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_4 Сигналы и обработчики »

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

Реализация сигналов и обработчиков

Для организации взаимодействия объектов вне схемы взаимосвязи используется механизм сигналов и обработчиков. Вместе с передачей сигнала еще передаются определенное множество данных. Механизм сигналов и обработчиков реализует схему взаимодействия объектов один ко многим.

Реализовать механизм взаимодействия объектов с использованием сигналов и обработчиков, с передачей вместе сигналом текстового сообщения (строковой переменной).

Для организации взаимосвязи по механизму сигналов и обработчиков в базовый класс добавить три метода:

1. Установления связи между сигналом текущего объекта и обработчиком целевого объекта;

2. Удаления (разрыва) связи между сигналом текущего объекта и обработчиком целевого объекта;

3. Выдачи сигнала от текущего объекта с передачей строковой переменной.

Методу выдачи сигнала передать указатель на метод сигнала и строковую переменную. Реализовать алгоритм:

1. Вызов метода сигнала с передачей строковой переменной по ссылке.

2. Цикл по всем связям сигнал-обработчик текущего объекта.

   2.1. Если в очередной связи сигнал-обработчик участвует метод сигнала, переданный по параметру, то вызвать метод обработчика очередного целевого объекта и передав в качестве аргумента строковую переменную по значению.

3. Конец цикла.

Для приведения указателя на метод сигнала и на метод обработчика использовать макроопределение с параметром препроцессора.

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

Система содержит объекты трех классов с номерами: 1,2,3. Классу корневого объекта соответствует номер 1. В каждом классе реализован один метод сигнала и один метод обработчика.

Реализовать алгоритм работы системы:

1. В методе построения дерева иерархи объектов:

    1.1. Построение иерархии объектов согласно вводу.

    1.2. Ввод и построение множества связей сигнал-обработчик для заданных пар объектов.

2. В методе отработки программы:

    2.1. Цикл до признака завершения ввода.

        2.1.1. Ввод наименования объекта и текста сообщения.

        2.1.2. Вызов сигнала заданного объекта и передача в качестве аргумента строковой переменной содержащей текст сообщения.

    2.2. Конец цикла.

Допускаем, что все входные данные вводятся корректно, контроль корректности входных данных можно реализовать для самоконтроля работы программы.

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: «уникальный номер связи»«наименование объекта выдающей сигнал»«наименование целевого объекта»
Уникальный номер связи – натуральное число.
Ввод информации для построения связей завершается строкой, которая содержит 0.

После завершения ввода связей построчно вводиться: «наименование объекта выдающей сигнал»«текст сообщения из одного слова без пробелов»
Последняя строка ввода содержит слово: endsignals

Описание выходных данных

Первая строка:
Object tree

Со второй строки вывести иерархию построенного дерева.
Следующая после вывода дерева объектов строка содержит:
Set connects

Далее, построчно:
«уникальный номер связи»«наименование объекта выдающей сигнал»«наименование целевого объекта»
Последовательность вывода совпадает с последовательностью ввода связей.
Разделитель один пробель.

Следующая после вывода информации о связях объектов строка содержит:
Emit signals

Далее, построчно:
Signal to «наименование целевого объекта» Text: «наименование объекта выдающей сигнал» -> «текст сообщения из одного слова без пробелов»
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.

X

Описание алгоритма

cl_base::cl_base(cl_base* p_parent)

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

cl_base::cl_base(cl_base* p_parent, bool infoSender, bool strorageMess)

№ шага	Предикат	Действие	№ перехода
1	if (infoSender)	set_object_name("cl_base");	2
	else		3
2	if (p_parent)	his->p_parent = p_parent; p_parent->addChildInfoSender(this);	Ø
	else	this->p_parent = 0;	Ø
3	if (strorageMess)	set_object_name("cl_base");	4
	else		Ø
4	if (p_parent)	this->p_parent = p_parent; p_parent->addChildstorageMess(this);	Ø
	else	this->p_parent = 0;	Ø

cl_base::cl_base(cl_base* p_parent, bool id)

№ шага	Предикат	Действие	№ перехода

1		set_object_name("cl_base");	2
2	if (p_parent)	this->p_parent = p_parent; p_parent->addChildInfoSender(this);	Ø
	else	this->p_parent = 0;	Ø

void cl_base::set_object_name(string object_name)

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

string cl_base::get_object_name(cl_base* p_parent)

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

void cl_base::set_parent(cl_base* p_parent)

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

void cl_base::add_child(cl_base* p_child)

№ шага	Предикат	Действие	№ перехода
1		children.push_back(p_child)	Ø

cl_base* cl_base::get_child(string object_name)

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

void cl_base::set_state(int c_state)

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

int cl_base::get_state(cl_base* p_parent)

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

void cl_base::setConnect(int id, string nameSender, string nameReceiver)

№ шага	Предикат	Действие	№ перехода

1		setID(id);
		setNameSender(nameSender);
		setNameReceiver(nameReceiver);
			Ø

void cl_base::signaling(string message, string nameSender)

№ шага	Предикат	Действие	№ перехода
1		setNameSender(nameSender); setMessageText(message);	Ø

void cl_base::setNameSender(string nameSender)

№ шага	Предикат	Действие	№ перехода
1		this->nameSender = nameSender;	Ø

string cl_base::getNameSender(cl_base* p_parent)

№ шага	Предикат	Действие	№ перехода
1		return p_parent->nameSender;	Ø

void cl_base::setNameReceiver(string nameReceiver)

№ шага	Предикат	Действие	№ перехода
1		this->nameReceiver = nameReceiver;	Ø

string cl_base::getNameReceiver(cl_base* p_parent)

№ шага	Предикат	Действие	№ перехода
1		return p_parent->nameReceiver;	Ø

void cl_base::setMessageText(string messageText)

№ шага	Предикат	Действие	№ перехода
1		this->messageText = messageText;	Ø

string cl_base::getMessageText(cl_base* p_parent)

№ шага	Предикат	Действие	№ перехода
1		return p_parent->messageText;	Ø

void cl_base::setID(int id)

№ шага	Предикат	Действие	№ перехода
1		this->id = id;	Ø

int cl_base::getID(cl_base* p_parent)

№ шага	Предикат	Действие	№ перехода
1		return p_parent->id;	Ø

void cl_base::addChildInfoSender(cl_base* p_child)

№ шага	Предикат	Действие	№ перехода
1		infoSender.push_back(p_child);	Ø

void cl_base::addChildstorageMess(cl_base* p_child)

№ шага	Предикат	Действие	№ перехода
1		storageMess.push_back(p_child);	Ø

cl_application::cl_application(string name)

№ шага	Предикат	Действие	№ перехода
1		set_object_name(name); set_state(1);	Ø

void cl_application::bild_tree_objects()

№ шага	Предикат	Действие	№ перехода
1		cl_2* ob_2; cl_3* ob_3; cl_4* ob_4;	2

		cl_5* ob_5; cl_6* ob_6; string nameParent, nameChild; int selectFamily; int state;	
2	true	cin >> nameParent;	3
	!(true)		15
3	nameParent == text_finish	break;	15
	!(nameParent == text_finish)		4
4		cin >> nameChild >> selectFamily >> state;	5
5	selectFamily == 2		6
	!(selectFamily == 2)		7

6	get_object_name(this) == nameParent	ob_2 = new cl_2((cl_base*)this);
		ob_2->set_object_name(nameChild);
		ob_2->set_state(state);
			2

	!(get_object_name(this) == nameParent)	addNewChild(this, nameParent, nameChild, state, selectFamily);	2
7	selectFamily == 3		8
	!(selectFamily == 3)		9
8	get_object_name(this) == nameParent	ob_3 = new cl_3((cl_base*)this); ob_3->set_object_name(nameChild);	2

		ob_3->set_state(state);	
	!(get_object_name(this) == nameParent)	addNewChild(this, nameParent, nameChild, state, selectFamily);	2
9	selectFamily == 4		10
	!(selectFamily == 4)		11

10	get_object_name(this) == nameParent	ob_4 = new cl_4((cl_base*)this);
		ob_4->set_object_name(nameChild);
		ob_4->set_state(state);
			2

	!(get_object_name(this) == nameParent)	addNewChild(this, nameParent, nameChild, state, selectFamily);	2
11	selectFamily == 5		12
	!(selectFamily == 5)		13

12	get_object_name(this) == nameParent	ob_5 = new cl_5((cl_base*)this);
		ob_5->set_object_name(nameChild);
		ob_5->set_state(state);
			2

	!(get_object_name(this) == nameParent)	addNewChild(this, nameParent, nameChild, state, selectFamily);	2
13	selectFamily == 6		14
	!(selectFamily == 6)		15

14	get_object_name(this) == nameParent	ob_6 = new cl_6((cl_base*)this);
		ob_6->set_object_name(nameChild);
		ob_6->set_state(state);
		break;
			2

	!(get_object_name(this) == nameParent)	addNewChild(this, nameParent, nameChild, state, selectFamily);	2
15		scanConnects(this);	Ø

		scanSignals(this);	

void cl_application::scanConnects(cl_base* ob_parent)

№ шага	Предикат	Действие	№ перехода
1	true	int id; string sender, receiver; cin >> id;	2
	!(true)		Ø
2	id == 0	break;	Ø

	!(id == 0)	cin >> sender >> receiver;
		cl_2* connectStart;
		connectStart = new cl_2((cl_base*)ob_parent, true, false);
		connectStart->setConnect(id, sender, receiver);
			1

void cl_application::scanSignals(cl_base* ob_parent)

№ шага	Предикат	Действие	№ перехода
1	true	string sender; string message; cin >> sender;	2
	!(true)		Ø
2	sender == "endsignals"	break;	Ø
	!(sender == "endsignals")	cin >> message;	1

		cl_2* messageStart;
		
		messageStart = new cl_2((cl_base*)ob_parent, false, true);
		
		messageStart->signaling(message, sender);	

void cl_application::printConnects(cl_base* ob_parent)

№ шага	Предикат	Действие	№ перехода
1		cout << endl << "Set connects";	2
2	ob_parent->infoSender.size() == 0	return;	Ø
	!(ob_parent->infoSender.size() == 0)		3
3		ob_parent->it_iS = ob_parent->infoSender.begin();	4

4	ob_parent->it_iS != ob_parent->infoSender.end()	cout << endl << getID((*(ob_parent->it_iS))) << " " << getNameSender((*(ob_parent->it_iS))) << " " << getNameReceiver(*(ob_parent->it_iS));
		
		ob_parent->it_iS++;
			4

	!(ob_parent->it_iS != ob_parent->infoSender.end())		Ø

void cl_application::printInfoWithMessage(cl_base* ob_parent)

№ шага	Предикат	Действие	№ перехода
1		cout << endl << "Emit signals";	2
2infoSender	ob_parent->storageMess.size() == 0	return;	Ø

it_iS	!(ob_parent->storageMess.size() == 0)		3
3		ob_parent->it_sM = ob_parent->storageMess.begin();	4

4	ob_parent->it_sM != ob_parent->storageMess.end()	returnMessage(this, getNameSender(*(ob_parent->it_sM)), getMessageText((*(ob_parent->it_sM))));
		
		ob_parent->it_sM++;
			4

	!(ob_parent->it_sM != ob_parent->storageMess.end())		Ø

void cl_application::returnMessage(cl_base* ob_parent, string sender, string receiver)

№ шага	Предикат	Действие	№ перехода
1	ob_parent->infoSender.size() == 0	return;	Ø
	!(ob_parent->infoSender.size() == 0)		2
2		ob_parent->it_iS = ob_parent->infoSender.begin();	3
3	ob_parent->it_iS != ob_parent->infoSender.end()		4
	!(ob_parent->it_iS != ob_parent->infoSender.end())		Ø
4	sender == getNameSender((*(ob_parent->it_iS)))	cout << endl << "Signal to " << getNameReceiver((*(ob_parent->it_iS))) << " Text: " << sender << " -> " <<	5

		message;	
	!(sender == getNameSender((*(ob_parent->it_iS))))		5
5		ob_parent->it_it_iS++;	3

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

№ шага	Предикат	Действие	№ перехода
1		cl_2* ob_2; cl_3* ob_3; cl_4* ob_4; cl_5* ob_5; cl_6* ob_6;	2
2	selectFamily == 2		3
	!(selectFamily == 2)		7
3	for (size_t i = 0; i < ob_parent->children.size(); i++)		4
	i = ob_parent->children.size()		28

4	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);
			5

	!(get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)		3

5	get_state((cl_base*)ob_parent->children.at(i)) > 0	ob_2->set_state(state);	6
	!(get_state((cl_base*)ob_parent->children.at(i)) > 0)	ob_2->set_state(0);	6
6		return;	Ø
7	selectFamily == 3		8
	!(selectFamily == 3)		12
8	for (size_t i = 0; i < ob_parent->children.size(); i++)		9
	i = ob_parent->children.size())		28

9	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);
			10

	!(get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)		8
10	get_state((cl_base*)ob_parent->children.at(i)) > 0	ob_3->set_state(state);	11
	!(get_state((cl_base*)ob_parent->children.at(i)) > 0)	ob_3->set_state(0);	11
11		return;	Ø
12	selectFamily == 4		13
	!(selectFamily == 4)		17
13	for (size_t i = 0; i < ob_parent-		14

