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_3 Определение указателя на объект по его координате »

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

Определение указателя на объект по его координате

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

В составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов.
В качестве параметра методу передать путь объекта от корневого. Путь задать в следующем виде:

```
/ root / ob _1/ ob _2/ ob _3
```

Уникальность наименования требуется только относительно множества подчиненных объектов для любого головного объекта.

Если система содержит объекты с уникальными именами, то в методе реализовать определение указателя на объект посредством задания координаты в виде:

```
//«наименование объекта»
```

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в контрольной работе № 1.
Единственное различие. В строке ввода первым указать не наименование головного объекта, а путь к головному объекту.
Подразумевается, что к моменту ввода очередной строки соответствующая ветка на дереве иерархии уже построена.
Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Пример ввода иерархии дерева объектов.

```
root
/root                object_1              3                1
/root                object_2              2                1
/root/object_2            object_4            3             -1
/root/object_2             object_5            4             1
/root                object_3              3                1
/root/object_2             object_3            6             1
/root/object_1             object_7            5             1
/root/object_2/object_4          object_7          3          -1
endtree
```

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.
После ввода состава дерева иерархии построчно вводится координаты искомых объектов. Ввод завершается при вводе: //

Описание выходных данных

Первая строка:
Object tree

Со второй строки вывести иерархию построенного дерева.

Далее, построчно:
«координата объекта» Object name: «наименование объекта»
Разделитель один пробель.

Если объект не найден, то вывести:
«координата объекта» Object not found
Разделитель один пробель.

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

New function:

void cl_application::bild_tree_objects()

№ шага	Предикат	Действе	№ перехода
1	while(true)	char check, checkNext; cin >> check >> checkNext;	2
	false		7
2	if (checkNext == '/')		3
	else	scanElementsX(checkNext, this);	1

3		string text;
		text.push_back(check);
		text.push_back(checkNext);
		char charNext;
		charNext = getchar();
			4

4	while (charNext != '\n')		5
	charNext == '\n')		6
5	if (charNext != ' ')	text.push_back(charNext); charNext = getchar();	4
	else	break;	6
6	if (text == text_finish)	break;	7
	else;		1
7		processAccess(this);	Ø

void cl_application::scanElementsX(char nextChar, cl_base* ob_parent)

№ шага	Предикат	Действе	№ перехода
1		string ancestor;	2

		ancestor.push_back(nextChar);
		char charNext;
		charNext = getchar();	

2	while(charNext != '/')		3
	charNext == '/'		4
3	if (charNext != ' ')	ancestor.push_back(charNext); charNext = getchar();	2
	else	break	4
4	if (charNext == ' ')	addNewChild(this);	5
	else		5
5	if (charNext == '/')	doWithChildLink(this);	Ø
	else		Ø

void cl_application::doWithChildLink(cl_base* ob_parent)

№ шага	Предикат	Действе	№ перехода
1		string ancestor; char charNext; charNext = getchar();	2
2	while (charNext != '/')		3
	charNext == '/'		4
3	if (charNext != ' ')	ancestor.push_back(charNext); charNext = getchar();	2
	else	break	4
4		ob_parent->it_child = ob_parent->children.begin();	5
5	while (ob_parent->it_child != ob_parent->children.end())		6
	ob_parent->it_child == ob_parent->children.end()		Ø
6	if (get_object_name((*(ob_parent->it_child))) == ancestor && charNext == '/')	doWithChildLink((*(ob_parent->it_child))); break;	Ø
	else		7
7	if (get_object_name((*(ob_parent->it_child))) == ancestor && charNext == ' ')	addNewChild((*(ob_parent->it_child))); break;	Ø
	else	ob_parent->it_child++;	5

void cl_application::addNewChild(cl_base* ob_parent)

№ шага	Предикат	Действе	№ перехода
1		cl_2* ob_2; cl_3* ob_3; cl_4* ob_4; cl_5* ob_5; cl_6* ob_6; int selectFamily; int state; string nameObject; cin >> nameObject >> selectFamily >> state;	2

2	if (selectFamily == 2)	ob_2 = new cl_2((cl_base*)ob_parent);
		ob_2->set_object_name(nameObject);
		ob_2->set_state(state);
			Ø

	else if (selectFamily == 3)	ob_3 = new cl_3((cl_base*)ob_parent);
		ob_3->set_object_name(nameObject);
		ob_3->set_state(state);
			Ø

	else if (selectFamily == 4)	ob_4 = new cl_4((cl_base*)ob_parent);
		ob_4->set_object_name(nameObject);
		ob_4->set_state(state);
			Ø

	else if (selectFamily == 5)	ob_5 = new cl_5((cl_base*)ob_parent);
		ob_5->set_object_name(nameObject);
		ob_5->set_state(state);
			Ø

	else if (selectFamily == 6)	ob_6 = new cl_6((cl_base*)ob_parent);
		ob_6->set_object_name(nameObject);
		ob_6->set_state(state);
		}
			Ø

	else	return;	Ø

void cl_application::show_object_state()

№ шага	Предикат	Действе	№ перехода
1		show_state_next(this, 0);	2
2		resultLink(this);	Ø

void cl_application::resultLink(cl_base* ob_parent)

№ шага	Предикат	Действе	№ перехода
1	if (ob_parent->childrenLink.size() == 0)	return;	Ø
	else		2
2		ob_parent->it_childLink = ob_parent->childrenLink.begin();	3
3	while (ob_parent-		4

	>it_childLink != ob_parent->childrenLink.end())		
	ob_parent->it_childLink == ob_parent->childrenLink.end()		Ø
4	if (getStateDo((*(ob_parent->it_childLink))))	cout << endl << getLinkName((*(ob_parent->it_childLink))) << " Object name: " << getCheckName((*(ob_parent->it_childLink)));	5
	else	cout << endl << getLinkName((*(ob_parent->it_childLink))) << " Object not found";	5
5		ob_parent->it_childLink++;	3

void cl_application::processAccess(cl_base* ob_parent)

№ шага	Предикат	Действе	№ перехода
1	while (true)	string linkName; cin >> linkName;	2
	false		Ø
2	if (linkName == text_finish2)	break;	Ø
	else	connectWithRoot(0, linkName, ob_parent, false);	1

void cl_application::connectWithRoot(int count, string linkName, cl_base* ob_parent, bool notFound)

№ шага	Предикат	Действе	№ перехода

1		cl_2* ob_2L;
		ob_2L = new cl_2((cl_base*)ob_parent, true);
		ob_2L->setLinkName(linkName);
		ob_2L->setStateDo(false);
			2

2	if (linkName[count] == '/')		3
	else		Ø
3		count++; string name;	4
4	while (linkName[count] != '/')	name.push_back(linkName[count]);	5
	linkName[count] == '/'		6
5	if (count == (linkName.size() - 1))	break;	6
	else	count++;	4

6	if (linkName[count] == '/')		7
	else		17
7	if (name.size() == 0)		8
	else		13
8	if (checkMember(ob_parent))	count++; string name;	9
	else		Ø
9	while (linkName[count] != '/')	name.push_back(linkName[count]);	10
	linkName[count] == '/'		11
10	if (count == (linkName.size() - 1))	break;	11
	else	count++;	9
11	if (checkMemberDeep(this, name))		12
	else		Ø
12	if (getStateDo((cl_2*)ob_2L) == false)	ob_2L->setCheckName(name); ob_2L->setStateDo(true);	Ø
13	else		Ø
	if (name == get_object_name(ob_parent))		14
	else		Ø
14	if (checkMember(ob_parent))	ob_parent->it_child = ob_parent->children.begin();	15
	else		Ø
15	while (ob_parent->it_child != ob_parent->children.end())	connectAccess(count, linkName, ob_2L, (*(ob_parent->it_child)));	16
	ob_parent->it_child == ob_parent->children.end()		Ø
16		ob_parent->it_child++;	15
17	if (name == get_object_name(ob_parent))		18
	else		Ø
18	if (getStateDo((cl_2*)ob_2L) == false)	ob_2L->setCheckName(name); ob_2L->setStateDo(true);	Ø
	else		Ø

void cl_application::connectAccess(int count, string linkName, cl_2* ob_2L, cl_base* ob_parent)

№ шага	Предикат	Действе	№ перехода
1		count++; string name;	2
2	while (linkName[count] != '/')	name.push_back(linkName[count]);	3
	linkName[count] == '/'		4
3	if (count == (linkName.size() - 1))	break;	4
	else	count++;	2

4	if (linkName[count] == '/')		5
	else		15
5	if (name.size() == 0)		6
	else		11
6	if (checkMember(ob_parent))	count++; string name;	7
	else		Ø
7	while (linkName[count] != '/')	name.push_back(linkName[count]);	8
	linkName[count] == '/'		9
8	if (count == (linkName.size() - 1))	break;	9
	else	count++;	7
9	if (checkMemberDeep(this, name))		10
	else		Ø
10	if (getStateDo((cl_2*)ob_2L) == false)	ob_2L->setCheckName(name); ob_2L->setStateDo(true);	Ø
	else		Ø
11	if (name == get_object_name(ob_parent))		12
	else		Ø
12	if (checkMember(ob_parent))	ob_parent->it_child = ob_parent->children.begin();	13
	else		Ø
13	while (ob_parent->it_child != ob_parent->children.end())	connectAccess(count, linkName, ob_2L, (*(ob_parent->it_child)));	14
	ob_parent->it_child == ob_parent->children.end()		Ø
14		ob_parent->it_child++;	13
15	if (name == get_object_name(ob_parent))		16
	else		Ø
16	if (getStateDo((cl_2*)ob_2L) == false)	ob_2L->setCheckName(name); ob_2L->setStateDo(true);	Ø
	else		Ø

bool cl_application::checkMemberDeep(cl_base* ob_parent, string name)

№ шага	Предикат	Действе	№ перехода
1		bool result = false;	2

		ob_parent->it_child = ob_parent->children.begin();	
2	while (ob_parent->it_child != ob_parent->children.end())		3
	ob_parent->it_child == ob_parent->children.end()		6
3	if (name == get_object_name(*(ob_parent->it_child)))	result = true;	4
	else		4
4	if (checkMemberDeep((*(ob_parent->it_child)), name))	result = true;	5
	else		5
5		ob_parent->it_child++;	2
6		return result;	Ø

bool cl_application::checkMember(cl_base* ob_parent)

№ шага	Предикат	Действе	№ перехода
1	if (ob_parent->children.size() == 0)	return false;	Ø
	else	return true;	Ø

void cl_base::setLinkName(string linkName)

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

string cl_base::getLinkName(cl_base* p_parent)

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

void cl_base::setCheckName(string checkName)

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

string cl_base::getCheckName(cl_base* p_parent)

№ шага	Предикат	Действе	№ перехода
1		return p_parent-	Ø

		>checkName;	

void cl_base::addChildLink(cl_base* p_child)

№ шага	Предикат	Действе	№ перехода
1		childrenLink.push_back(p_child); 	Ø

void cl_base::setStateDo(bool stateDo)

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

bool cl_base::getStateDo(cl_base* p_parent)

