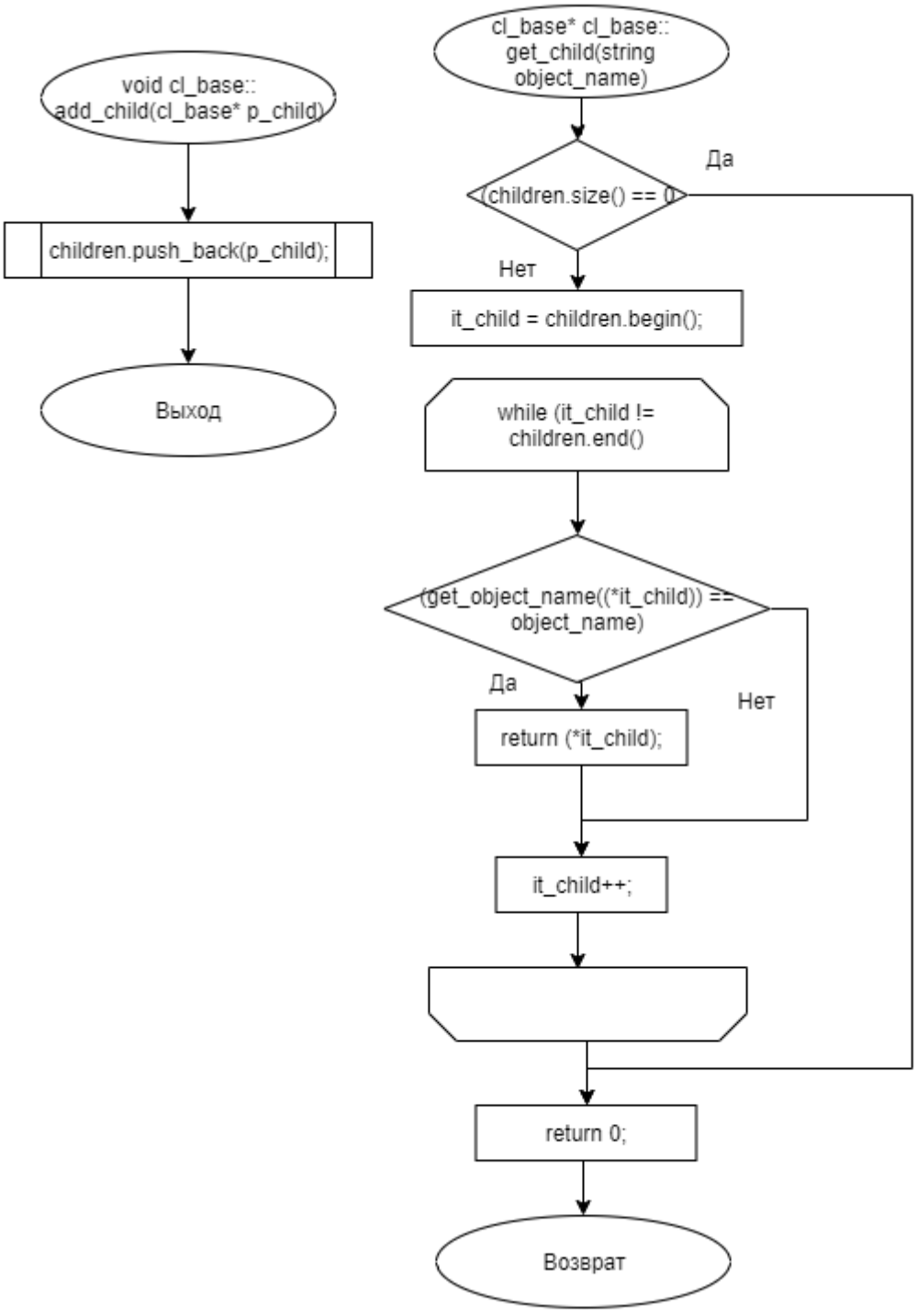
№ шага	Предикат	Действе	№ перехода
1		string name;	2
2		cin >> name;	3
3		cl_application ob_application(name);	4
4		ob_application.bild_tree_objects();	5
5		cout << "Test result" << endl;	6
6		return ob_application.exec_app();	Ø