	>children.size(); i++)		
	i = ob_parent->children.size())		28

14	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);
			15

	!(get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)		13
15	get_state((cl_base*)ob_parent->children.at(i)) > 0	ob_4->set_state(state);	16
	!(get_state((cl_base*)ob_parent->children.at(i)) > 0)	ob_4->set_state(0);	16
16		return;	Ø
17	selectFamily == 5		18
	!(selectFamily == 5)		22
18	for (size_t i = 0; i < ob_parent->children.size(); i++)		19
	i = ob_parent->children.size())		28

19	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);
			20

	!(get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)		18
20	get_state((cl_base*)ob_parent-	ob_5->set_state(state);	21

	>children.at(i)) > 0		
	!(get_state((cl_base*)ob_parent->children.at(i)) > 0)	ob_5->set_state(0);	21
21		return;	Ø
22	selectFamily == 6		23
	!(selectFamily == 6)		28
23	for (size_t i = 0; i < ob_parent->children.size(); i++)		24
	i = ob_parent->children.size())		28

24	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);
			25

	!(get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent)		23
25	get_state((cl_base*)ob_parent->children.at(i)) > 0	ob_6->set_state(state);	26
	!(get_state((cl_base*)ob_parent->children.at(i)) > 0)	ob_6->set_state(0);	26
26		return;	Ø
28		ob_parent->it_child = ob_parent->children.begin();	29
29	ob_parent->it_child != ob_parent->children.end()	addNewChild((*(ob_parent->it_child)), nameParent, nameChild, state, selectFamily);	29

		ob_parent->it_child++;	
	!(ob_parent->it_child != ob_parent->children.end())		Ø

int cl_application::exec_app()

№ шага	Предикат	Действие	№ перехода
1		show_object_state(); return 0;	Ø

void cl_application::show_object_state()

№ шага	Предикат	Действие	№ перехода

1		show_state_next(this, 0);
		
		printConnects(this);
		
		printInfoWithMessage(this);
			Ø

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

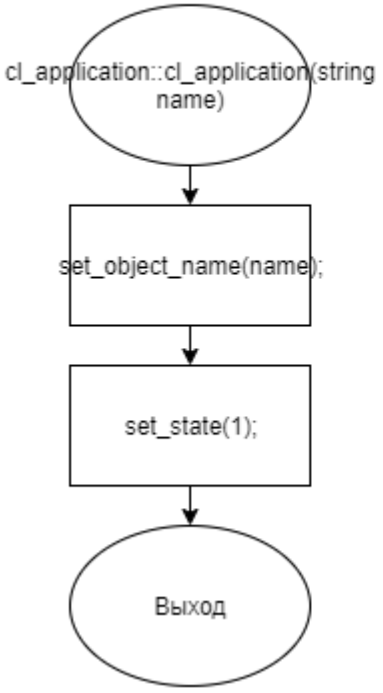
№ шага	Предикат	Действие	№ перехода
1	i == 0	cout << endl << get_object_name(ob_parent);	2
	!(i == 0)	cout << endl << setw(4 * i) << " " << get_object_name(ob_parent);	2
2	ob_parent->children.size() == 0	return;	Ø
	!(ob_parent->children.size() == 0)		3
3		ob_parent->it_child = ob_parent->children.begin();	4
4	ob_parent->it_child != ob_parent->children.end()	show_state_next((*(ob_parent->it_child)), i+1);	4

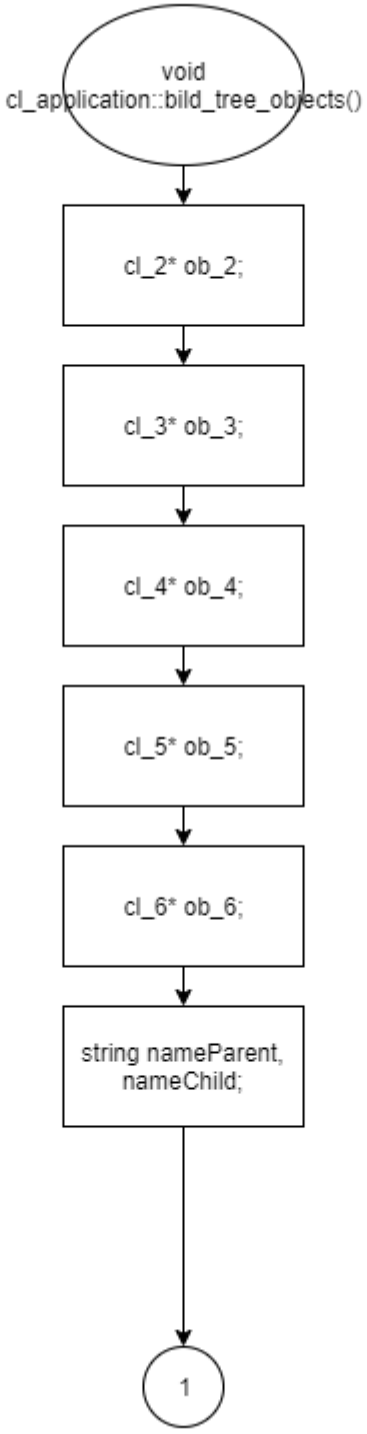
		ob_parent->it_child++;	
	!(ob_parent->it_child != ob_parent->children.end())		Ø

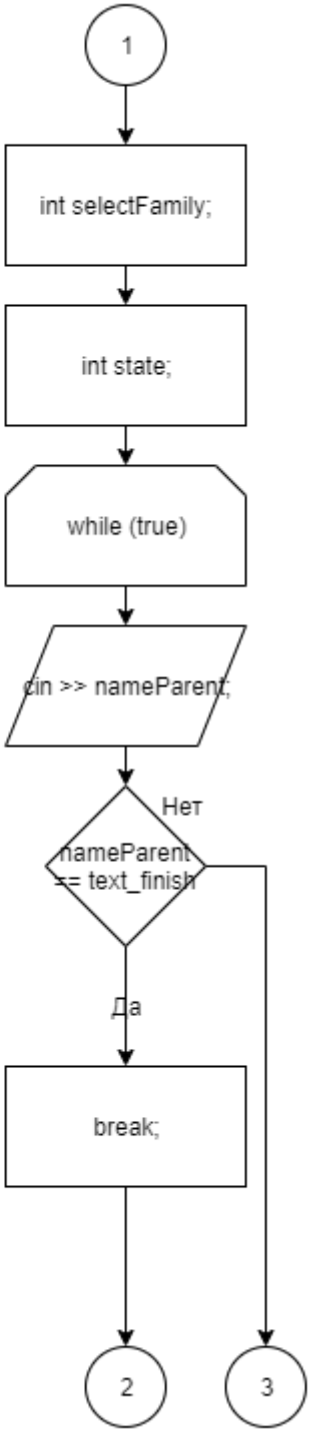
int main()

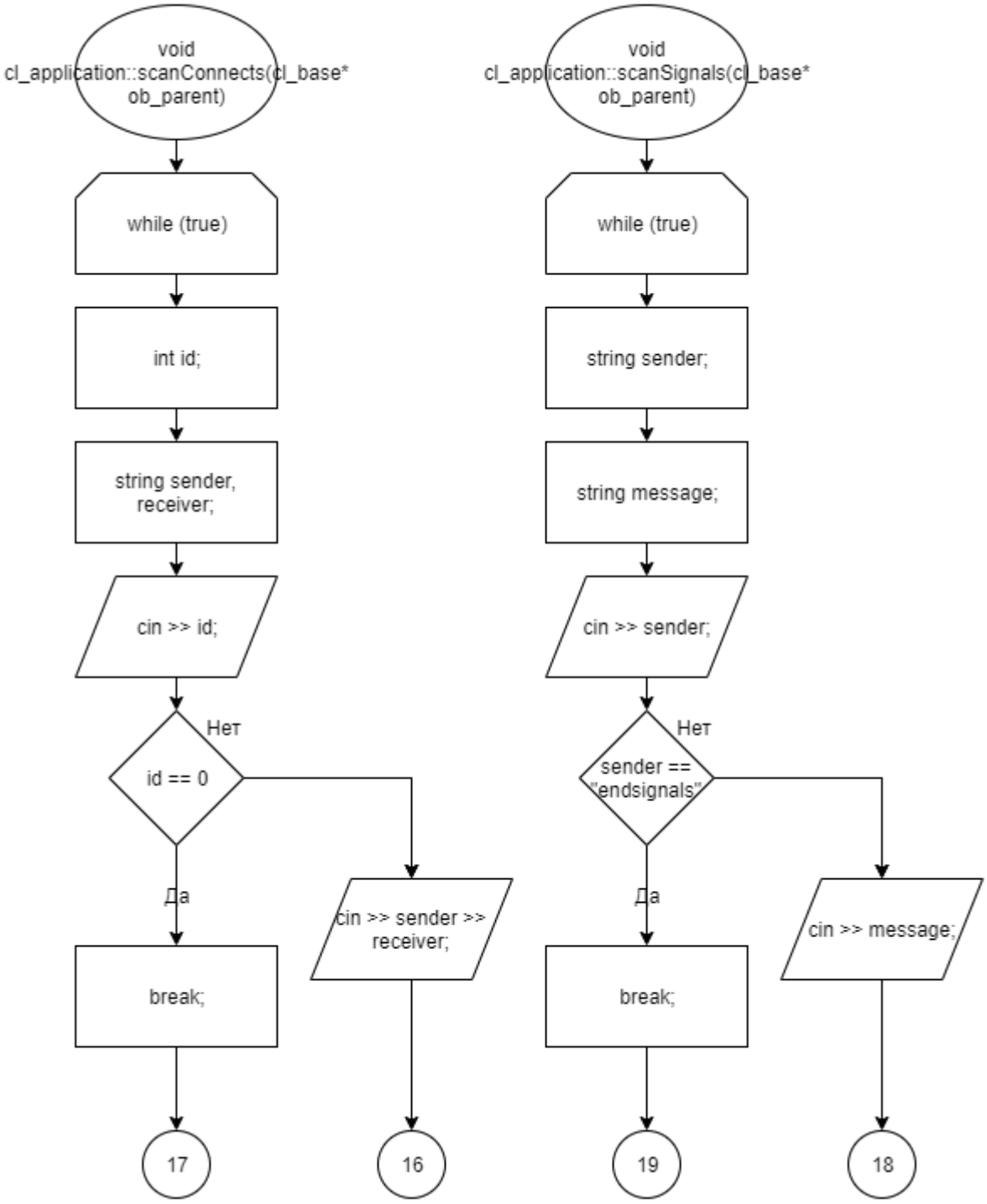
№ шага	Предикат	Действие	№ перехода
1		string name; cin >> name;	2
2		cl_application ob_application(name); ob_application.bild_tree_objects();	3
3		cout << "Object tree";	4
4		return ob_application.exec_app();	Ø