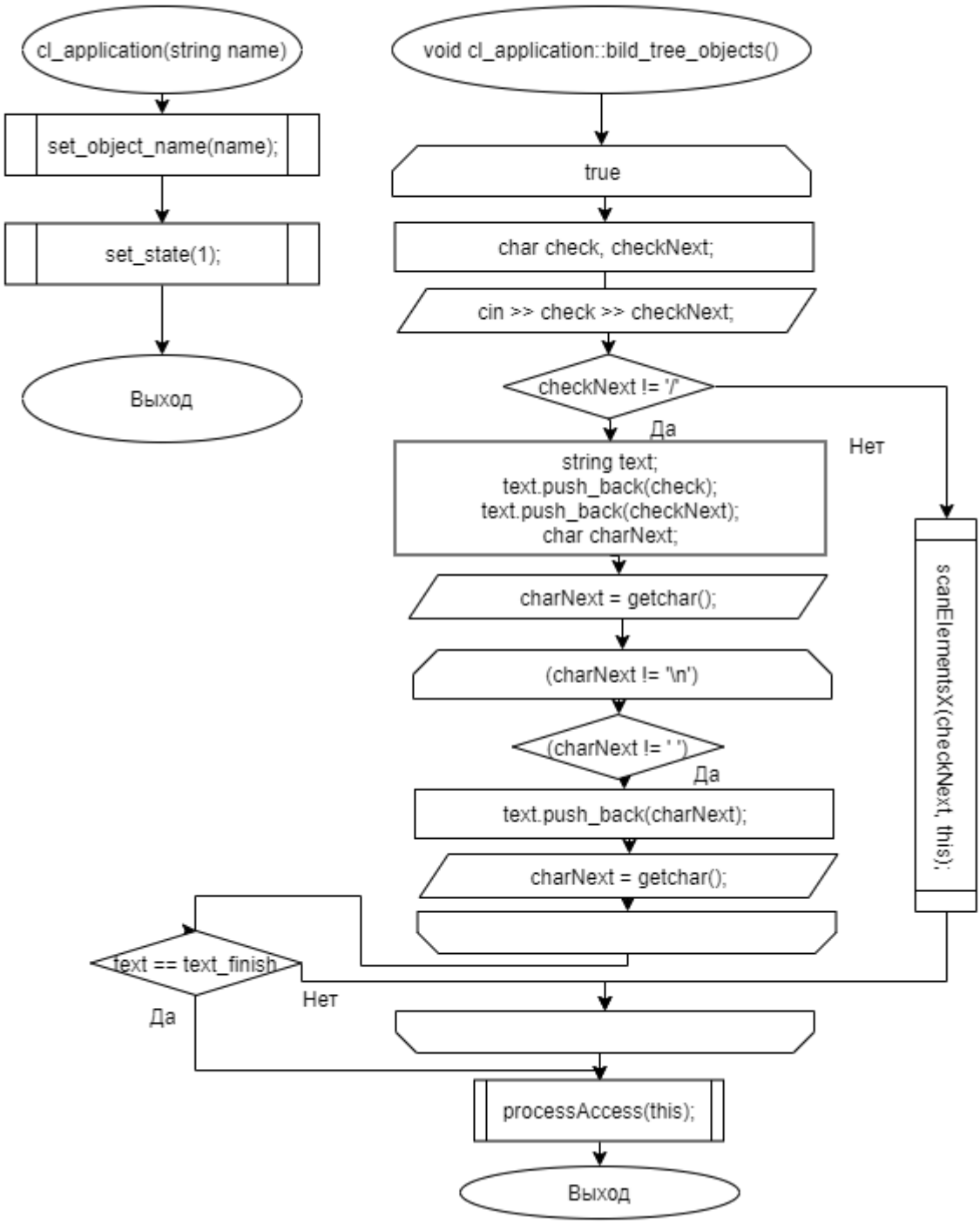
№ шага	Предикат	Действе	№ перехода
1		return p_parent->stateDo;	Ø

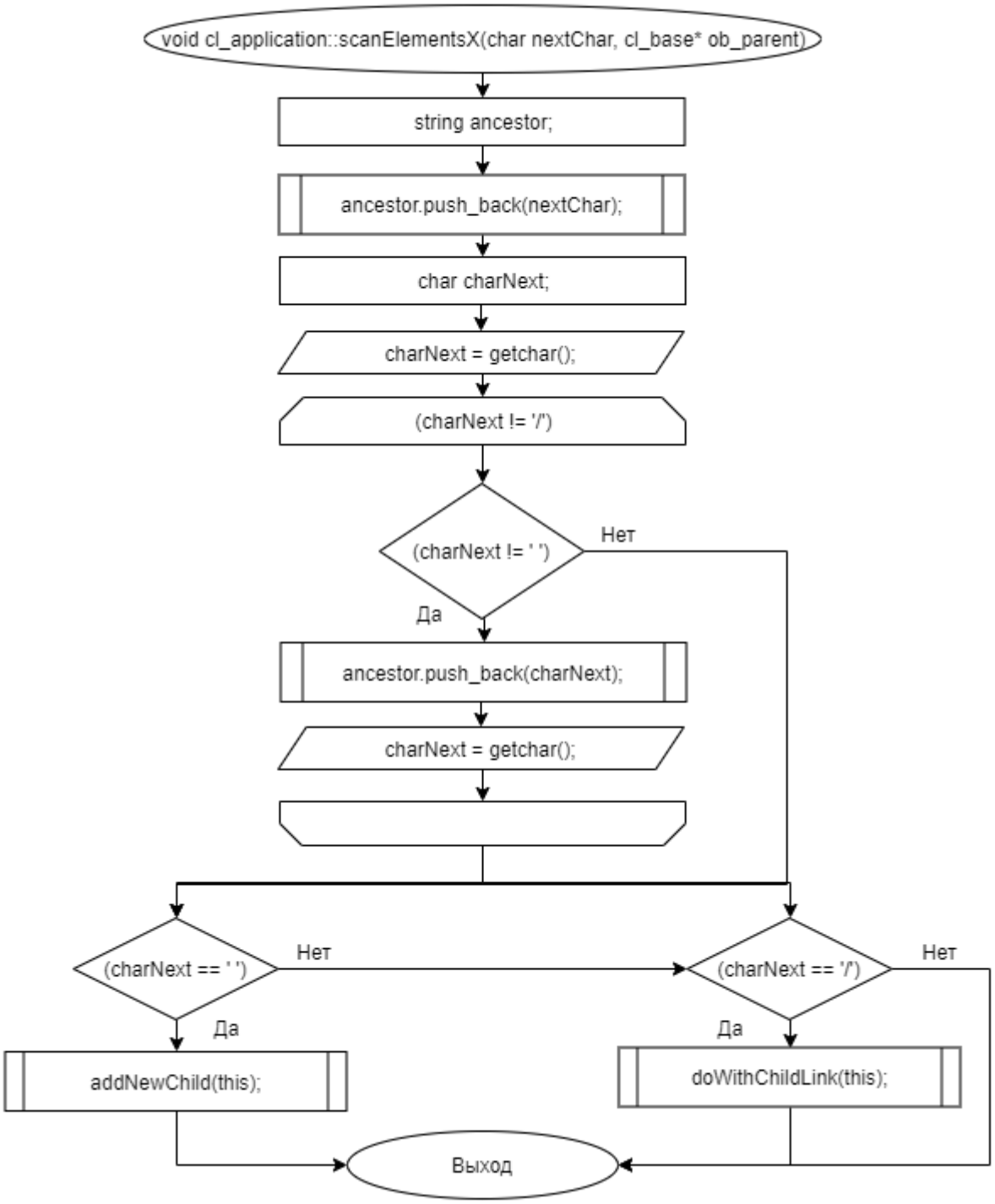
int main()