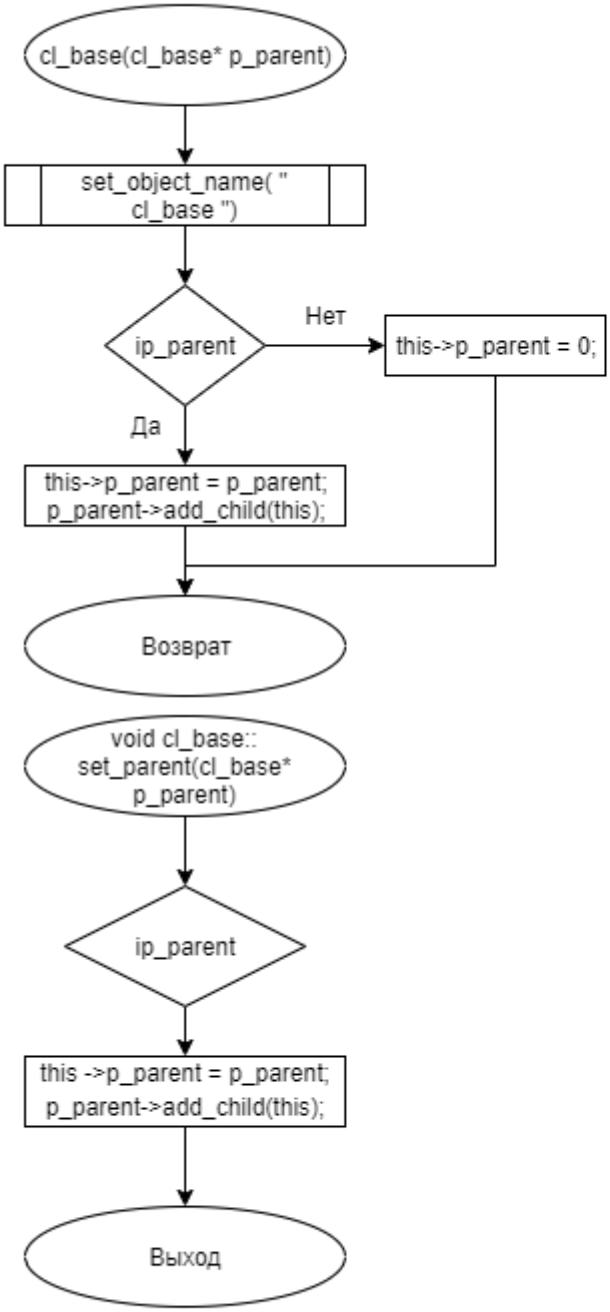
Блок-схема алгоритма

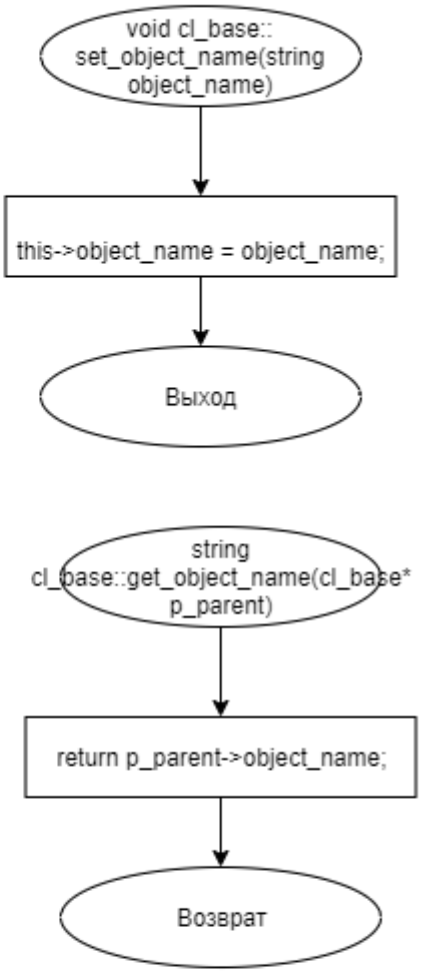

cl_application(string name)
set_object_name(name);
set_state(1);
Выход
void cl_application::bild_tree_objects()
cl_2* ob_2;
cl_3* ob_3;
cl_4* ob_4;
string nameParent, nameChild
string nameParent;
int selectFamily, state;
while(true)
cin >> nameParent
nameParent == text_finish
Да
E
Нет
cin >> nameChild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