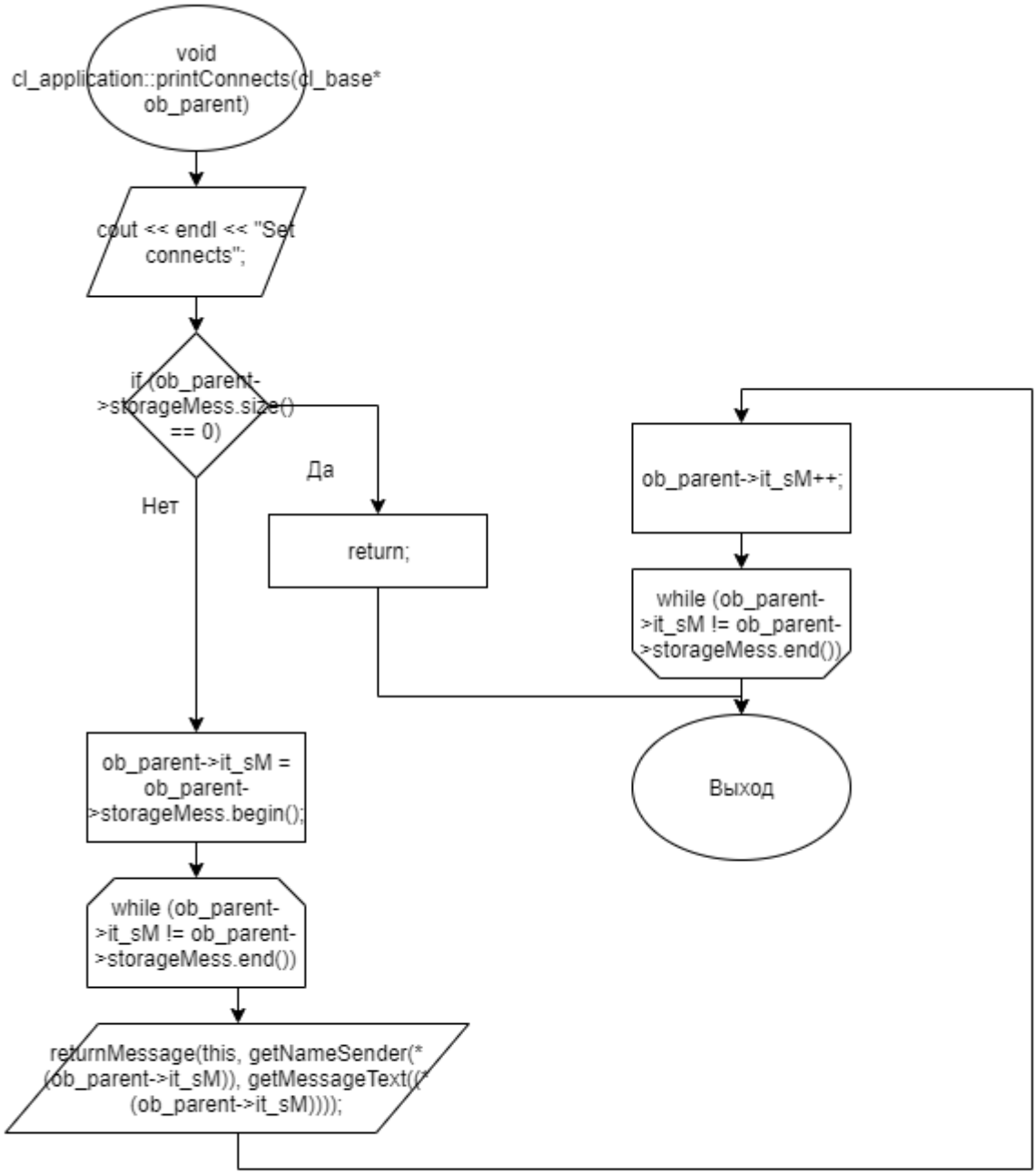
Блок-схема алгоритма

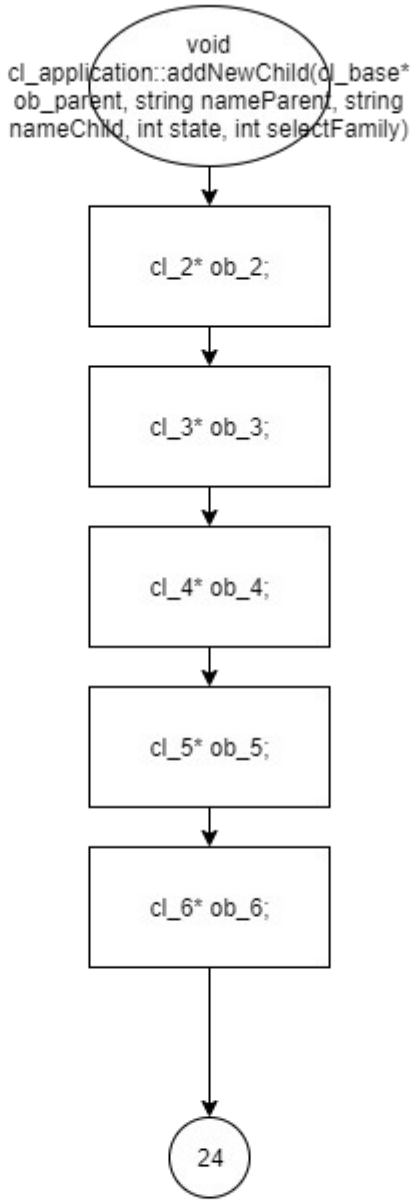

cl_application::cl_application(string name)
set_object_name(name);
set_state(1);
Выход

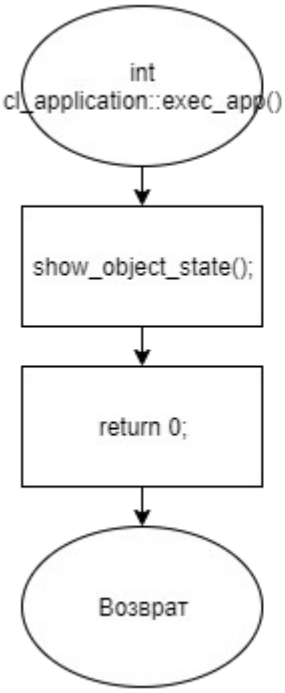

void cl_application::bild_tree_objects()
cl_2* ob_2;
cl_3* ob_3;
cl_4* ob_4;
cl_5* ob_5;
cl_6* ob_6;
string nameParent, nameChild;
1

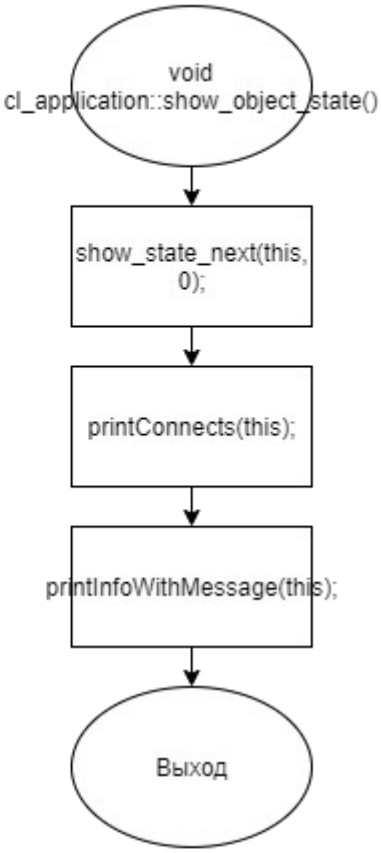

1
int selectFamily;
int state;
while (true)
cin >> nameParent;
Нет
nameParent == text_finish
Да
break;
2
3

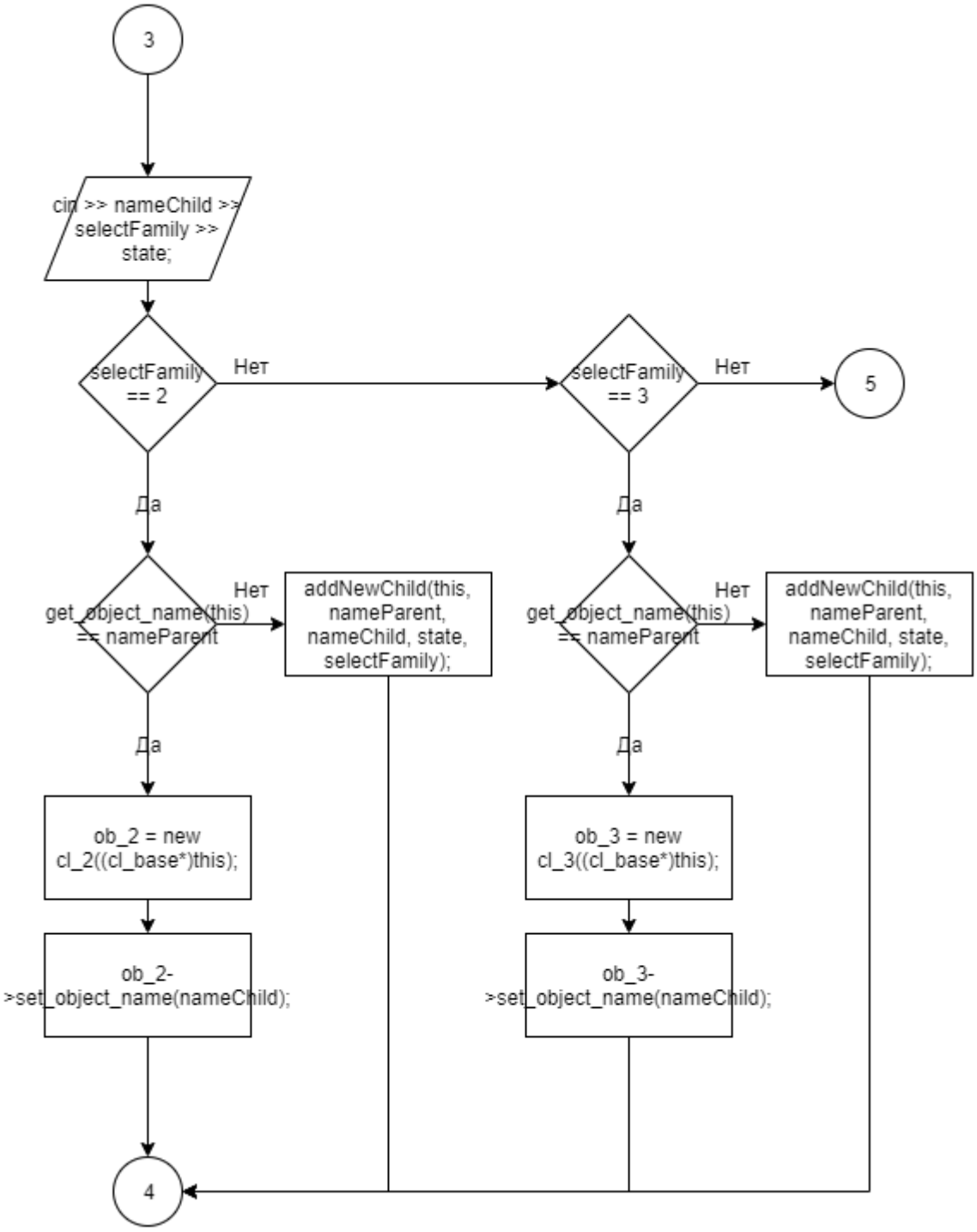

void cl_application::scanConnects(cl_base* ob_parent)
while (true)
int id;
string sender, receiver;
cin >> id;
id == 0
Нет
Да
break;
cin >> sender >> receiver;
17
16
void cl_application::scanSignals(cl_base* ob_parent)
while (true)
string sender;
string message;
cin >> sender;
sender == "endsignals"
Нет
Да
break;
cin >> message;
19
18

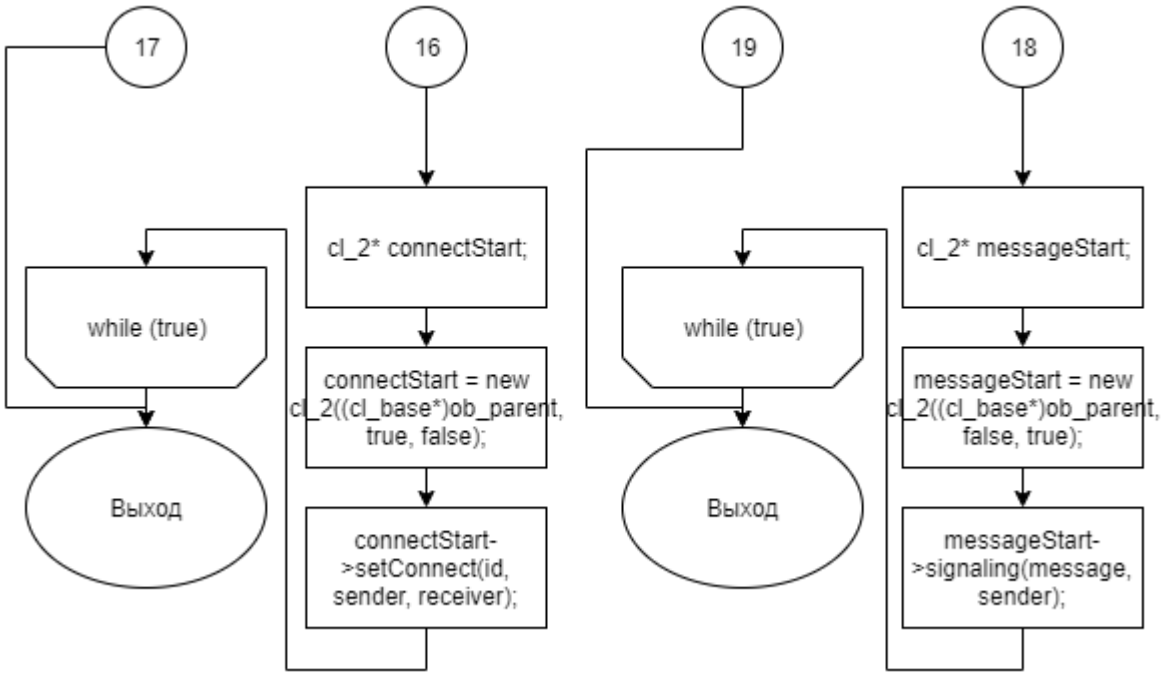

void cl_application::printConnects(cl_base* ob_parent)
cout << endl << "Set connects";
if (ob_parent->storageMess.size() == 0)
Да
return;
Нет
ob_parent->it_sM = ob_parent->storageMess.begin();
while (ob_parent->it_sM != ob_parent->storageMess.end())
returnMessage(this, getNameSender(*(ob_parent->it_sM)), getMessageText((*(ob_parent->it_sM))));
ob_parent->it_sM++;
while (ob_parent->it_sM != ob_parent->storageMess.end())
Выход