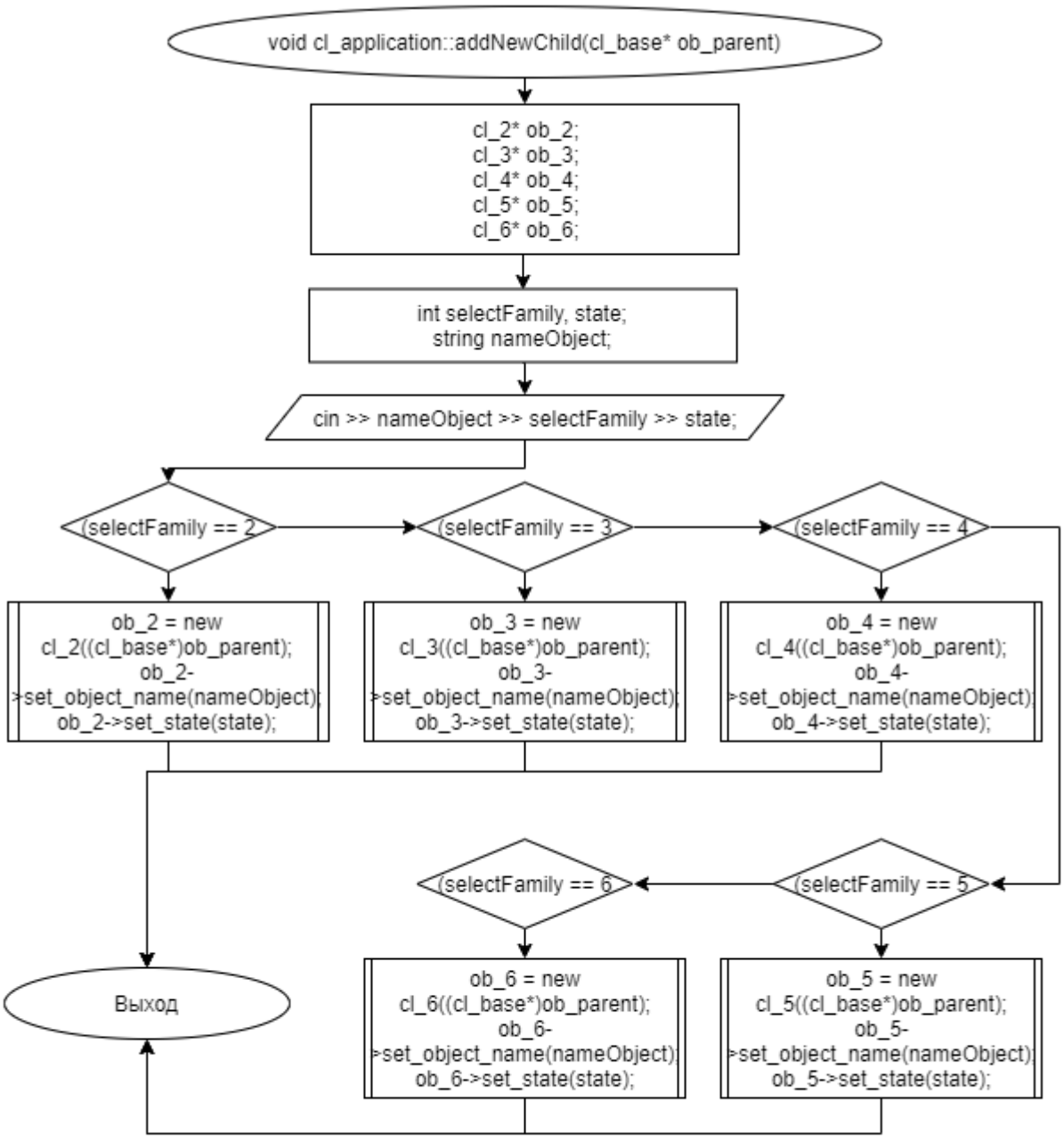
№ шага	Предикат	Действе	№ перехода

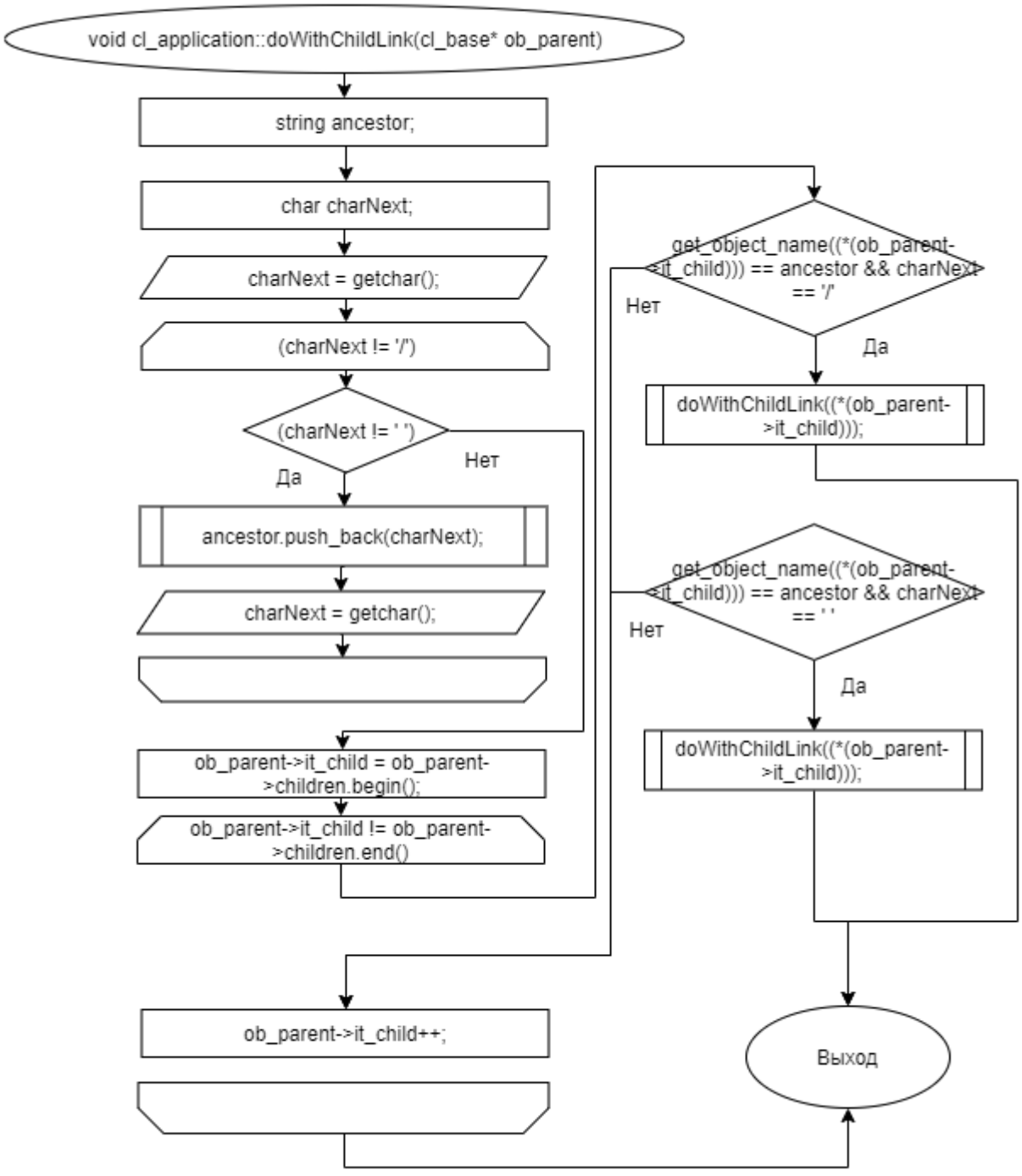
1		string name;
		cin >> name;
		
		cl_application ob_application(name);
		
		ob_application.bild_tree_objects();
		
		cout << "Object tree";
		
		return ob_application.exec_app();
			Ø

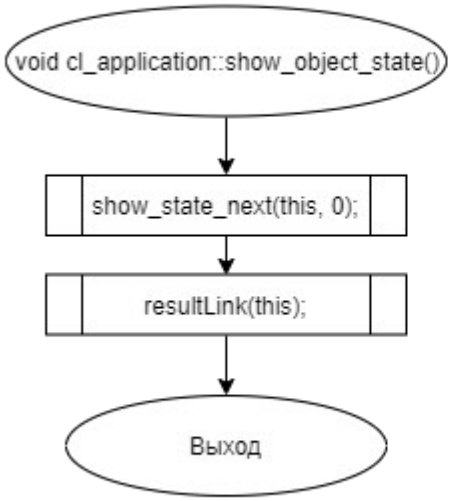
Блок-схема алгоритма

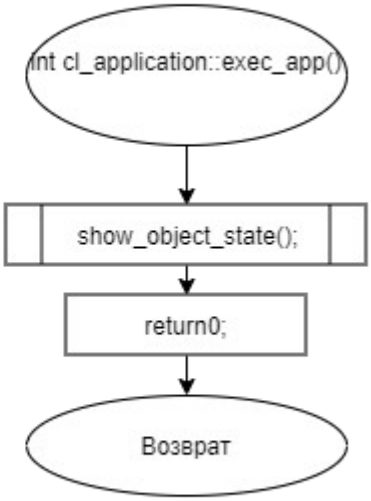

cl_application(string name)
set_object_name(name);
set_state(1);
Выход
void cl_application::bild_tree_objects()
true
char check, checkNext;
cin >> check >> checkNext;
checkNext != '/'
Да
Нет
string text;
text.push_back(check);
text.push_back(checkNext);
char charNext;
charNext = getchar();
(charNext != '\n')
(charNext != ' ')
Да
text.push_back(charNext);
charNext = getchar();
scanElementsX(checkNext, this);
text == text_finish
Да
Нет
processAccess(this);
Выход

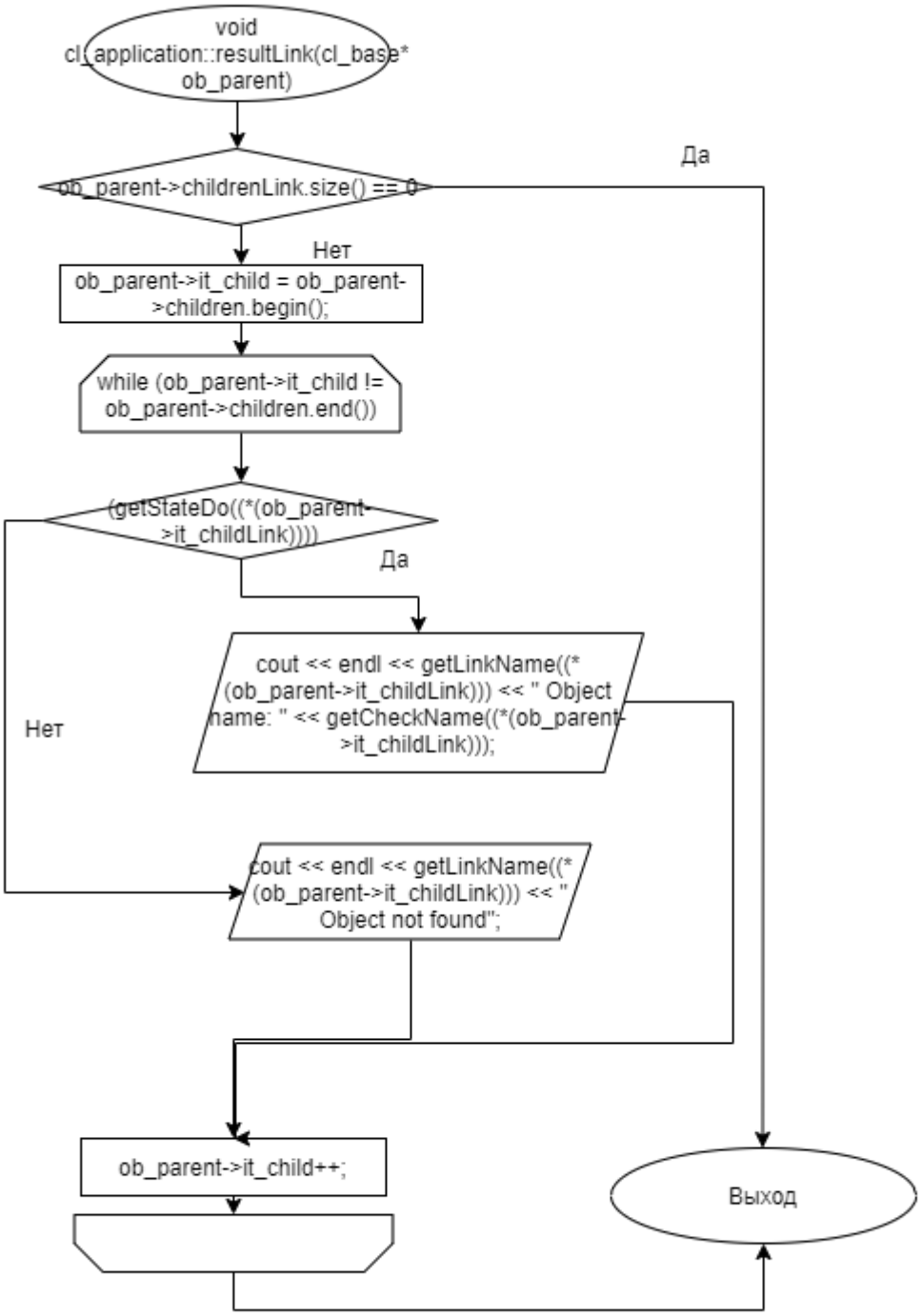

void cl_application::scanElementsX(char nextChar, cl_base* ob_parent)
string ancestor;
ancestor.push_back(nextChar);
char charNext;
charNext = getchar();
(charNext != '/')
(charNext != ' ')
Нет
Да
ancestor.push_back(charNext);
charNext = getchar();
(charNext == ' ')
Нет
Да
addNewChild(this);
(charNext == '/')
Нет
Да
doWithChildLink(this);
Выход