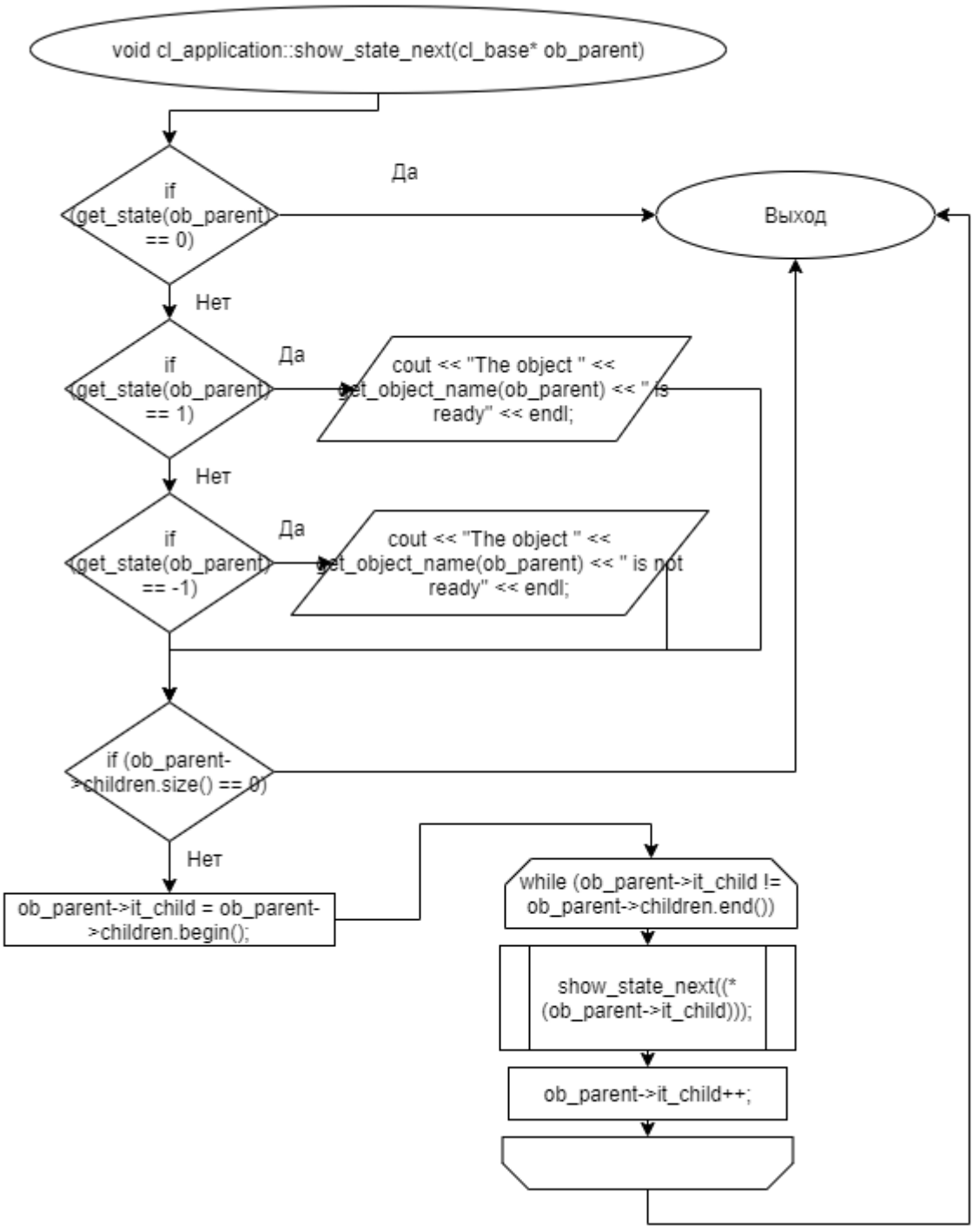

void cl_base::
set_state(int c_state)
this->c_state =
c_state;
Выход
int cl_base::
get_state(cl_base* p_parent)
return p_parent-
>c_state;
Возврат

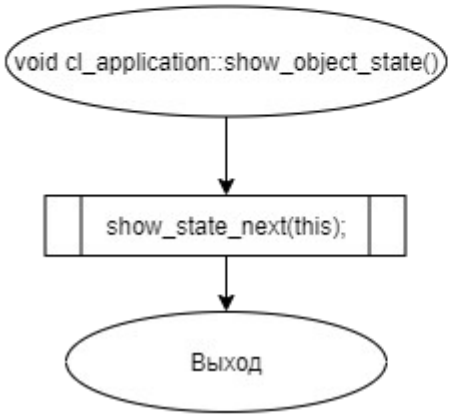

Начало
string name;
cin >> name;
cl_application ob_application(name);
ob_application.bild_tree_objects();
cout << "Test result" << endl;
return ob_application.exec_app();
Конец

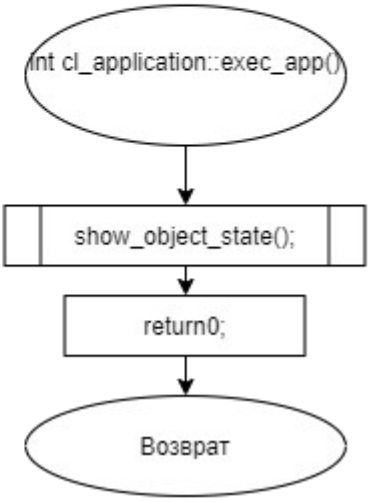

void cl_base::
add_child(cl_base* p_child)
children.push_back(p_child);
Выход
cl_base* cl_base::
get_child(string
object_name)
(children.size() == 0)
Да
Нет
it_child = children.begin();
while (it_child !=
children.end()
(get_object_name((*it_child)) ==
object_name)
Да
Нет
return (*it_child);
it_child++;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