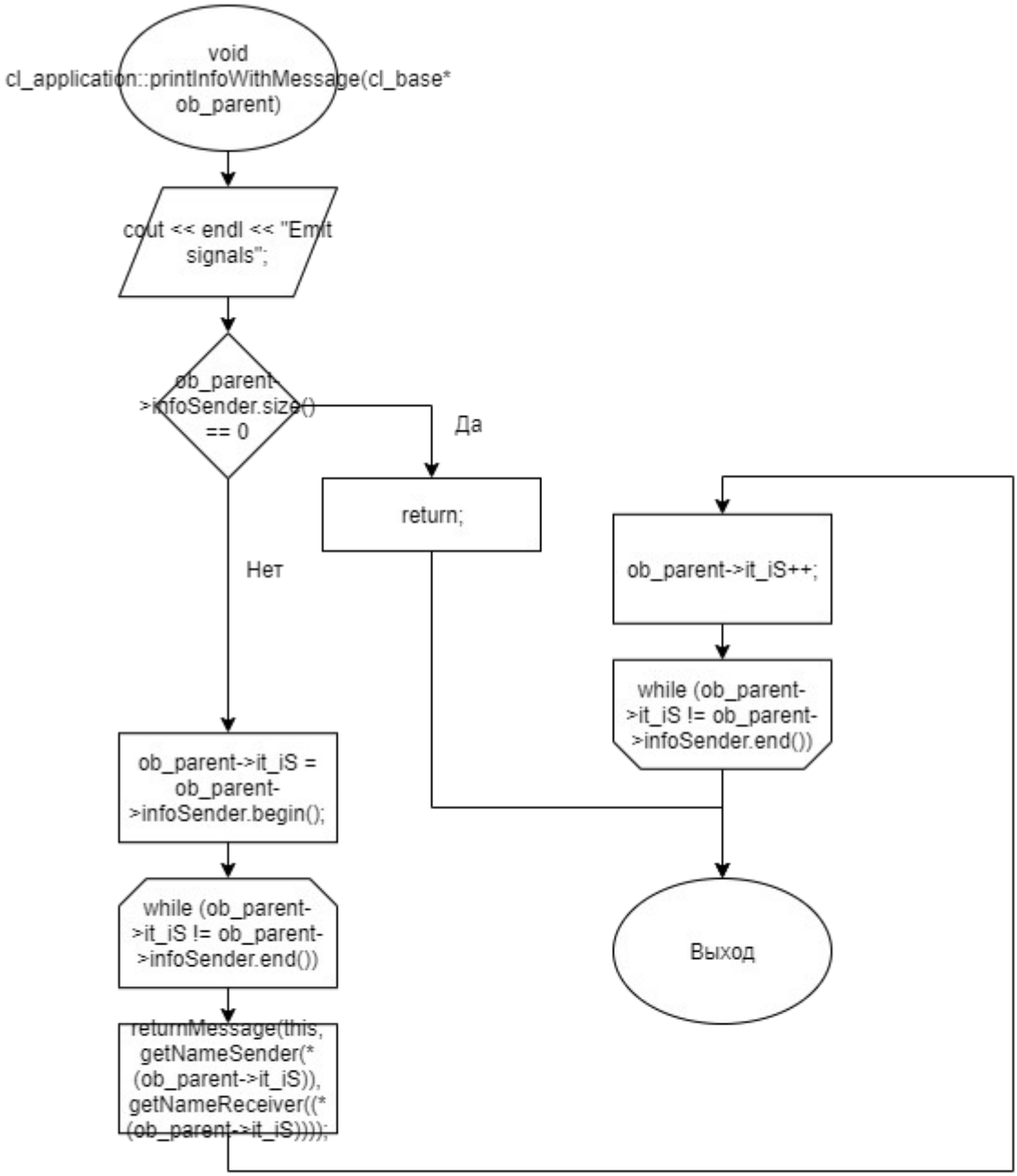

void
cl_application::addNewChild(cl_base*
ob_parent, string nameParent, string
nameChild, int state, int selectFamily)
cl_2* ob_2;
cl_3* ob_3;
cl_4* ob_4;
cl_5* ob_5;
cl_6* ob_6;
24

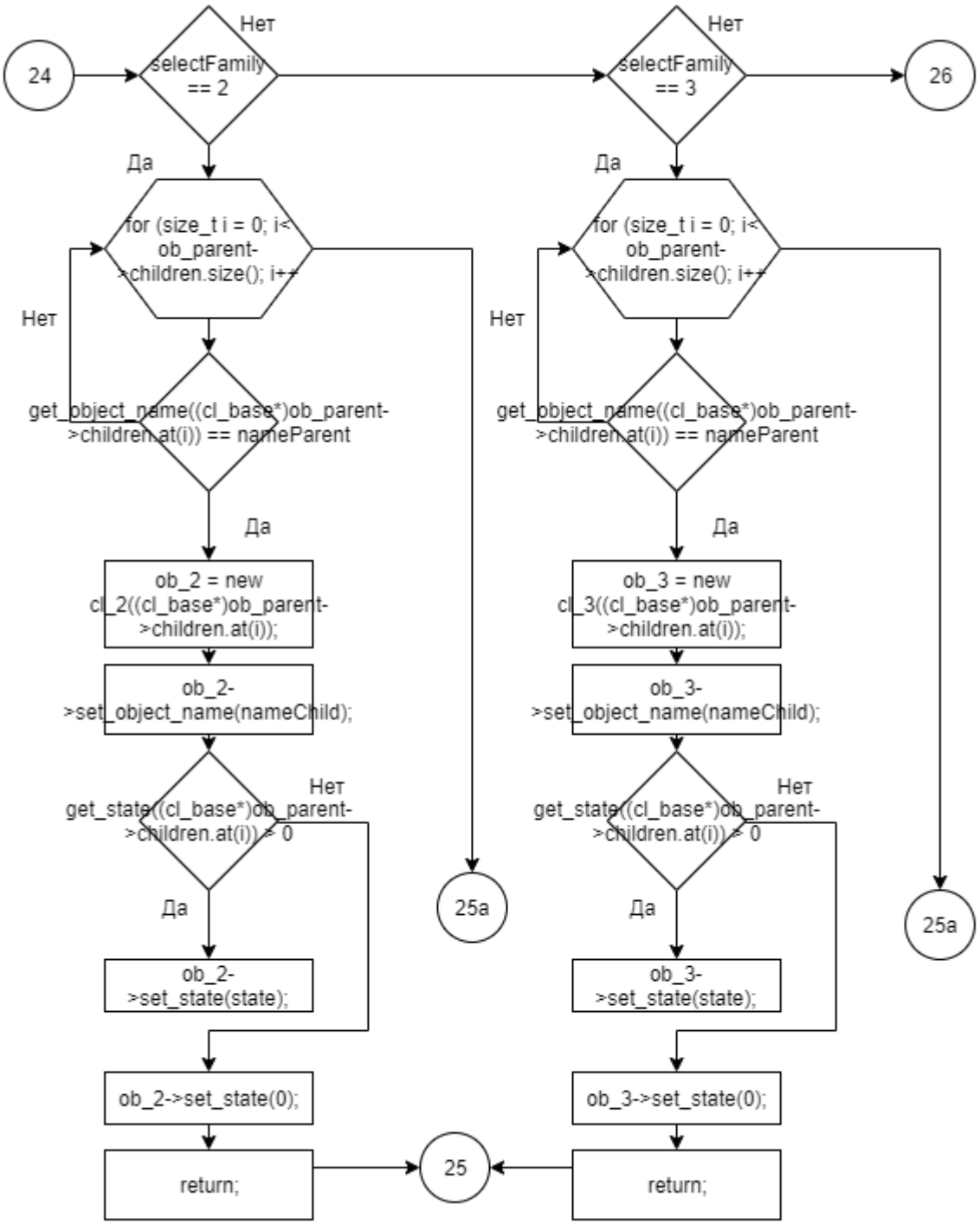

int
cl_application::exec_app()
show_object_state();
return 0;
Возврат

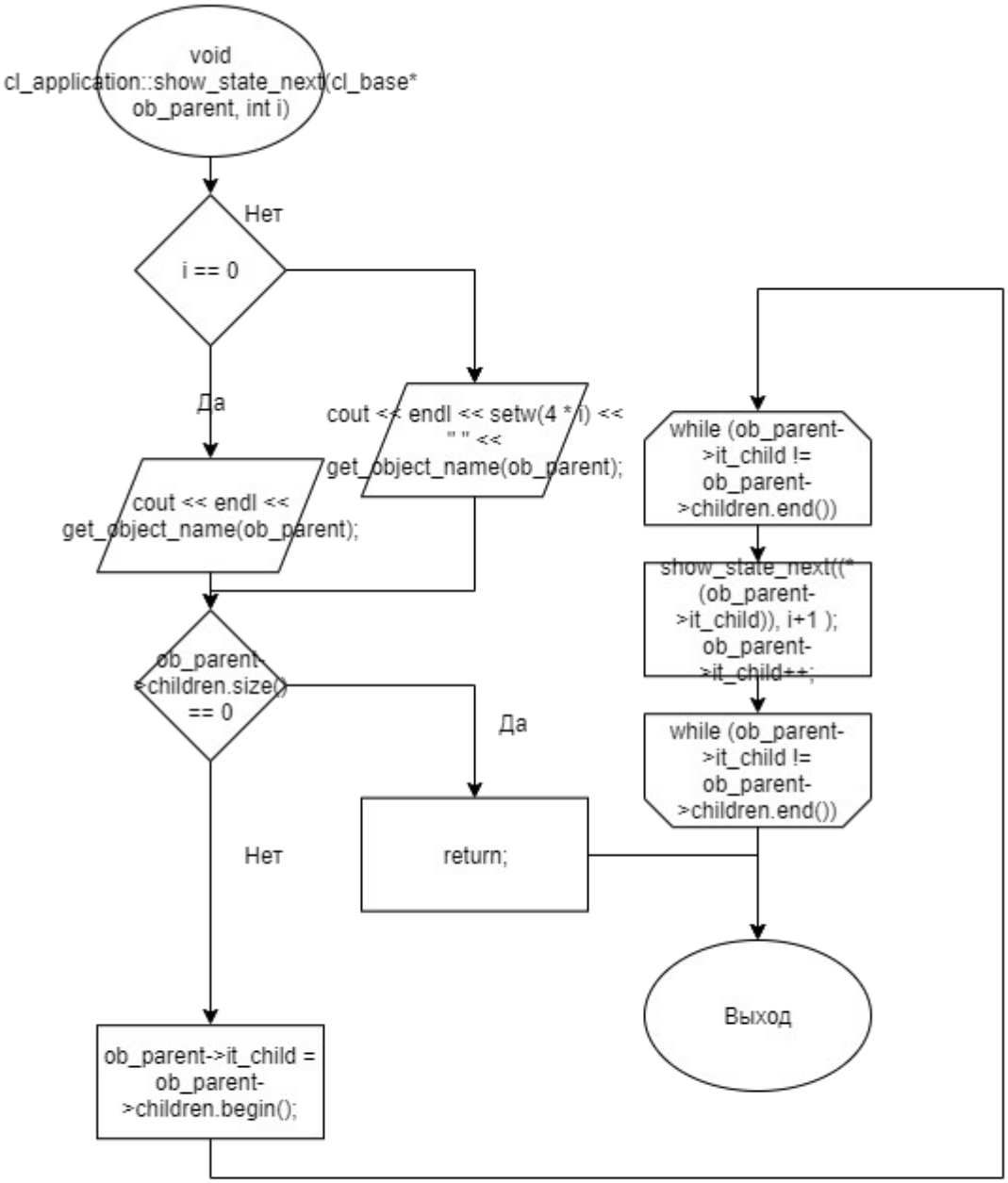

void
cl_application::show_object_state()
show_state_next(this, 0);
printConnects(this);
printInfoWithMessage(this);
Выход