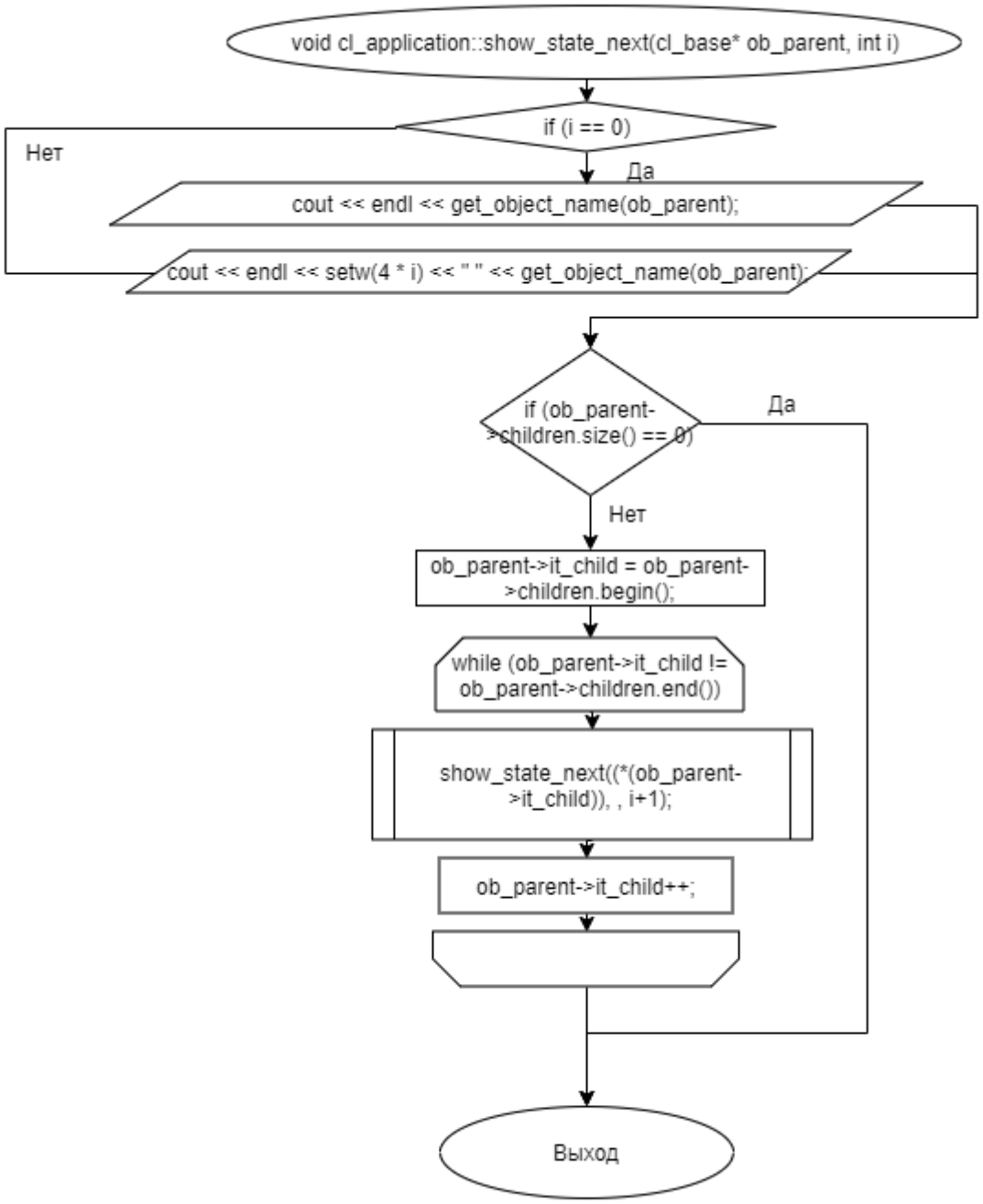

void cl_application::addNewChild(cl_base* ob_parent)
cl_2* ob_2;
cl_3* ob_3;
cl_4* ob_4;
cl_5* ob_5;
cl_6* ob_6;
int selectFamily, state;
string nameObject;
cin >> nameObject >> selectFamily >> state;
(selectFamily == 2
(selectFamily == 3
(selectFamily == 4
ob_2 = new cl_2((cl_base*)ob_parent); ob_2->set_object_name(nameObject); ob_2->set_state(state);
ob_3 = new cl_3((cl_base*)ob_parent); ob_3->set_object_name(nameObject); ob_3->set_state(state);
ob_4 = new cl_4((cl_base*)ob_parent); ob_4->set_object_name(nameObject); ob_4->set_state(state);
(selectFamily == 6
(selectFamily == 5
Выход
ob_6 = new cl_6((cl_base*)ob_parent); ob_6->set_object_name(nameObject); ob_6->set_state(state);
ob_5 = new cl_5((cl_base*)ob_parent); ob_5->set_object_name(nameObject); ob_5->set_state(state);

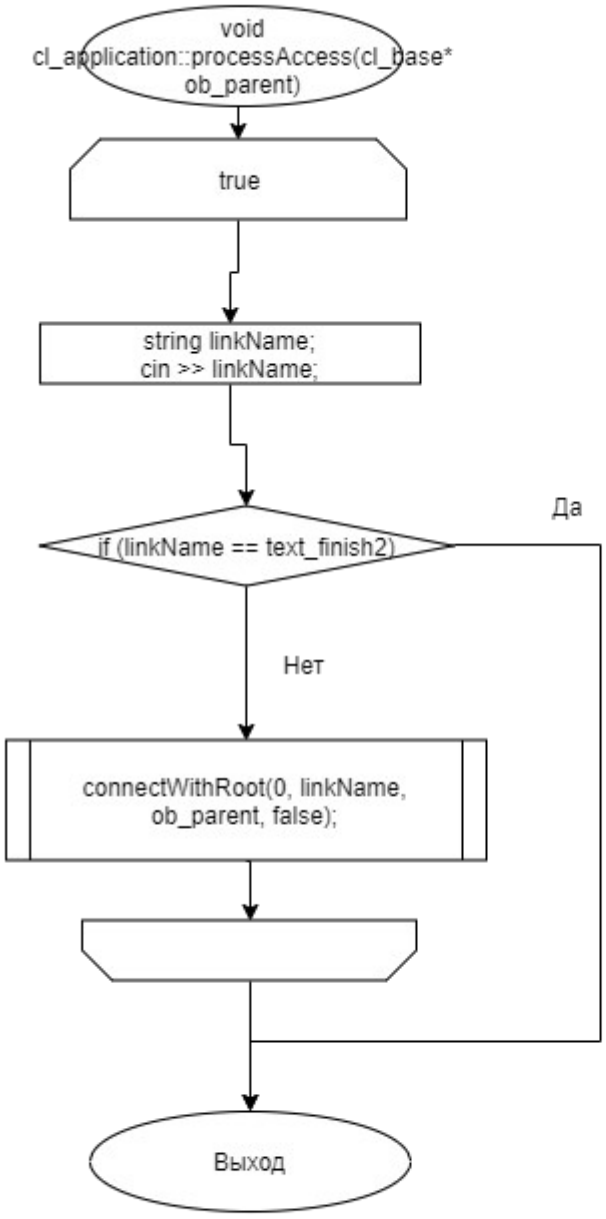

void cl_application::doWithChildLink(cl_base* ob_parent)
string ancestor;
char charNext;
charNext = getchar();
(charNext != '/')
(charNext != ' ')
Да
Нет
ancestor.push_back(charNext);
charNext = getchar();
ob_parent->it_child = ob_parent->children.begin();
ob_parent->it_child != ob_parent->children.end()
ob_parent->it_child++;
get_object_name((*(ob_parent->it_child))) == ancestor && charNext == '/'
Нет
Да
doWithChildLink((*(ob_parent->it_child)));
get_object_name((*(ob_parent->it_child))) == ancestor && charNext == ' '
Нет
Да
doWithChildLink((*(ob_parent->it_child)));
Выход

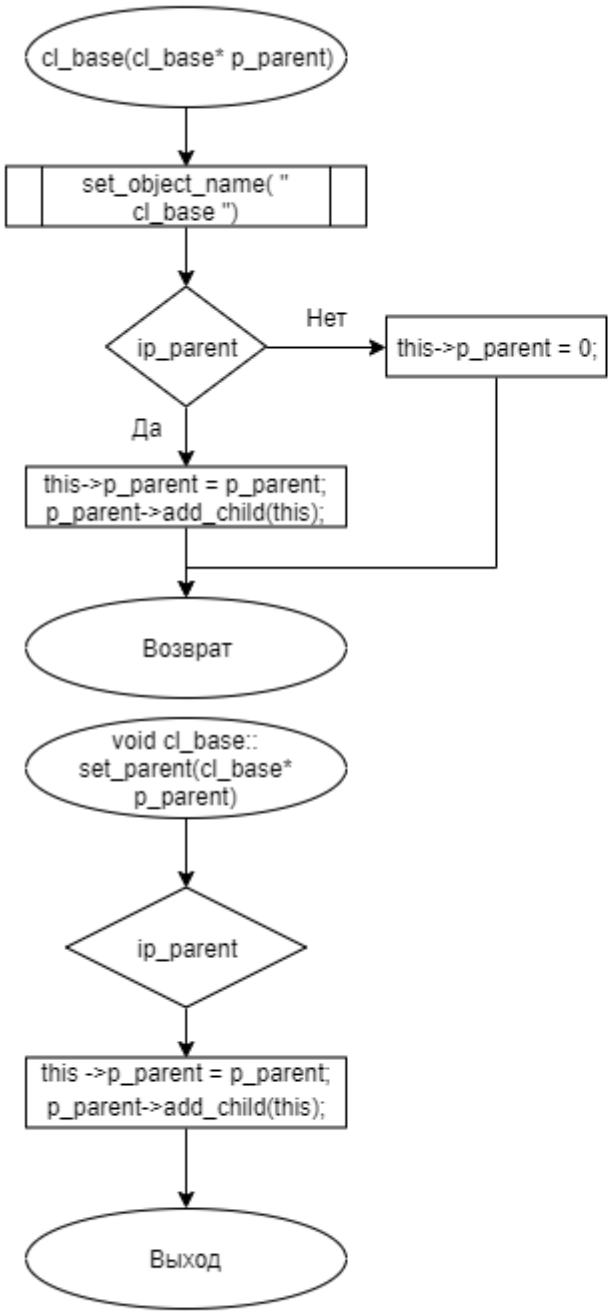

void cl_application::show_object_state()
show_state_next(this, 0);
resultLink(this);
Выход