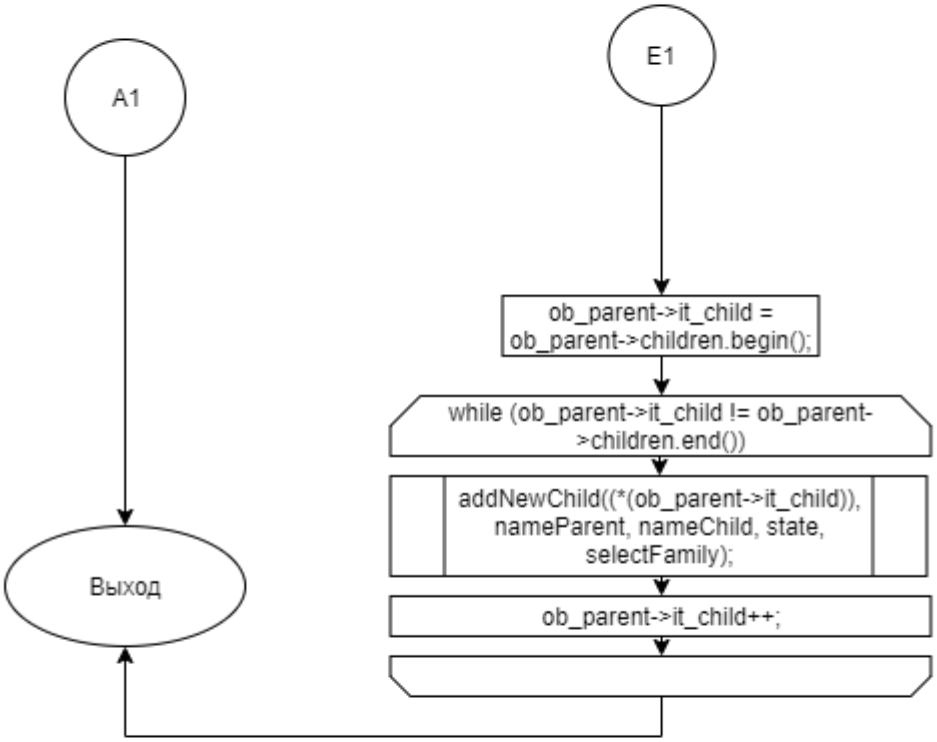

cl_base(cl_base* p_parent)
set_object_name(" cl_base ")
ip_parent
Нет
this->p_parent = 0;
Да
this->p_parent = p_parent;
p_parent->add_child(this);
Возврат
void cl_base::
set_parent(cl_base*
p_parent)
ip_parent
this ->p_parent = p_parent;
p_parent->add_child(this);
Выход