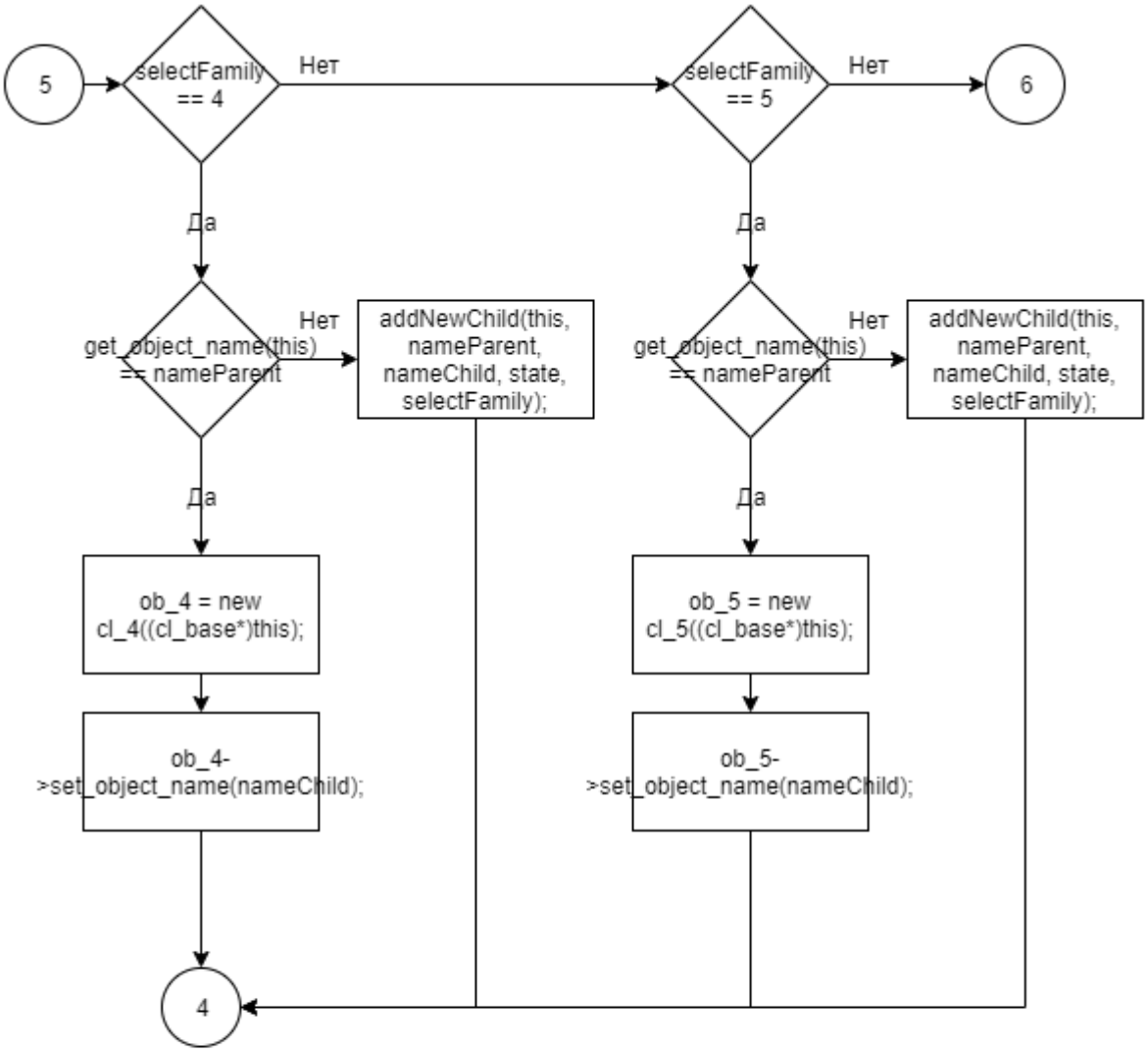

3
cin >> nameChild >> selectFamily >> state;
selectFamily == 2
Нет
selectFamily == 3
Нет
5
Да
Да
get_object_name(this) == nameParent
Нет
addNewChild(this, nameParent, nameChild, state, selectFamily);
get_object_name(this) == nameParent
Нет
addNewChild(this, nameParent, nameChild, state, selectFamily);
Да
Да
ob_2 = new cl_2((cl_base*)this);
ob_3 = new cl_3((cl_base*)this);
ob_2->set_object_name(nameChild);
ob_3->set_object_name(nameChild);
4

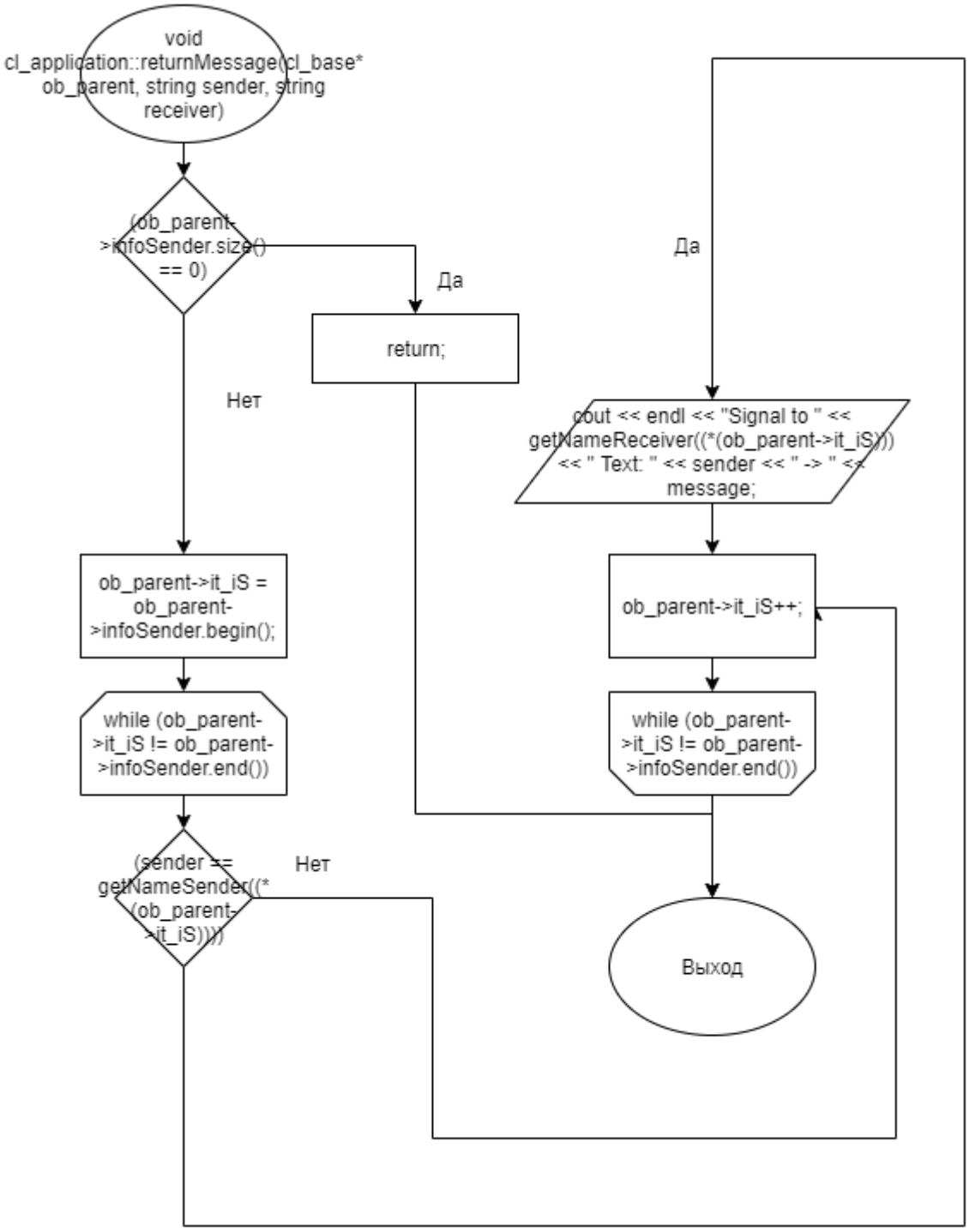

17
16
19
18
while (true)
cl_2* connectStart;
while (true)
cl_2* messageStart;
Выход
connectStart = new cl_2((cl_base*)ob_parent, true, false);
Выход
messageStart = new cl_2((cl_base*)ob_parent, false, true);
connectStart->setConnect(id, sender, receiver);
messageStart->signaling(message, sender);

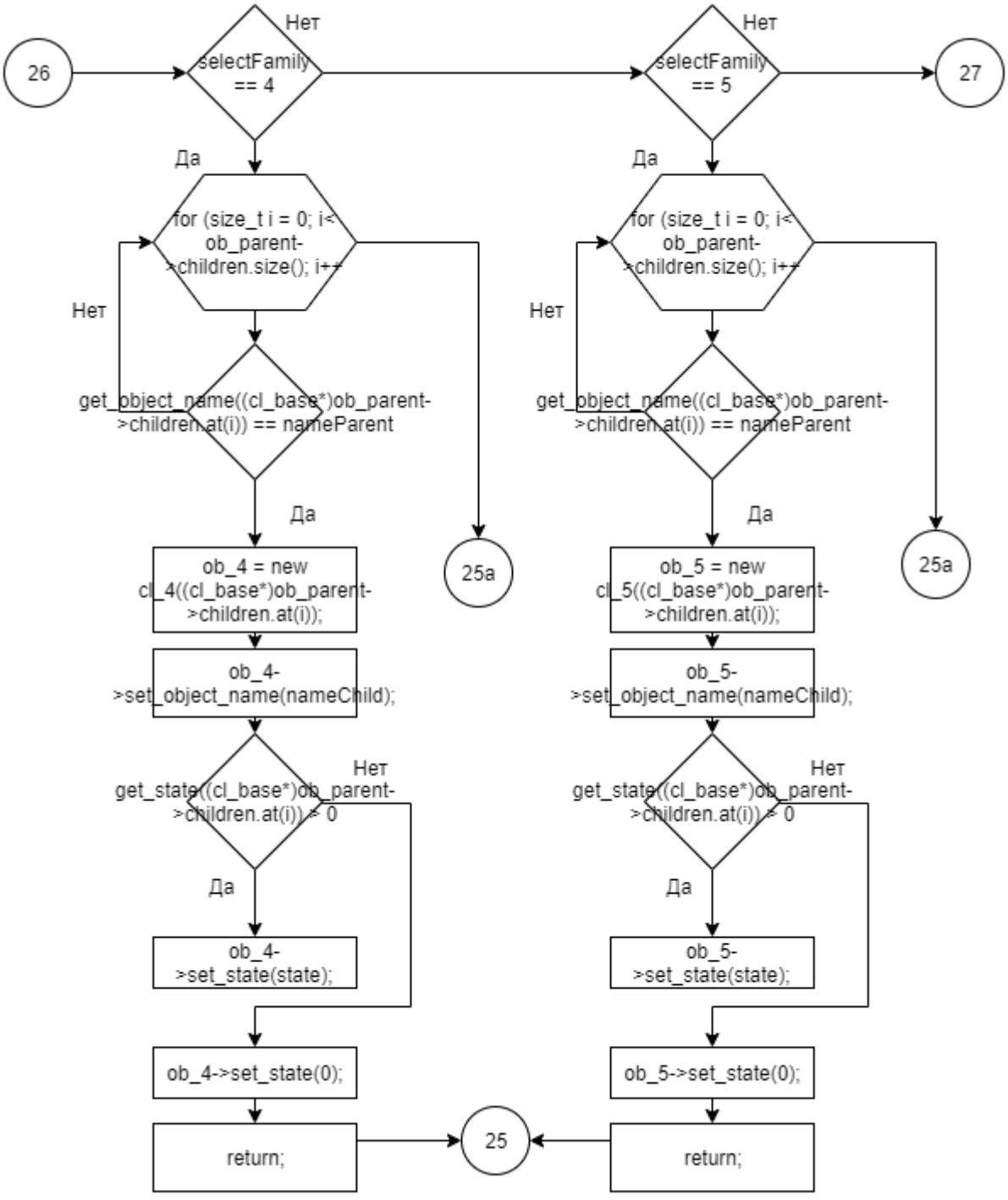

void cl_application::printInfoWithMessage(cl_base* ob_parent)
cout << endl << "Emit signals";
ob_parent->infoSender.size() == 0
Да
return;
Нет
ob_parent->it_iS = ob_parent->infoSender.begin();
while (ob_parent->it_iS != ob_parent->infoSender.end())
returnMessage(this, getNameSender(*(ob_parent->it_iS)), getNameReceiver((*(ob_parent->it_iS))));
ob_parent->it_iS++;
while (ob_parent->it_iS != ob_parent->infoSender.end())
Выход