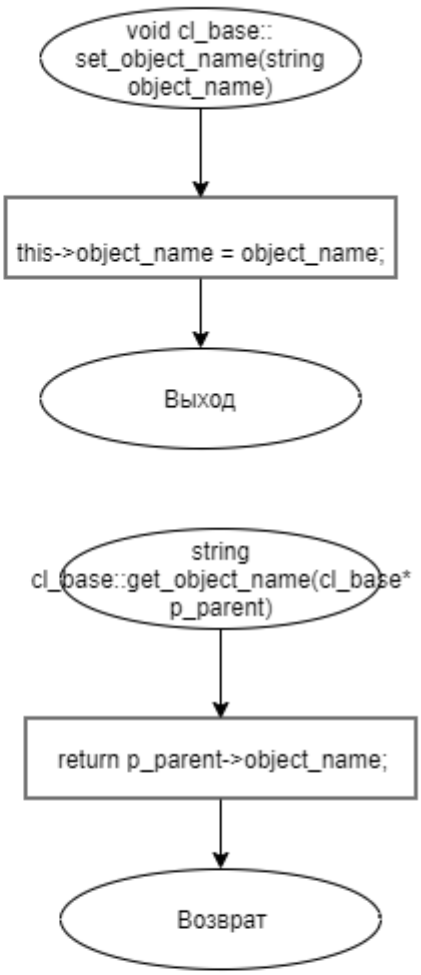

int cl_application::exec_app()
show_object_state();
return0;
Возврат

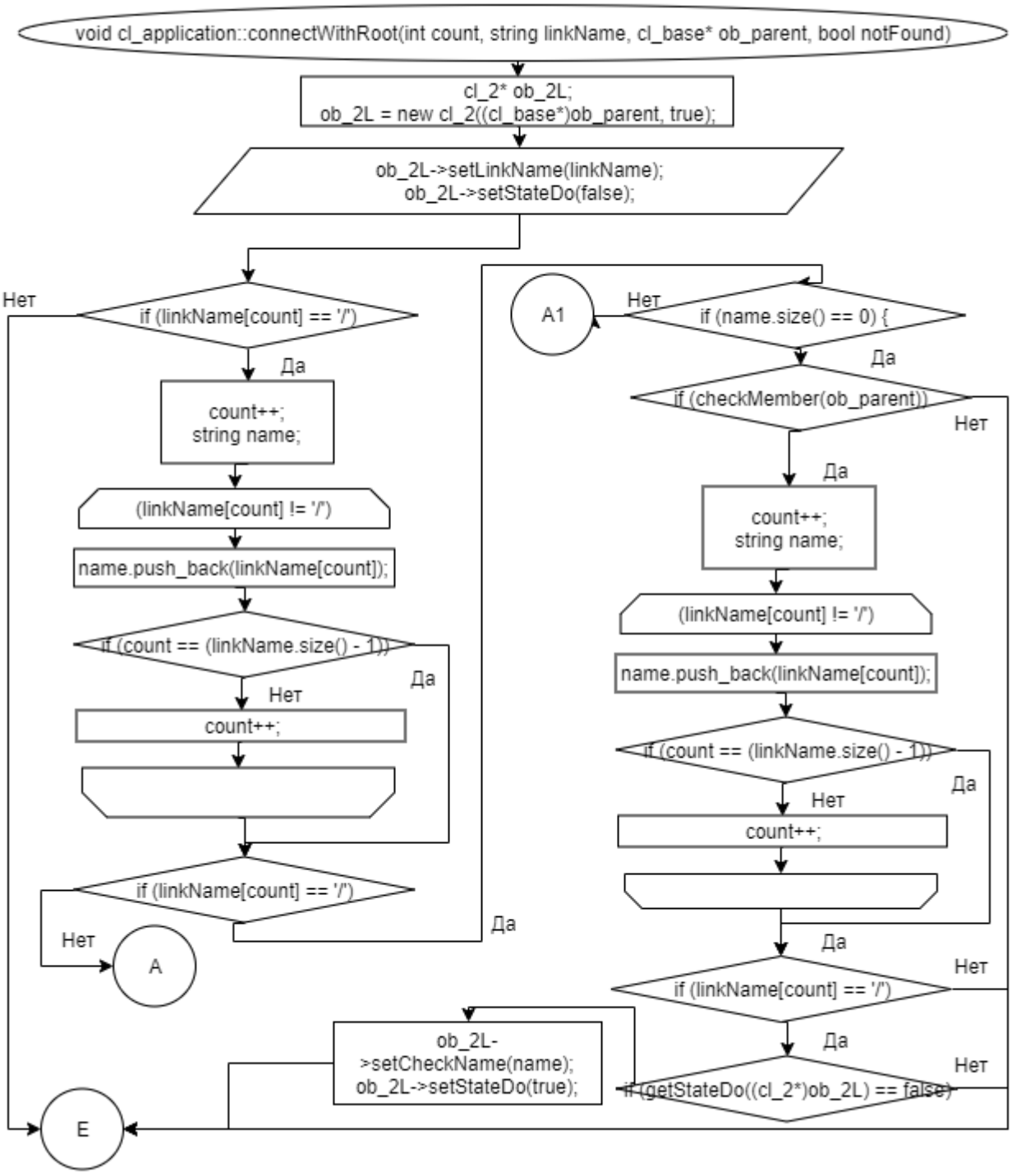

void cl_application::resultLink(cl_base* ob_parent)
ob_parent->childrenLink.size() == 0
Да
Нет
ob_parent->it_child = ob_parent->children.begin();
while (ob_parent->it_child != ob_parent->children.end())
(getStateDo((*(ob_parent->it_childLink))))
Да
Нет
cout << endl << getLinkName((*(ob_parent->it_childLink))) << " Object name: " << getCheckName((*(ob_parent->it_childLink)));
cout << endl << getLinkName((*(ob_parent->it_childLink))) << " Object not found";
ob_parent->it_child++;
Выход

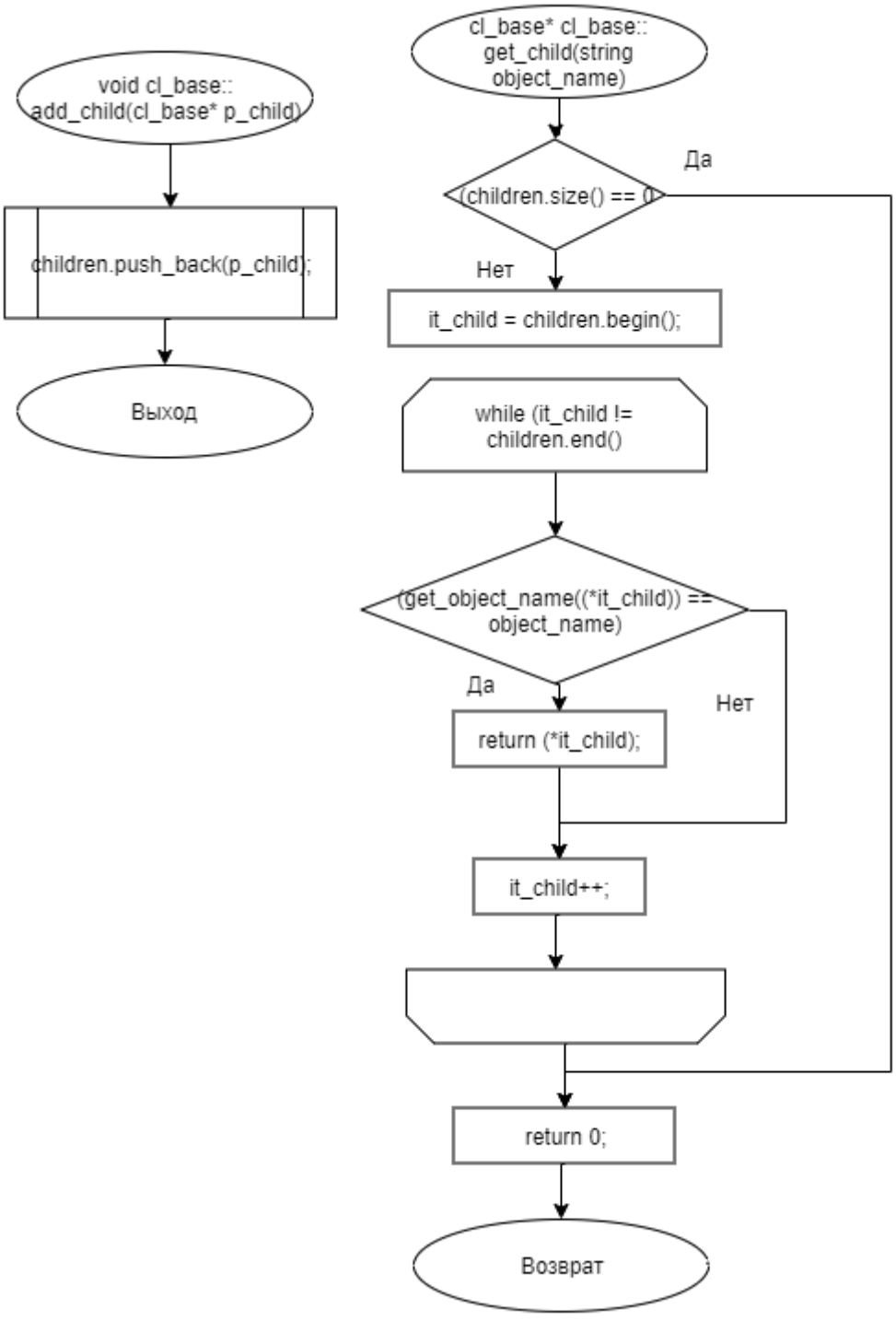

void cl_application::show_state_next(cl_base* ob_parent, int i)
if (i == 0)
Нет
Да
cout << endl << get_object_name(ob_parent);
cout << endl << setw(4 * i) << " " << get_object_name(ob_parent);
if (ob_parent->children.size() == 0)
Да
Нет
ob_parent->it_child = ob_parent->children.begin();
while (ob_parent->it_child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

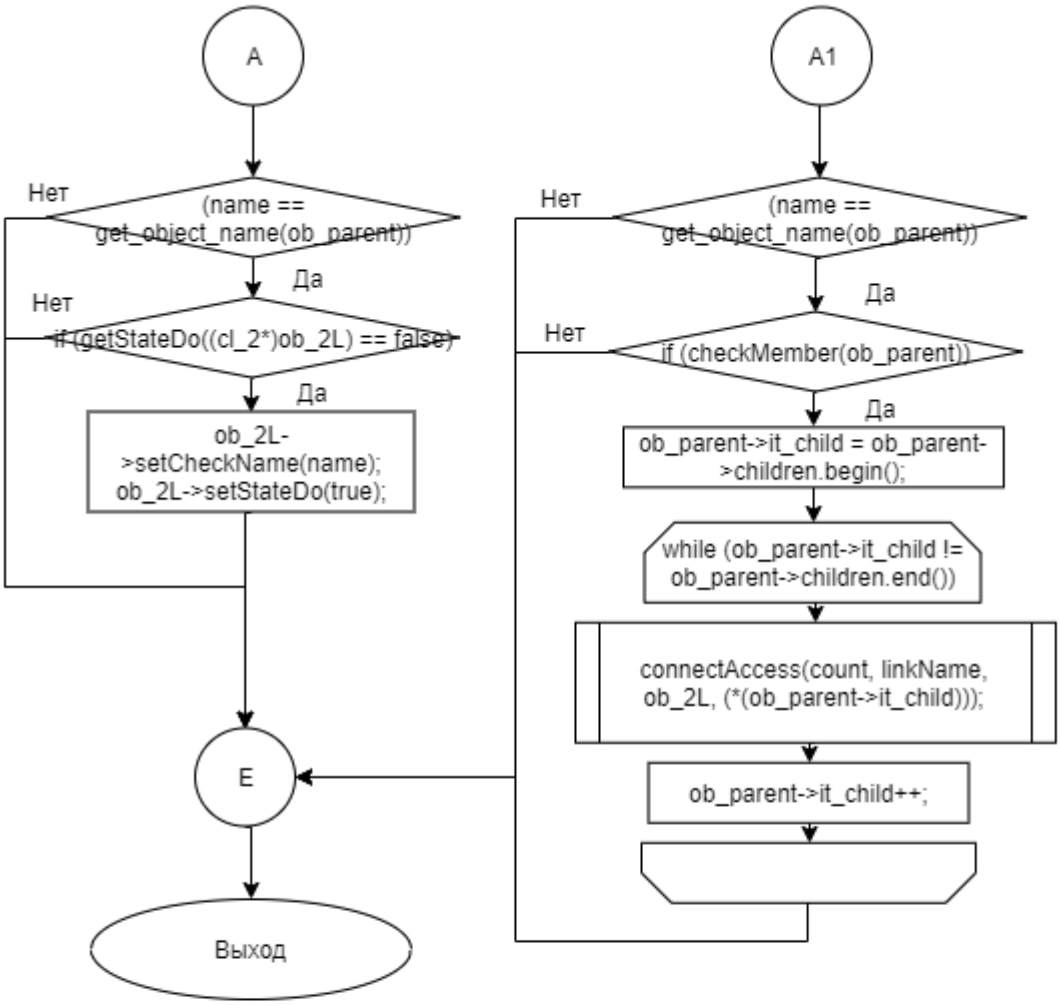

void
cl_application::processAccess(cl_base*
ob_parent)
true
string linkName;
cin >> linkName;
if (linkName == text_finish2)
Да
Нет
connectWithRoot(0, linkName,
ob_parent, false);
Выход

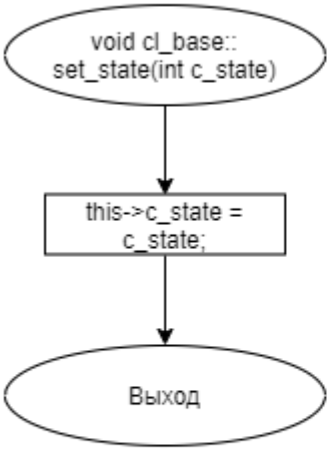

cl_base(cl_base* p_parent)
set_object_name(" cl_base ")
ip_parent
Нет
this->p_parent = 0;
Да
this->p_parent = p_parent;
p_parent->add_child(this);
Возврат
void cl_base::
set_parent(cl_base*
p_parent)
ip_parent
this ->p_parent = p_parent;
p_parent->add_child(this);
Выход