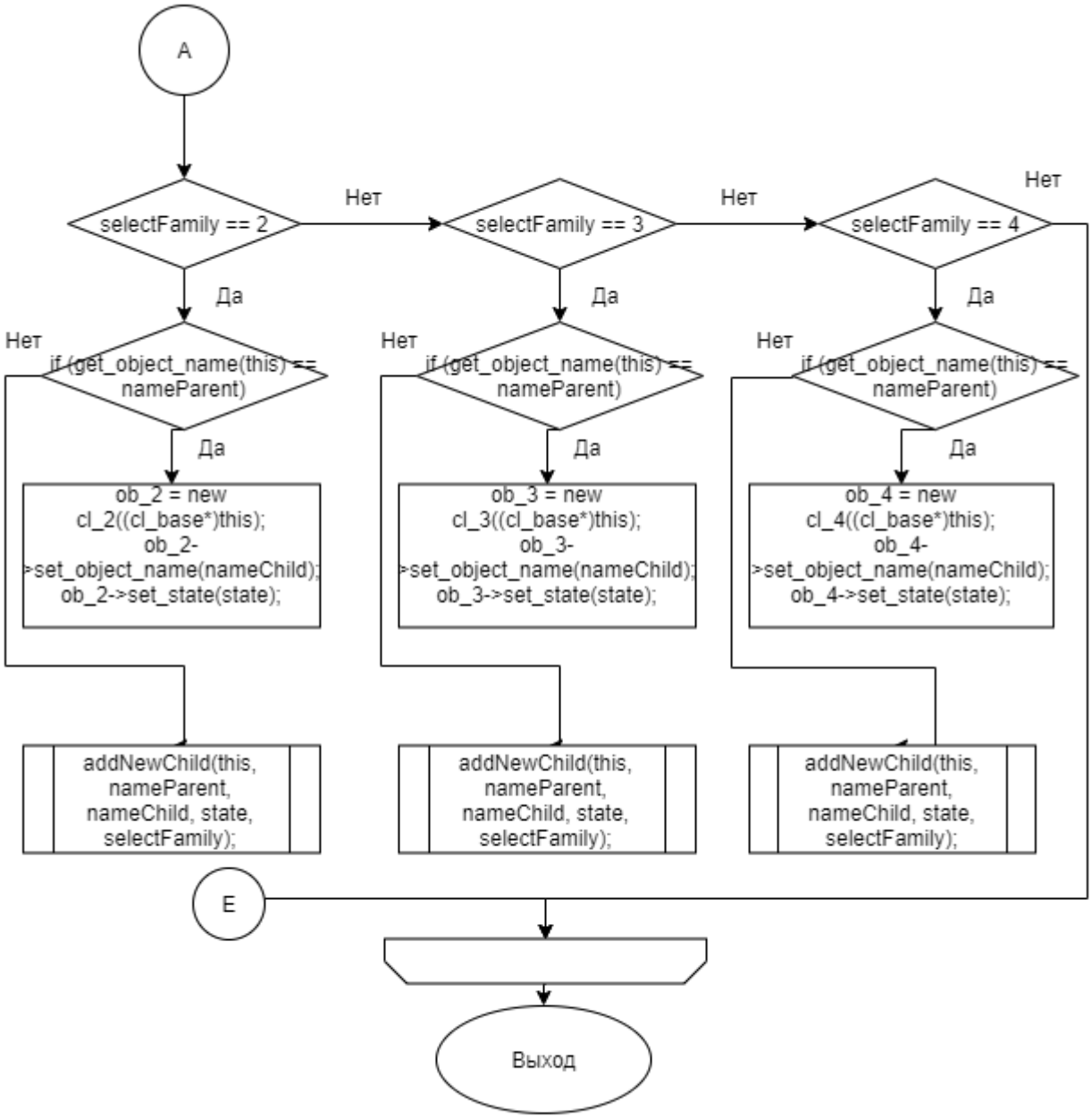

void cl_base::
set_object_name(string
object_name)
this->object_name = object_name;
Выход
string
cl_base::get_object_name(cl_base*
p_parent)
return p_parent->object_name;
Возврат