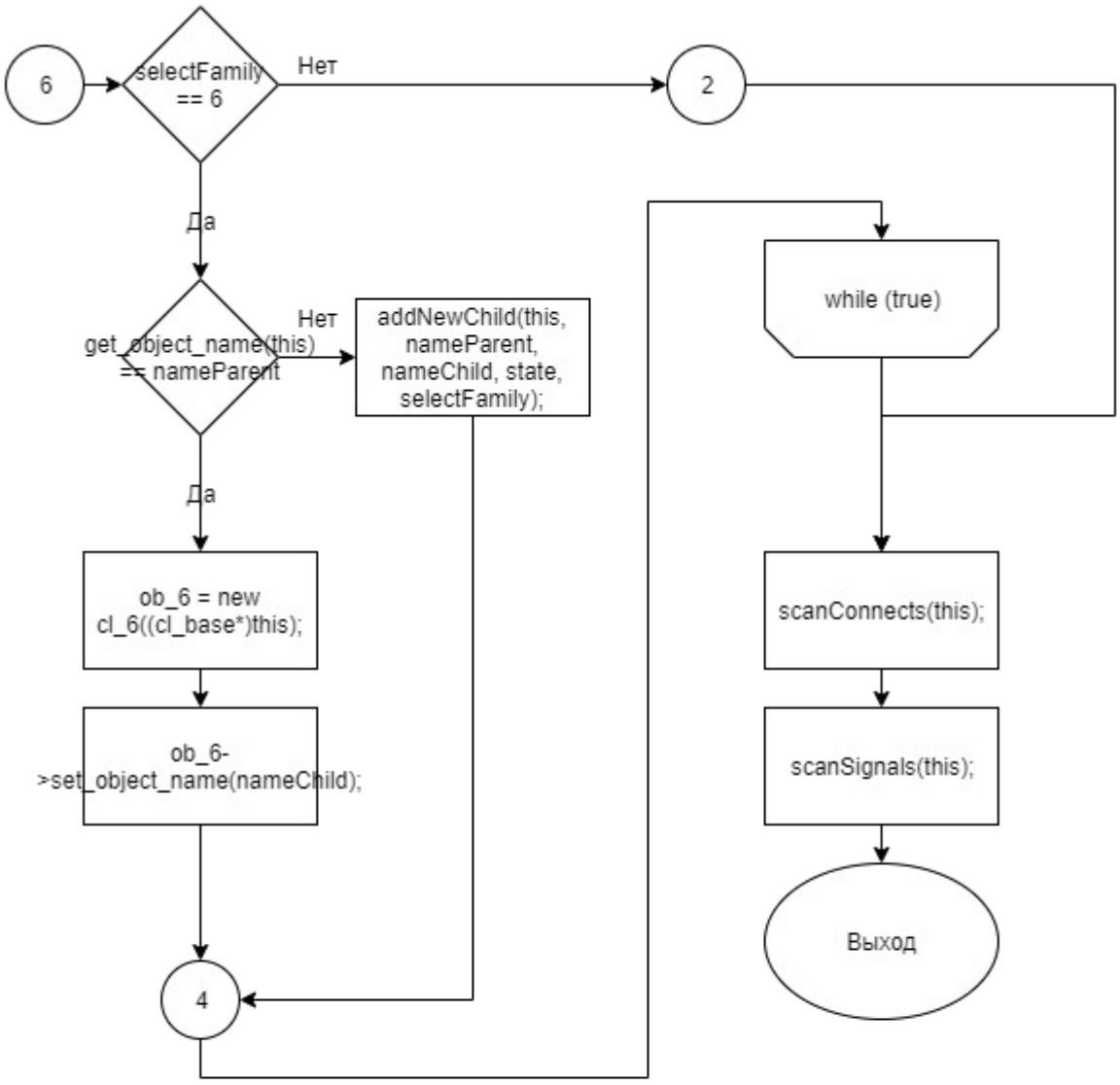

24
selectFamily == 2
Нет
Да
selectFamily == 3
Нет
26
for (size_t i = 0; i< ob_parent->children.size(); i++
Нет
Да
get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent
Нет
Да
ob_2 = new cl_2((cl_base*)ob_parent->children.at(i));
ob_2->set_object_name(nameChild);
get_state((cl_base*)ob_parent->children.at(i)) > 0
Нет
Да
ob_2->set_state(state);
ob_2->set_state(0);
return;
25a
ob_3 = new cl_3((cl_base*)ob_parent->children.at(i));
ob_3->set_object_name(nameChild);
ob_3->set_state(state);
ob_3->set_state(0);
return;
25

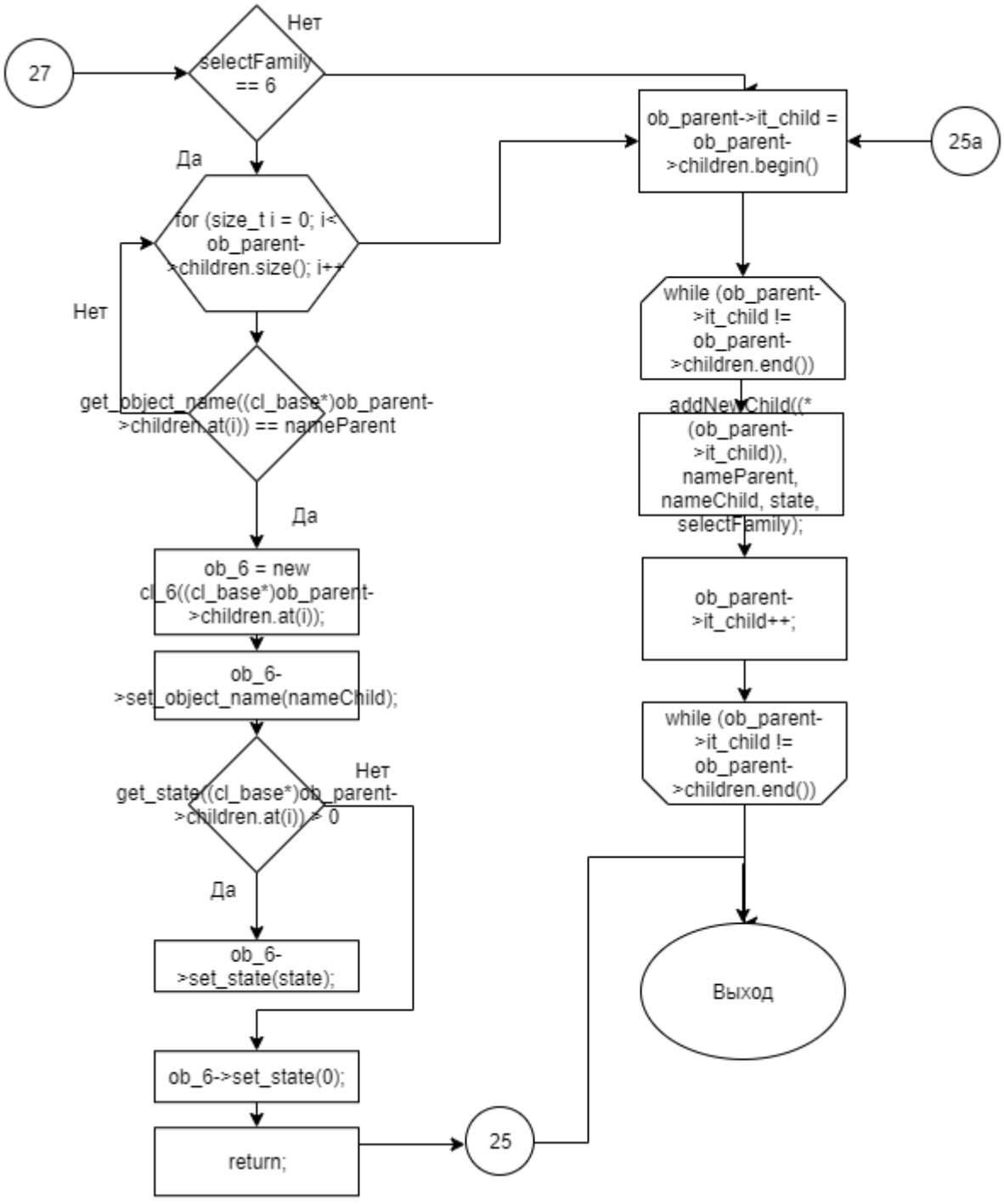

void cl_application::show_state_next(cl_base* ob_parent, int i)
i == 0
Нет
Да
cout << endl << setw(4 * i) << " " << get_object_name(ob_parent);
cout << endl << get_object_name(ob_parent);
ob_parent->children.size() == 0
Да
Нет
return;
ob_parent->it_child = ob_parent->children.begin();
while (ob_parent->it_child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

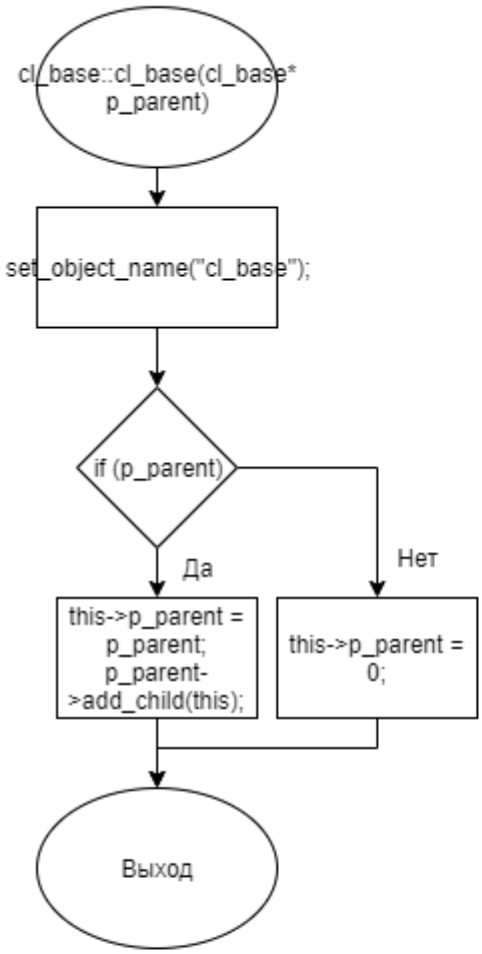

5
selectFamily == 4
Нет
selectFamily == 5
Нет
6
Да
Да
get_object_name(this) == nameParent
Нет
addNewChild(this, nameParent, nameChild, state, selectFamily);
get_object_name(this) == nameParent
Нет
addNewChild(this, nameParent, nameChild, state, selectFamily);
Да
Да
ob_4 = new cl_4((cl_base*)this);
ob_5 = new cl_5((cl_base*)this);
ob_4->set_object_name(nameChild);
ob_5->set_object_name(nameChild);
4

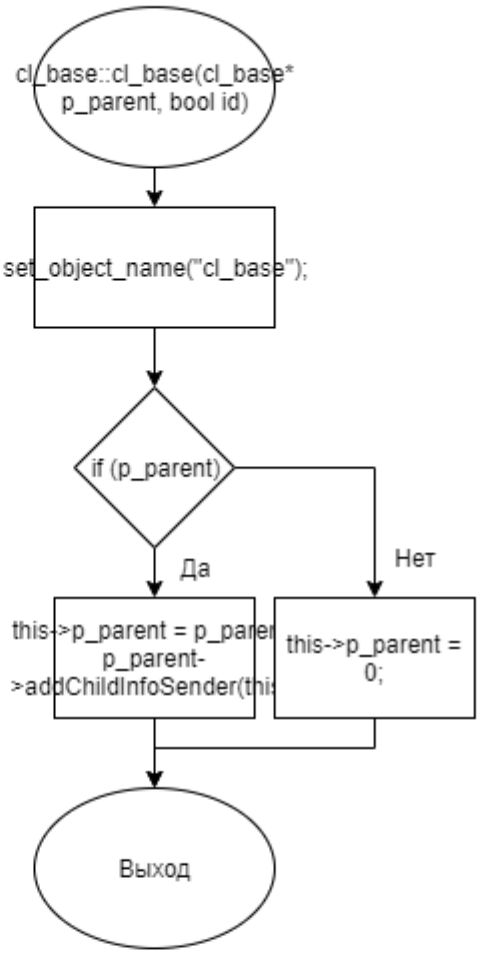

void cl_application::returnMessage(cl_base* ob_parent, string sender, string receiver)
(ob_parent->infoSender.size() == 0)
Да
return;
Нет
ob_parent->it_iS = ob_parent->infoSender.begin();
while (ob_parent->it_iS != ob_parent->infoSender.end())
(sender == getNameSender((*(ob_parent->it_iS))))
Нет
Да
cout << endl << "Signal to " << getNameReceiver((*(ob_parent->it_iS))) << " Text: " << sender << " -> " << message;
ob_parent->it_iS++;
while (ob_parent->it_iS != ob_parent->infoSender.end())
Выход