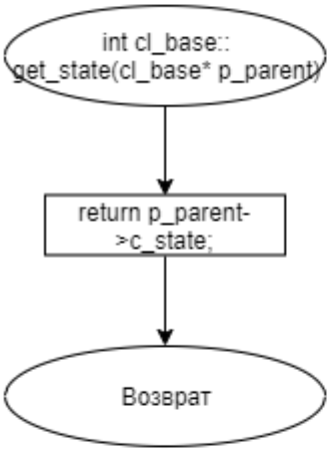

void cl_base::
set_object_name(string
object_name)
this->object_name = object_name;
Выход
string
cl_base::get_object_name(cl_base*
p_parent)
return p_parent->object_name;
Возврат

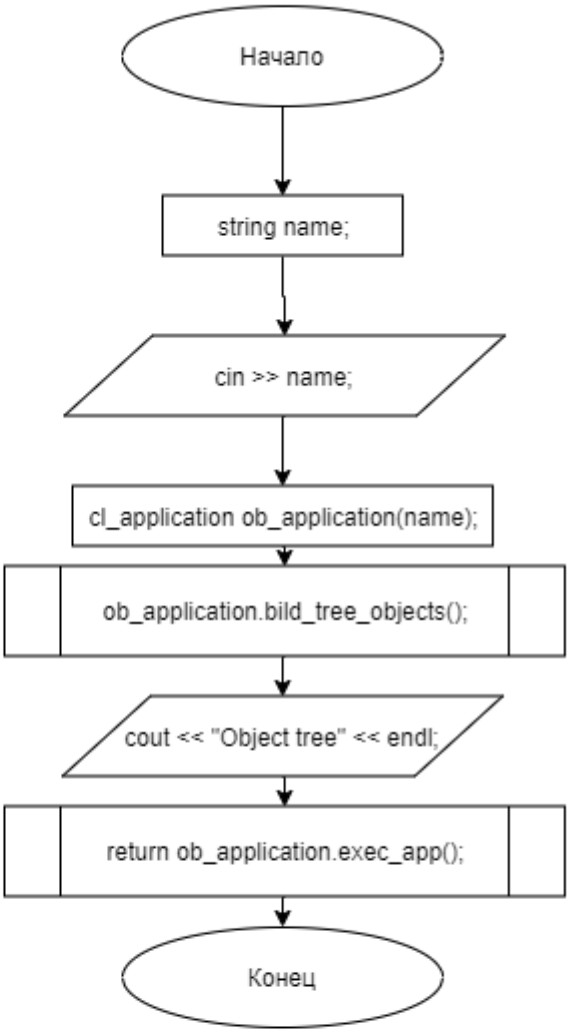

void cl_application::connectWithRoot(int count, string linkName, cl_base* ob_parent, bool notFound)
cl_2* ob_2L;
ob_2L = new cl_2((cl_base*)ob_parent, true);
ob_2L->setLinkName(linkName);
ob_2L->setStateDo(false);
if (linkName[count] == '/')
Нет
Да
count++;
string name;
(linkName[count] != '/')
name.push_back(linkName[count]);
if (count == (linkName.size() - 1))
Да
Нет
count++;
if (linkName[count] == '/')
Нет
Да
A
A1
if (name.size() == 0) {
Нет
Да
if (checkMember(ob_parent))
Нет
Да
count++;
string name;
(linkName[count] != '/')
name.push_back(linkName[count]);
if (count == (linkName.size() - 1))
Да
Нет
count++;
Да
if (linkName[count] == '/')
Нет
Да
if (getStateDo((cl_2*)ob_2L) == false)
Нет
ob_2L-
>setCheckName(name);
ob_2L->setStateDo(true);
E

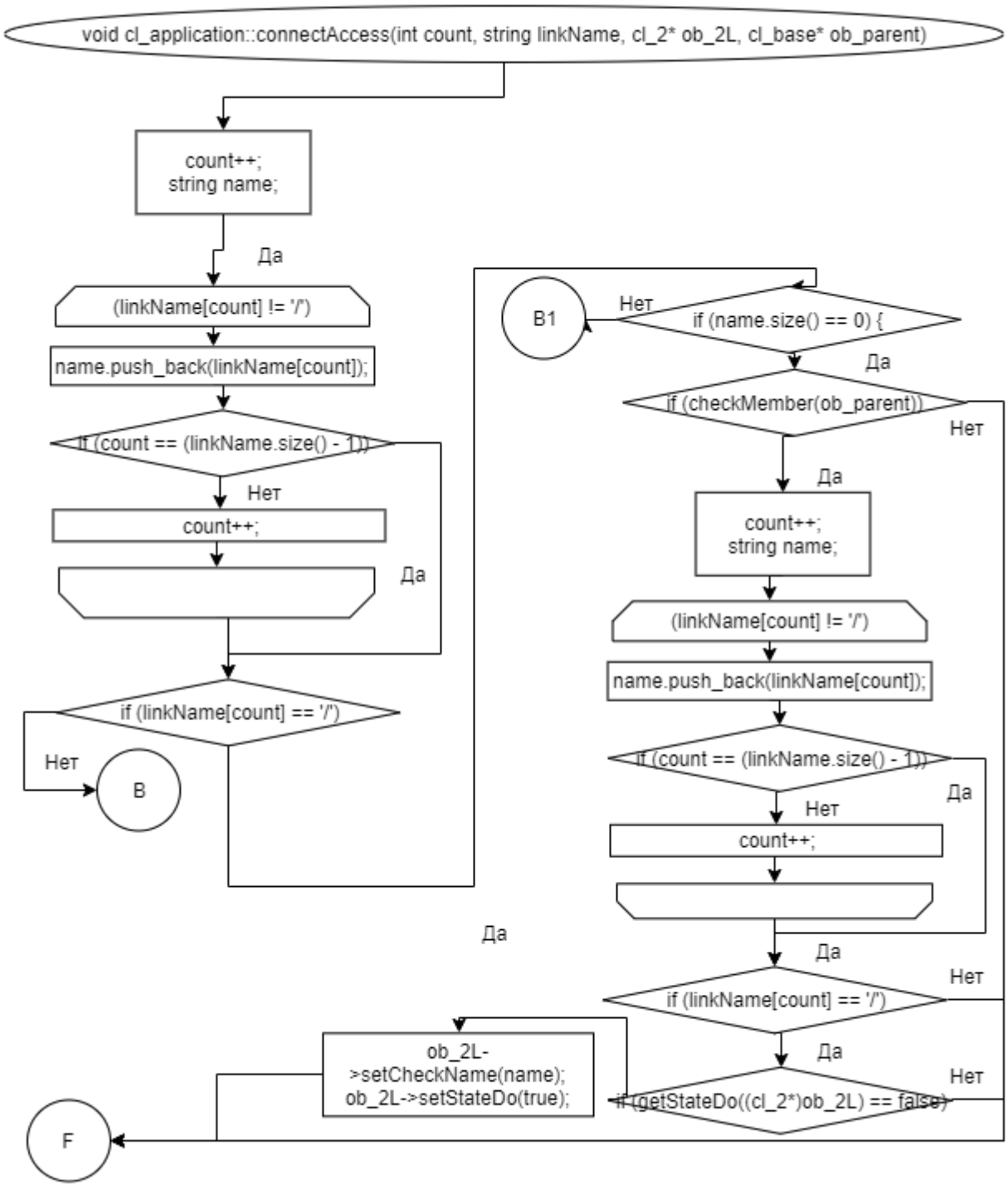

void cl_base::
add_child(cl_base* p_child)
children.push_back(p_child);
Выход
cl_base* cl_base::
get_child(string
object_name)
(children.size() == 0)
Да
Нет
it_child = children.begin();
while (it_child !=
children.end()
(get_object_name((*it_child)) ==
object_name)
Да
Нет
return (*it_child);
it_child++;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

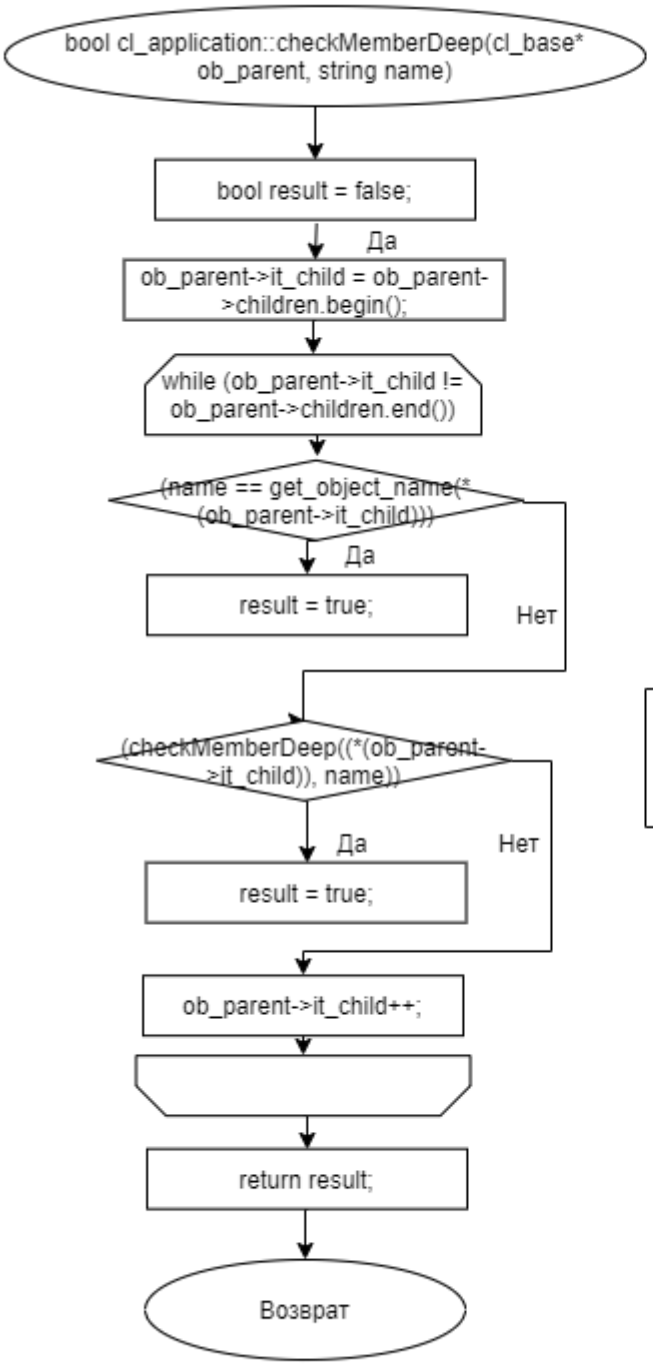

A
Нет
(name ==
get_object_name(ob_parent))
Да
Нет
if (getStateDo((cl_2*)ob_2L) == false)
Да
ob_2L-
>setCheckName(name);
ob_2L->setStateDo(true);
E
Выход
A1
Нет
(name ==
get_object_name(ob_parent))
Да
Нет
if (checkMember(ob_parent))
Да
ob_parent->it_child = ob_parent-
>children.begin();
while (ob_parent->it_child !=
ob_parent->children.end())
connectAccess(count, linkName,
ob_2L, (*(ob_parent->it_child)));
ob_parent->it_child++;