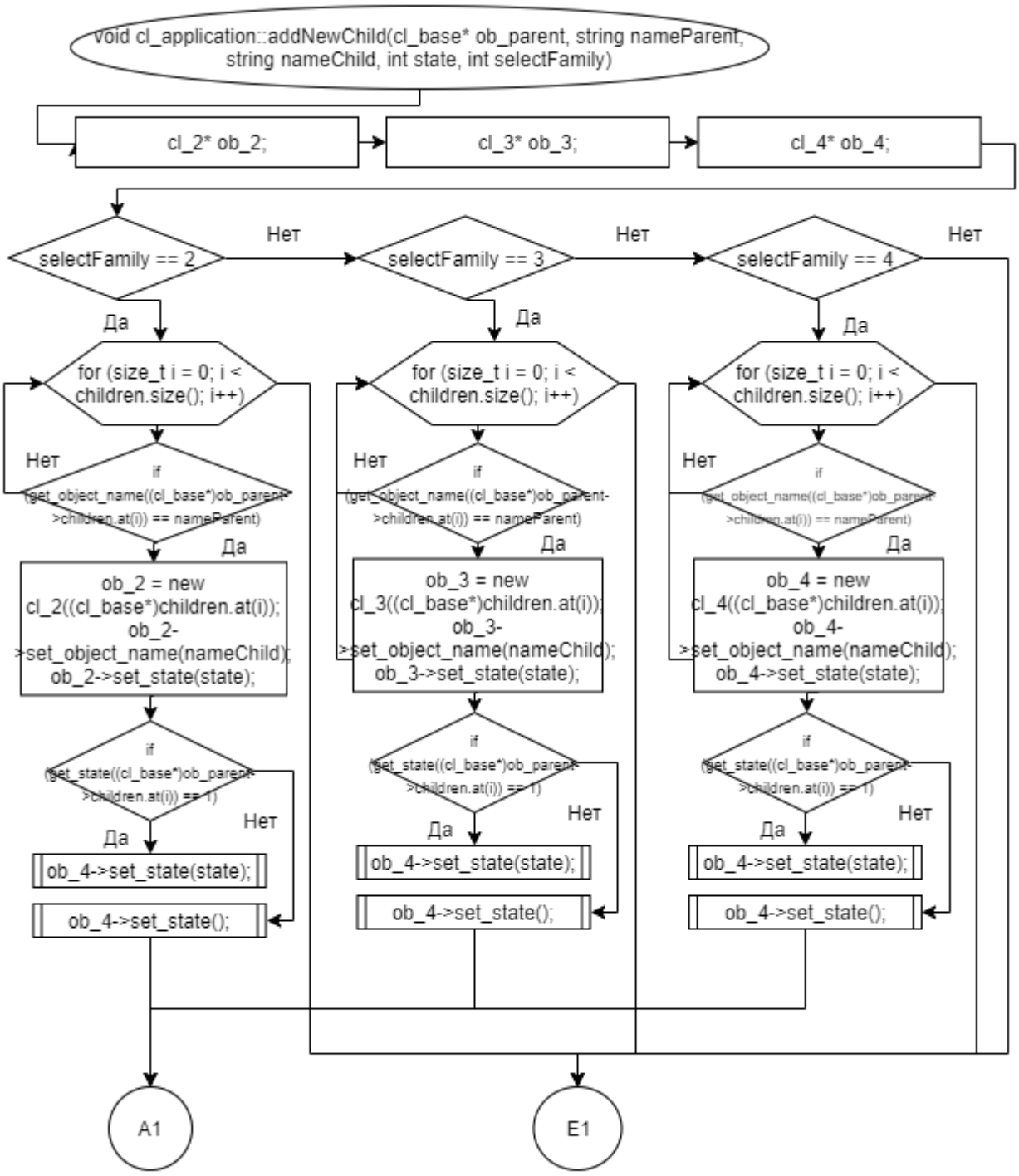

void cl_application::show_state_next(cl_base* ob_parent)
if (get_state(ob_parent) == 0)
Да
Выход
Нет
if (get_state(ob_parent) == 1)
Да
cout << "The object " << get_object_name(ob_parent) << " is ready" << endl;
Нет
if (get_state(ob_parent) == -1)
Да
cout << "The object " << get_object_name(ob_parent) << " is not ready" << endl;
if (ob_parent->children.size() == 0)
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)));
ob_parent->it_child++;


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


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


A1
E1
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


A
selectFamily == 2
Нет
selectFamily == 3
Нет
selectFamily == 4
Нет
Да
Да
Да
Нет
if (get_object_name(this) == nameParent)
Нет
if (get_object_name(this) == nameParent)
Нет
if (get_object_name(this) == nameParent)
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
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_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_application.cpp

```
#include "cl_application.h"
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;
        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
                        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;
        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)) ==
1) {
```

```
                                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)) == 1) {
                                        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)) == 1) {
                                        ob_4->set_state(state);
                                }
                                else {
                                        ob_4->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);
}
```

```
void cl_application::show_state_next(cl_base* ob_parent) {
        if (get_state(ob_parent) == 0)
                return;
        else if (get_state(ob_parent) == 1) {
                cout << "The object " << get_object_name(ob_parent) << " is
ready" << endl;
        }
        else if (get_state(ob_parent) == -1) {
                cout << "The object " << get_object_name(ob_parent) << " is
not ready" << endl;
        }
        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))); 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_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);


};
```

```
#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 << "Test result" << endl;
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 endtree	Test result The object app_root is ready The object object_1 is ready The object object_7 is ready The object object_2 is ready The object object_4 is not ready The object object_5 is ready The object object_6 is ready The object object_3 is ready	Test result The object app_root is ready The object object_1 is ready The object object_7 is ready The object object_2 is ready The object object_4 is not ready The object object_5 is ready The object object_6 is ready The object object_3 is ready