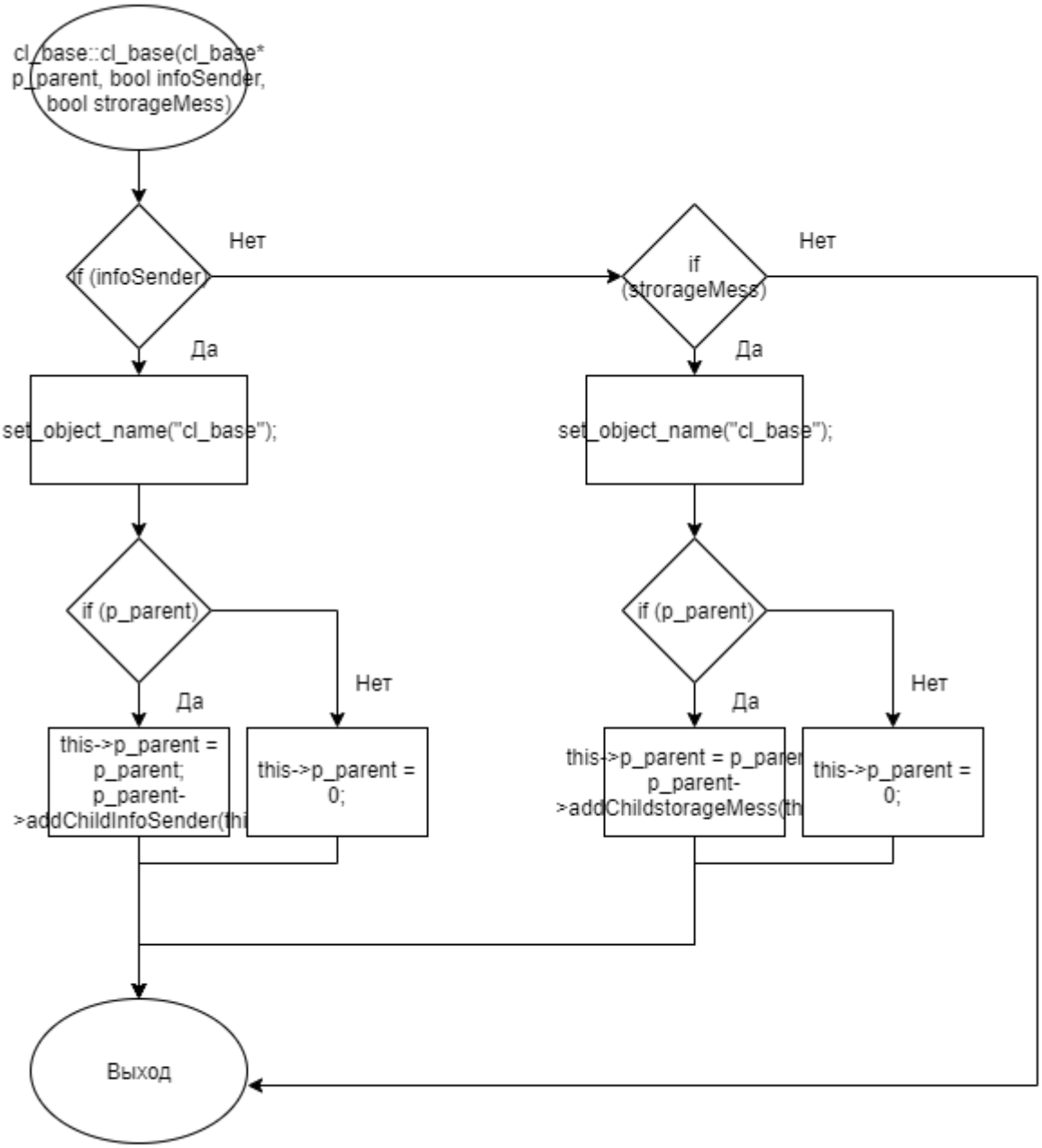

26
selectFamily == 4
Нет
Да
for (size_t i = 0; i< ob_parent->children.size(); i++
Нет
get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent
Да
ob_4 = new cl_4((cl_base*)ob_parent->children.at(i));
ob_4->set_object_name(nameChild);
get_state((cl_base*)ob_parent->children.at(i)) > 0
Нет
Да
ob_4->set_state(state);
ob_4->set_state(0);
return;
25a
selectFamily == 5
27
ob_5 = new cl_5((cl_base*)ob_parent->children.at(i));
ob_5->set_object_name(nameChild);
ob_5->set_state(state);
ob_5->set_state(0);
return;
25

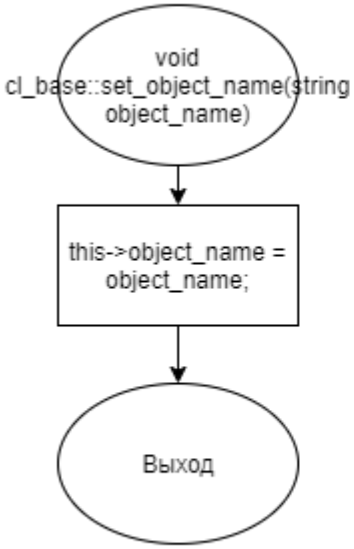

6
selectFamily == 6
Нет
2
Да
get_object_name(this) == nameParent
Нет
addNewChild(this, nameParent, nameChild, state, selectFamily);
Да
ob_6 = new cl_6((cl_base*)this);
ob_6->set_object_name(nameChild);
4
while (true)
scanConnects(this);
scanSignals(this);
Выход

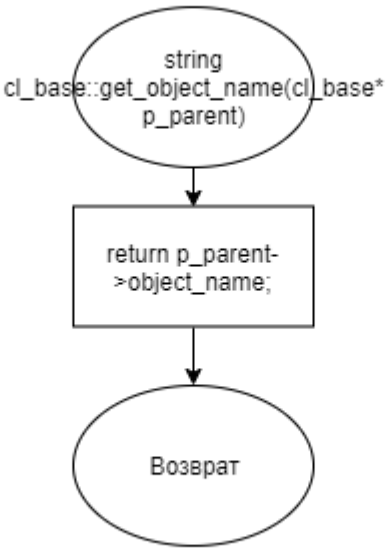

27
selectFamily == 6
Нет
Да
for (size_t i = 0; i< ob_parent->children.size(); i++
Нет
get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent
Да
ob_6 = new cl_6((cl_base*)ob_parent->children.at(i));
ob_6->set_object_name(nameChild);
get_state((cl_base*)ob_parent->children.at(i)) > 0
Нет
Да
ob_6->set_state(state);
ob_6->set_state(0);
return;
25
ob_parent->it_child = ob_parent->children.begin()
25a
while (ob_parent->it_child != ob_parent->children.end())
addNewChild((*(ob_parent->it_child)), nameParent, nameChild, state, selectFamily);
ob_parent->it_child++;
while (ob_parent->it_child != ob_parent->children.end())
Выход

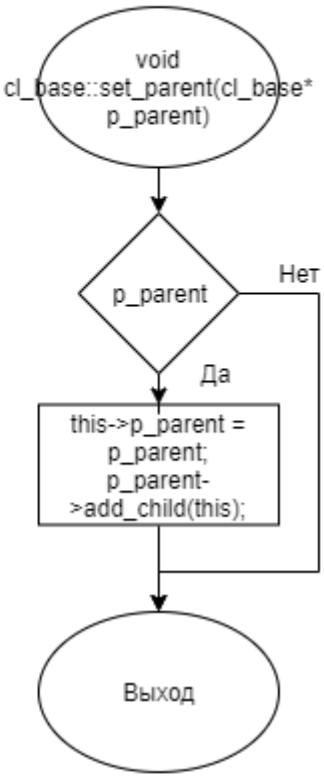

cl_base::cl_base(cl_base* p_parent)
set_object_name("cl_base");
if (p_parent)
Да
Нет
this->p_parent = p_parent; p_parent->add_child(this);
this->p_parent = 0;
Выход

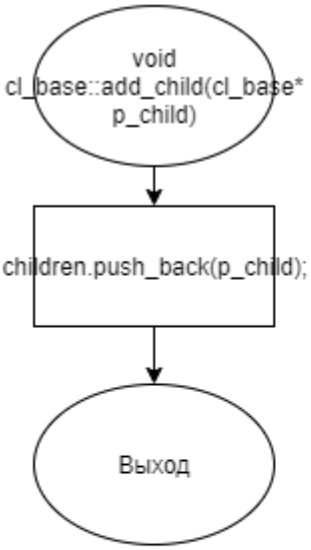

cl_base::cl_base(cl_base* p_parent, bool id)
set_object_name("cl_base");
if (p_parent)
Да
Нет
this->p_parent = p_parent; p_parent->addChildInfoSender(this
this->p_parent = 0;
Выход

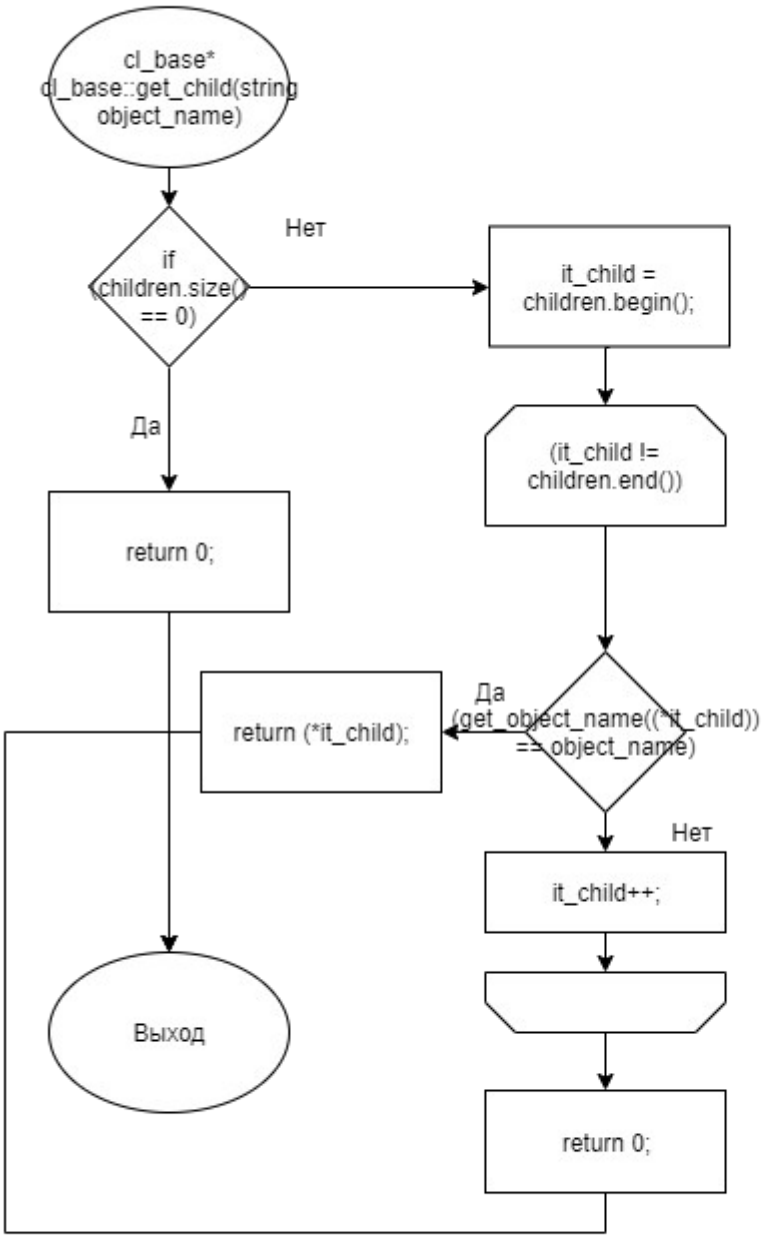

cl_base::cl_base(cl_base* p_parent, bool infoSender, bool strorageMess)
if (infoSender)
Нет
Да
if (strorageMess)
Нет
Да
set_object_name("cl_base");
set_object_name("cl_base");
if (p_parent)
Да
Нет
if (p_parent)
Да
Нет
this->p_parent = p_parent; p_parent->addChildInfoSender(thi
this->p_parent = 0;
this->p_parent = p_paren p_parent->addChildstorageMess(th
this->p_parent = 0;
Выход

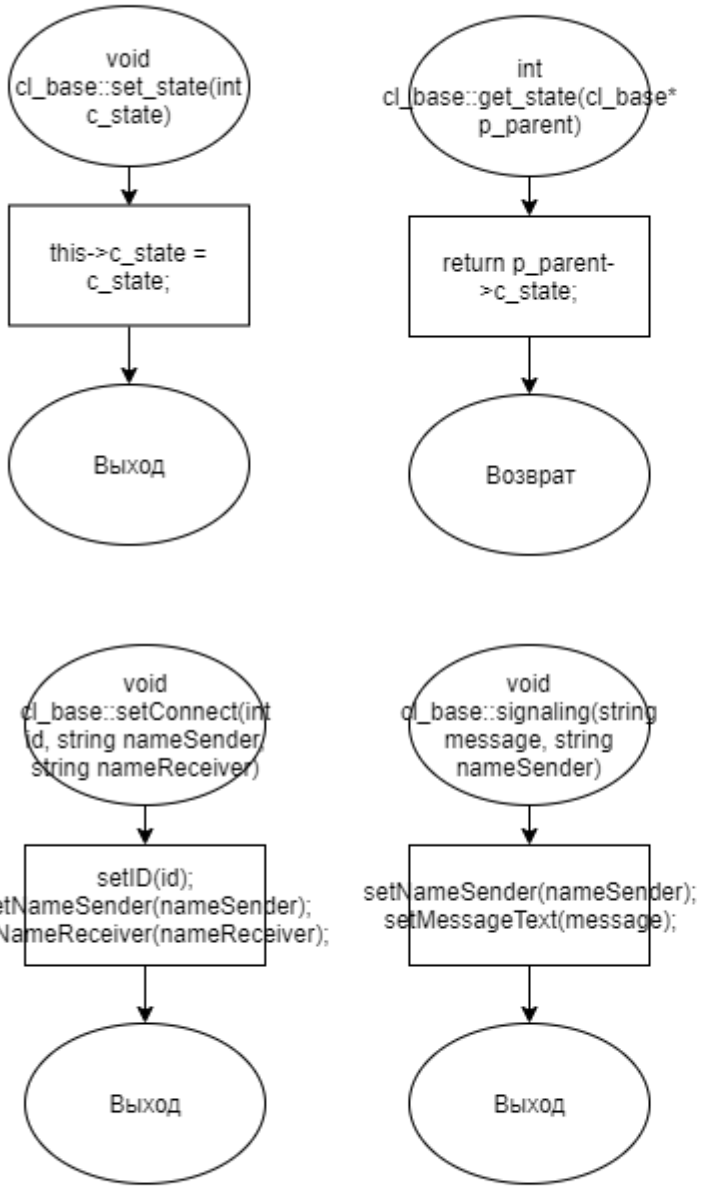

void cl_base::set_object_name(string object_name)
this->object_name = object_name;
Выход