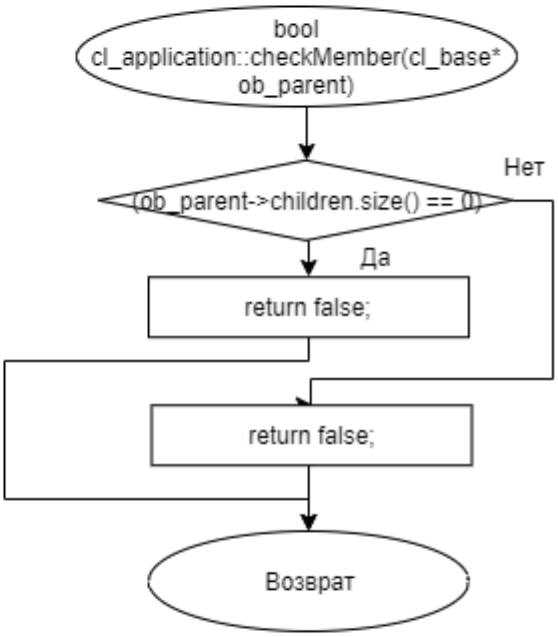

void cl_base::
set_state(int c_state)
this->c_state =
c_state;
Выход

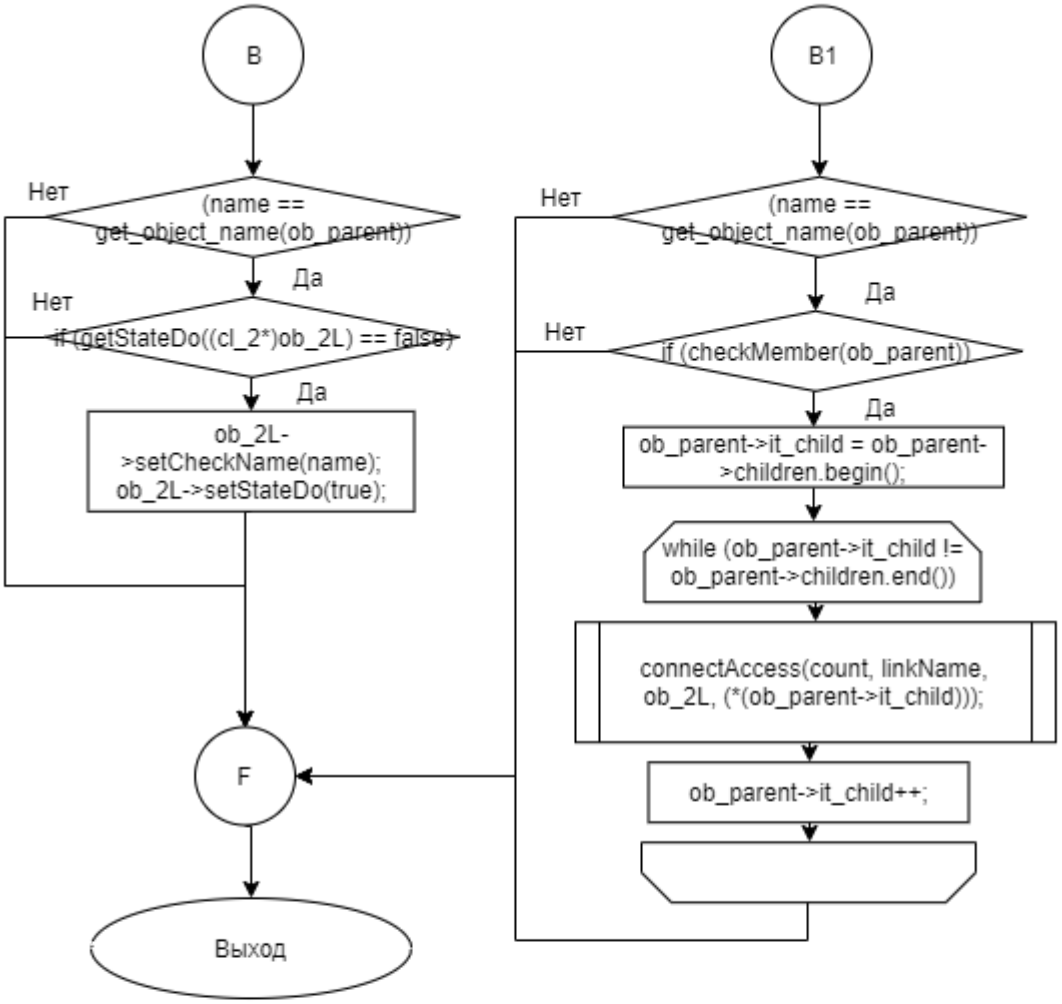

int cl_base::
get_state(cl_base* p_parent)
return p_parent-
>c_state;
Возврат

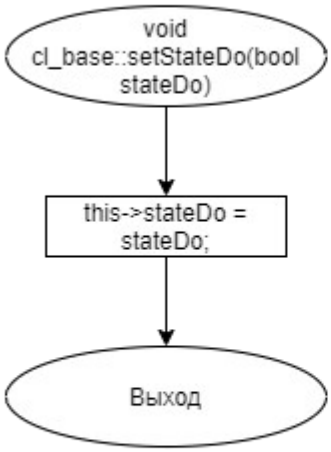

Начало
string name;
cin >> name;
cl_application ob_application(name);
ob_application.bild_tree_objects();
cout << "Object tree" << endl;
return ob_application.exec_app();
Конец

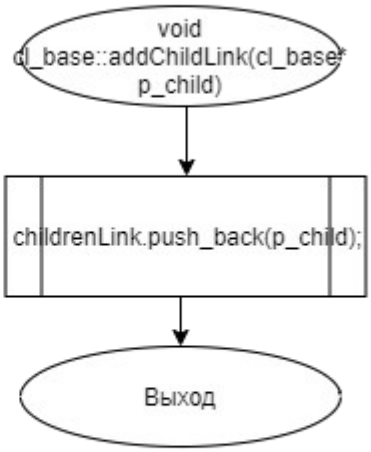

void cl_application::connectAccess(int count, string linkName, cl_2* ob_2L, cl_base* ob_parent)
count++;
string name;
Да
(linkName[count] != '/')
name.push_back(linkName[count]);
if (count == (linkName.size() - 1))
Нет
count++;
Да
if (linkName[count] == '/')
Нет
B
B1
Нет
if (name.size() == 0) {
Да
if (checkMember(ob_parent))
Нет
Да
count++;
string name;
(linkName[count] != '/')
name.push_back(linkName[count]);
if (count == (linkName.size() - 1))
Да
Нет
count++;
Да
Да
if (linkName[count] == '/')
Нет
Да
ob_2L->setCheckName(name);
ob_2L->setStateDo(true);
if (getStateDo((cl_2*)ob_2L) == false)
Нет
F

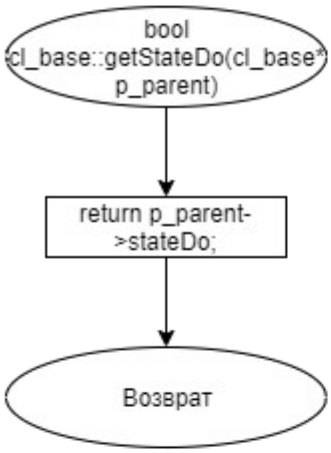

bool cl_application::checkMemberDeep(cl_base* ob_parent, string name)
bool result = false;
Да
ob_parent->it_child = ob_parent->children.begin();
while (ob_parent->it_child != ob_parent->children.end())
(name == get_object_name(*(ob_parent->it_child)))
Да
result = true;
Нет
(checkMemberDeep((*(ob_parent->it_child)), name))
Да
Нет
result = true;
ob_parent->it_child++;
return result;
Возврат

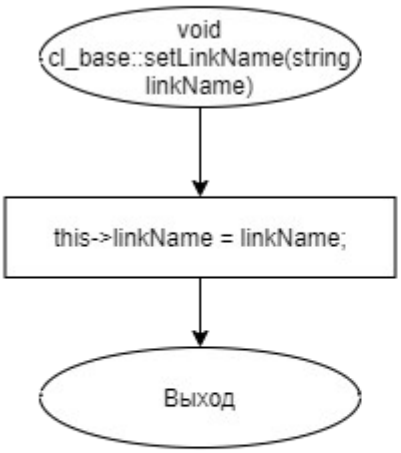

bool cl_application::checkMember(cl_base* ob_parent)
(ob_parent->children.size() == 0)
Нет
Да
return false;
return false;
Возврат

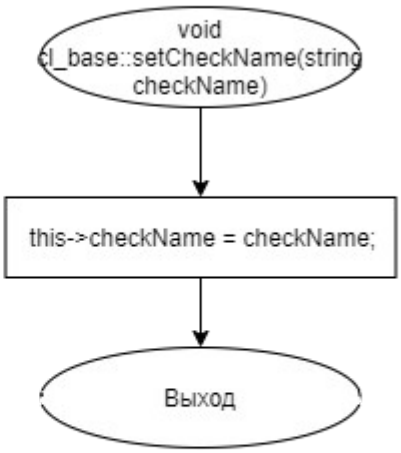

B
Нет
(name ==
get_object_name(ob_parent))
Да
Нет
if (getStateDo((cl_2*)ob_2L) == false)
Да
ob_2L-
>setCheckName(name);
ob_2L->setStateDo(true);
F
Выход
B1
Нет
(name ==
get_object_name(ob_parent))
Да
Нет
if (checkMember(ob_parent))
Да
ob_parent->it_child = ob_parent-
>children.begin();
while (ob_parent->it_child !=
ob_parent->children.end())
connectAccess(count, linkName,
ob_2L, (*(ob_parent->it_child)));
ob_parent->it_child++;

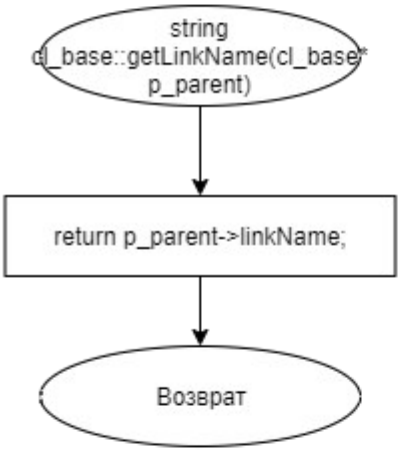

void cl_base::setStateDo(bool stateDo)
this->stateDo = stateDo;
Выход

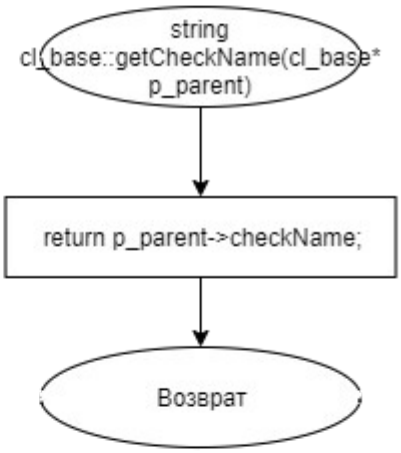

void cl_base::addChildLink(cl_base* p_child)
childrenLink.push_back(p_child);
Выход


bool cl_base::getStateDo(cl_base* p_parent)
return p_parent->stateDo;
Возврат


void cl_base::setLinkName(string linkName)
this->linkName = linkName;
Выход


void cl_base::setCheckName(string checkName)
this->checkName = checkName;
Выход


string cl_base::getLinkName(cl_base* p_parent)
return p_parent->linkName;
Возврат


string cl_base::getCheckName(cl_base* p_parent)
return p_parent->checkName;
Возврат

Код программы

Файл cl_2.cpp

```
#include "cl_2.h"

cl_2::cl_2(cl_base* p_parent) : cl_base(p_parent) {}
cl_2::cl_2(cl_base* p_parent, bool checkLink) : cl_base(p_parent, true) {}
```

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);
cl_2(cl_base* p_parent, bool checkLink);
};

#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() {
	while (true)
	{
		char check, checkNext;
		cin >> check >> checkNext;
		if (check != '/') {
			string text;
			text.push_back(check);
			text.push_back(checkNext);
			char charNext;
			charNext = getchar();
			while (charNext != '\n') {
				if (charNext != ' ') {
					text.push_back(charNext);
					charNext = getchar();
				}
				else
					break;

			}
			if (text == text_finish) {
				break;
			}
		}
		else {
			scanElementsX(checkNext, this);
		}
	}
	processAccess(this);

}
void cl_application::scanElementsX(char nextChar, cl_base* ob_parent) {

	string ancestor;
	ancestor.push_back(nextChar);
	char charNext;
	charNext = getchar();
	while (charNext != '/') {
		if (charNext != ' ') {
			ancestor.push_back(charNext);
			charNext = getchar();

		}
		else break;
	}
	if (charNext == ' ') {
		addNewChild(this);
	}
	if (charNext == '/') {
```

```
                doWithChildLink(this);
        }
}
void cl_application::doWithChildLink(cl_base* ob_parent) {
        string ancestor;
        char charNext;
        charNext = getchar();
        while (charNext != '/') {
                if (charNext != ' ') {
                        ancestor.push_back(charNext);
                        charNext = getchar();

                }
                else break;
        }

        ob_parent->it_child = ob_parent->children.begin();
        while (ob_parent->it_child != ob_parent->children.end()) {
                if (get_object_name((*(ob_parent->it_child))) == ancestor &&
charNext == '/') {
                        doWithChildLink((*(ob_parent->it_child)));
                        break;
                }
                if (get_object_name((*(ob_parent->it_child))) == ancestor &&
charNext == ' ') {
                        addNewChild((*(ob_parent->it_child)));
                        break;
                }
                ob_parent->it_child++;
        }
}
void cl_application::addNewChild(cl_base* ob_parent) {

        cl_2* ob_2;
        cl_3* ob_3;
        cl_4* ob_4;
        cl_5* ob_5;
        cl_6* ob_6;
        int selectFamily;
        int state;
        string nameObject;
        cin >> nameObject >> selectFamily >> state;

        if (selectFamily == 2) {
                ob_2 = new cl_2((cl_base*)ob_parent);
                ob_2->set_object_name(nameObject);
                ob_2->set_state(state);
        }
        else if (selectFamily == 3) {
                ob_3 = new cl_3((cl_base*)ob_parent);
                ob_3->set_object_name(nameObject);
                ob_3->set_state(state);
        }
        else if (selectFamily == 4) {
                ob_4 = new cl_4((cl_base*)ob_parent);
                ob_4->set_object_name(nameObject);
                ob_4->set_state(state);
        }
        else if (selectFamily == 5) {
                ob_5 = new cl_5((cl_base*)ob_parent);
```

```
                ob_5->set_object_name(nameObject);
                ob_5->set_state(state);
        }
        else if (selectFamily == 6) {
                ob_6 = new cl_6((cl_base*)ob_parent);
                ob_6->set_object_name(nameObject);
                ob_6->set_state(state);
        }
        else return;
}
void cl_application::resultLink(cl_base* ob_parent) {
if (ob_parent->childrenLink.size() == 0) {
        return;
}
ob_parent->it_childLink = ob_parent->childrenLink.begin();
while (ob_parent->it_childLink != ob_parent->childrenLink.end()) {
        if (getStateDo((*(ob_parent->it_childLink)))) {
                cout << endl << getLinkName((*(ob_parent->it_childLink))) << "
Object name: " << getCheckName((*(ob_parent->it_childLink)));
        }
        else {
                cout << endl << getLinkName((*(ob_parent->it_childLink))) << "
Object not found";
        }
        ob_parent->it_childLink++;
}
}
void cl_application::processAccess(cl_base* ob_parent) {
        while (true) {
                string linkName;
                cin >> linkName;
                if (linkName == text_finish2) {
                        break;
                }
                else {
                        connectWithRoot(0, linkName, ob_parent, false);
                }
        }
}
void cl_application::connectWithRoot(int count, string linkName, cl_base*
ob_parent, bool notFound) {

        cl_2* ob_2L;
        ob_2L = new cl_2((cl_base*)ob_parent, true);
        ob_2L->setLinkName(linkName);
        ob_2L->setStateDo(false);


        if (linkName[count] == '/') {
                count++;
                string name;
                while (linkName[count] != '/') {
                        name.push_back(linkName[count]);
                        if (count == (linkName.size() - 1)) {
                                break;
                        }
```

```
                        else
                                count++;
                }

                if (linkName[count] == '/') {
                        if (name.size() == 0) {
                                if (checkMember(ob_parent)) {
                                        count++;
                                        string name;
                                        while (linkName[count] != '/') {

name.push_back(linkName[count]);
                                                if (count == (linkName.size()
- 1)) {
                                                        break;
                                                }
                                                else
                                                        count++;
                                        }
                                        if (checkMemberDeep(this, name)) {
                                                if (getStateDo((cl_2*)ob_2L)
== false) {
                                                        ob_2L-
>setCheckName(name);
                                                        ob_2L-
>setStateDo(true);
                                                }
                                        }
                                }
                        }
                        else {
                                if (name == get_object_name(ob_parent)) {
                                        if (checkMember(ob_parent)) {
                                                ob_parent->it_child =
ob_parent->children.begin();
                                                while (ob_parent->it_child !=
ob_parent->children.end()) {
                                                        connectAccess(count,
linkName, ob_2L, (*(ob_parent->it_child)));
                                                        ob_parent->it_child++;
                                                }
                                        }
                                }
                        }
                }
                else {

                        if (name == get_object_name(ob_parent)) {
                                if (getStateDo((cl_2*)ob_2L) == false) {
                                        ob_2L->setCheckName(name);
                                        ob_2L->setStateDo(true);
                                }
                        }
                }
        }
}
void cl_application::connectAccess(int count, string linkName, cl_2* ob_2L,
cl_base* ob_parent) {
        count++;
        string name;
```

```
        while (linkName[count] != '/') {
                name.push_back(linkName[count]);
                if (count == (linkName.size() - 1)) {
                        break;
                }
                else
                        count++;
        }

        if (linkName[count] == '/') {
                if (name.size() == 0) {
                        if (checkMember(ob_parent)) {
                                count++;
                                string name;
                                while (linkName[count] != '/') {
                                        name.push_back(linkName[count]);
                                        if (count == (linkName.size() - 1)) {
                                                break;
                                        }
                                        else
                                                count++;
                                }
                                if (checkMemberDeep(this, name)) {
                                        if (getStateDo((cl_2*)ob_2L) == false)
{
                                                ob_2L->setCheckName(name);
                                                ob_2L->setStateDo(true);
                                        }
                                }
                        }
                }
                else {
                        if (name == get_object_name(ob_parent)) {
                                if (checkMember(ob_parent)) {
                                        ob_parent->it_child = ob_parent-
>children.begin();
                                        while (ob_parent->it_child !=
ob_parent->children.end()) {
                                                connectAccess(count, linkName,
ob_2L, (*(ob_parent->it_child)));
                                                ob_parent->it_child++;
                                        }
                                }
                        }
                }
        }
        else {

                if (name == get_object_name(ob_parent)) {
                        if (getStateDo((cl_2*)ob_2L) == false) {
                                ob_2L->setCheckName(name);
                                ob_2L->setStateDo(true);
                        }
                }
        }
}
```

```
bool cl_application::checkMemberDeep(cl_base* ob_parent, string name) {
	bool result = false;
	ob_parent->it_child = ob_parent->children.begin();
	while (ob_parent->it_child != ob_parent->children.end()) {
		if (name == get_object_name(*(ob_parent->it_child))) {
			result = true;
		}
		if (checkMemberDeep((*(ob_parent->it_child)), name)) {
			result = true;
		}
		ob_parent->it_child++;
	}
	return result;
}
bool cl_application::checkMember(cl_base* ob_parent) {
	if (ob_parent->children.size() == 0) {
		return false;
	} else
		return true;
}


int cl_application::exec_app() {
	show_object_state();

	return 0;
}

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


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();
void scanElementsX(char nextChar, cl_base* ob_parent);
void doWithChildLink(cl_base* ob_parent);
void addNewChild(cl_base* ob_parent);
void resultLink(cl_base* ob_parent);
void processAccess(cl_base* ob_parent);
void connectWithRoot(int count, string linkName, cl_base* ob_parent, bool
notFound);
void connectAccess(int count, string linkName, cl_2* ob_2L, cl_base*
ob_parent);
bool checkMemberDeep(cl_base* ob_parent, string name);
bool checkMember(cl_base* ob_parent);
int exec_app();
void show_object_state();
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;
        }
```

```
}

cl_base::cl_base(cl_base* p_parent, bool checkLink) {
        set_object_name("cl_base");
        if (p_parent) {
                this->p_parent = p_parent;
                p_parent->addChildLink(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;
}

//Start new 3.3

void cl_base::setLinkName(string linkName) {
        this->linkName = linkName;
}
string cl_base::getLinkName(cl_base* p_parent) {
        return p_parent->linkName;
}
void cl_base::setCheckName(string checkName) {
        this->checkName = checkName;
}
string cl_base::getCheckName(cl_base* p_parent) {
        return p_parent->checkName;
}
void cl_base::setStateDo(bool stateDo) {
        this->stateDo = stateDo;
}
bool cl_base::getStateDo(cl_base* p_parent) {
        return p_parent->stateDo;
}

void cl_base::addChildLink(cl_base* p_child) {
        childrenLink.push_back(p_child);
}

//End new 3.3
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);

//New 3.3
cl_base(cl_base* p_parent, bool checkLink);
//

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);

//Start New 3.3
void setLinkName(string linkName);
string getLinkName(cl_base* p_parent);
void setCheckName(string checkName);
string getCheckName(cl_base* p_parent);
void setStateDo(bool stateDo);
bool getStateDo(cl_base* p_parent);
void addChildLink(cl_base* p_child);

vector < cl_base* > childrenLink;
vector < cl_base* > ::iterator it_childLink;
//End New 3.3

vector < cl_base* > children;
vector < cl_base* > ::iterator it_child;
string text_finish = "endtree";
string text_finish2 = "//";


private:

//Start New 3.3
string linkName;
string checkName;
bool stateDo;
//End New 3.3

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
root /root obj1 2 1 /root obj2 2 -1 /root/obj1 obj3 2 1 /root/obj1 obj4 2 1 /root/obj2 obj3 2 1 /root/obj2 obj6 2 1 endtree /root/obj2/obj3 /root/obj1/obj3 //obj3 //	Object tree root obj1 obj3 obj4 obj2 obj3 obj6 /root/obj2/obj3 Object name: obj3 /root/obj1/obj3 Object name: obj3 //obj3 Object name: obj3	Object tree root obj1 obj3 obj4 obj2 obj3 obj6 /root/obj2/obj3 Object name: obj3 /root/obj1/obj3 Object name: obj3 //obj3 Object name: obj3
root /root object_1 3 1 /root object_2 2 1 /root/object_2 object_4 3 -1 /root/object_2 object_5 4 1 /root object_3 3 1 /root/object_2 object_3 6 1 /root/object_1 object_7 5 1 /root/object_2/object_4 object_7 3 -1 endtree /root/object_3 /root/object_2/object_3 //	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 /root/object_3 Object name: object_3 /root/object_2/object_3 Object name: object_3	Object tree root object_1 object_7 object_2 object_4 object_7 object_5 object_3 object_3 /root/object_3 Object name: object_3 /root/object_2/object_3 Object name: object_3