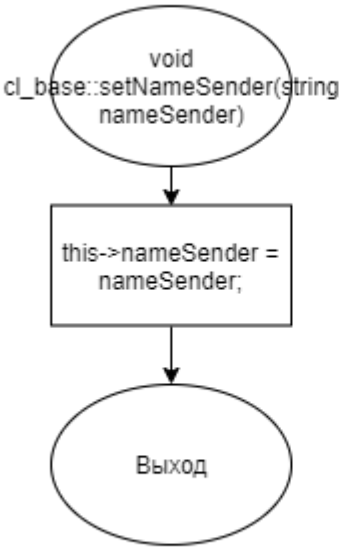

string cl_base::get_object_name(cl_base* p_parent)
return p_parent->object_name;
Возврат

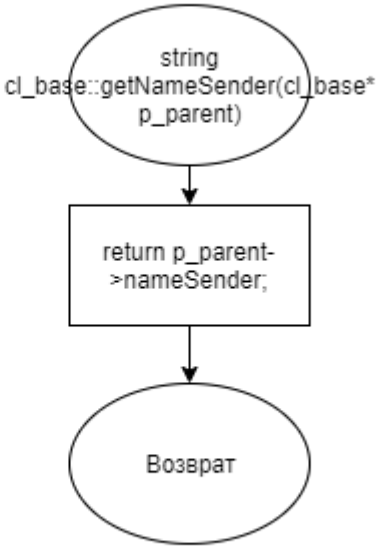

void cl_base::set_parent(cl_base* p_parent)
p_parent
Нет
Да
this->p_parent = p_parent; p_parent->add_child(this);
Выход

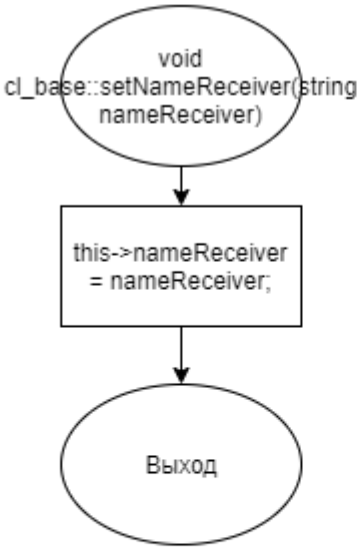

void cl_base::add_child(cl_base* p_child)
children.push_back(p_child);
Выход

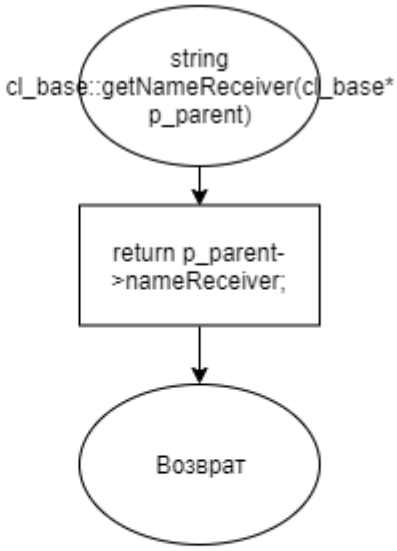

cl_base*
cl_base::get_child(string
object_name)
if
(children.size()
== 0)
Нет
Да
it_child =
children.begin();
(it_child !=
children.end())
return 0;
return (*it_child);
Да
(get_object_name((*it_child))
== object_name)
Нет
it_child++;
Выход
return 0;

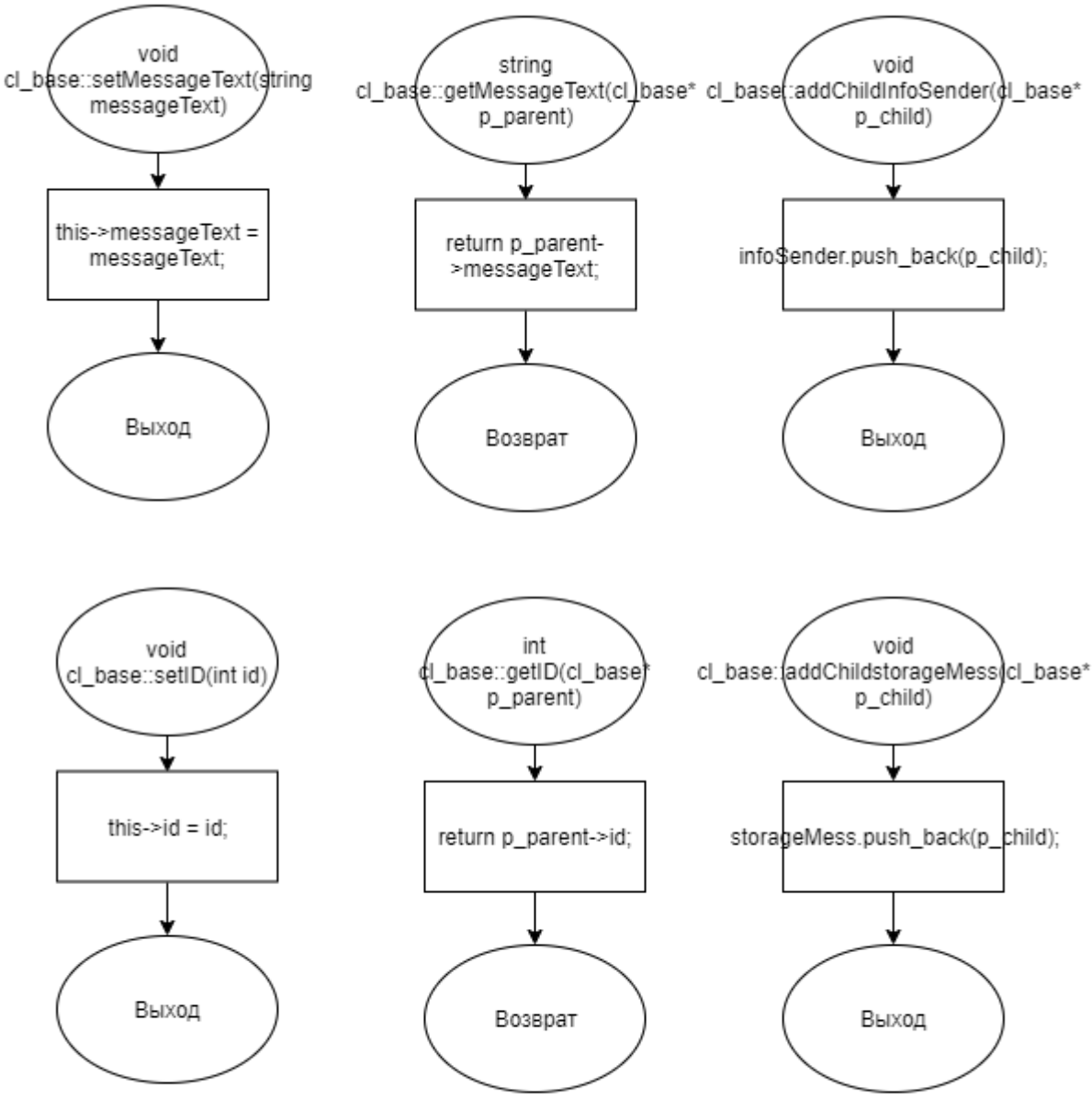

void cl_base::set_state(int c_state)
this->c_state = c_state;
Выход
int cl_base::get_state(cl_base* p_parent)
return p_parent->c_state;
Возврат
void cl_base::setConnect(int id, string nameSender, string nameReceiver)
setID(id);
setNameSender(nameSender);
setNameReceiver(nameReceiver);
Выход
void cl_base::signaling(string message, string nameSender)
setNameSender(nameSender);
setMessageText(message);
Выход

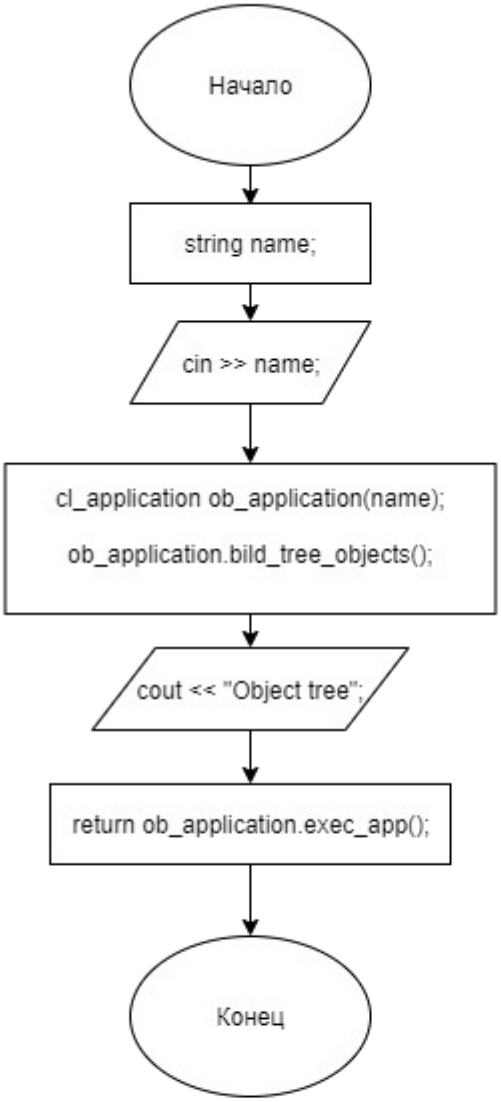

void cl_base::setNameSender(string nameSender)
this->nameSender = nameSender;
Выход


string cl_base::getNameSender(cl_base* p_parent)
return p_parent->nameSender;
Возврат


void cl_base::setNameReceiver(string nameReceiver)
this->nameReceiver = nameReceiver;
Выход


string cl_base::getNameReceiver(cl_base* p_parent)
return p_parent->nameReceiver;
Возврат


void cl_base::setMessageText(string messageText)
this->messageText = messageText;
Выход
string cl_base::getMessageText(cl_base* p_parent)
return p_parent->messageText;
Возврат
void cl_base::addChildInfoSender(cl_base* p_child)
infoSender.push_back(p_child);
Выход
void cl_base::setID(int id)
this->id = id;
Выход
int cl_base::getID(cl_base* p_parent)
return p_parent->id;
Возврат
void cl_base::addChildstorageMess(cl_base* p_child)
storageMess.push_back(p_child);
Выход



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 infoSender, bool strorageMess) :
cl_base(p_parent, infoSender, strorageMess) {}
cl_2::cl_2(cl_base* p_parent, bool id) : cl_base(p_parent, id) {}
```

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 infoSender, bool strorageMess);
cl_2(cl_base* p_parent, bool id);
};

#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;
        };
        scanConnects(this);
        scanSignals(this);
}

void cl_application::scanConnects(cl_base* ob_parent) {
        while (true) {
                int id;
                string sender, receiver;
                cin >> id;
                if (id == 0)
                        break;
                else {
                        cin >> sender >> receiver;
                        cl_2* connectStart;
                        connectStart = new cl_2((cl_base*)ob_parent, true,
false);
                        connectStart->setConnect(id, sender, receiver);
                }
        }
}


void cl_application::scanSignals(cl_base* ob_parent) {
        while (true) {
                string sender;
                string message;
                cin >> sender;
```

```
                if (sender == "endsignals") {
                        break;
                }
                else {
                        cin >> message;
                        cl_2* messageStart;
                        messageStart = new cl_2((cl_base*)ob_parent, false,
true);
                        messageStart->signaling(message, sender);
                }
        }
}

void cl_application::printConnects(cl_base* ob_parent) {
        cout << endl << "Set connects";
        if (ob_parent->infoSender.size() == 0)
                return;
        ob_parent->it_iS = ob_parent->infoSender.begin();
        while (ob_parent->it_iS != ob_parent->infoSender.end()) {
                cout << endl << getID((*(ob_parent->it_iS))) << " " <<
getNameSender((*(ob_parent->it_iS))) << " " << getNameReceiver(*(ob_parent-
>it_iS));
                ob_parent->it_iS++;
        }
}



void cl_application::printInfoWithMessage(cl_base* ob_parent) {
        cout << endl << "Emit signals";
        if (ob_parent->storageMess.size() == 0)
                return;
        ob_parent->it_sM = ob_parent->storageMess.begin();
        while (ob_parent->it_sM != ob_parent->storageMess.end()) {

                returnMessage(this, getNameSender(*(ob_parent->it_sM)),
getMessageText((*(ob_parent->it_sM))));

                ob_parent->it_sM++;
        }
}

void cl_application::returnMessage(cl_base* ob_parent, string sender, string
message) {
        if (ob_parent->infoSender.size() == 0)
                return;
        ob_parent->it_iS = ob_parent->infoSender.begin();
        while (ob_parent->it_iS != ob_parent->infoSender.end()) {
                if (sender == getNameSender((*(ob_parent->it_iS)))) {

                        cout << endl << "Signal to " <<
getNameReceiver((*(ob_parent->it_iS))) << " Text: " << sender << " -> " <<
message;
                }
                ob_parent->it_iS++;
        }
}
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);
        printConnects(this);
        printInfoWithMessage(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();
int exec_app();
void show_object_state();
void addNewChild(cl_base* ob_parent, string nameParent, string nameChild, int
state, int selectFamily);
//START NEW 3.4
void scanConnects(cl_base* ob_parent);
void scanSignals(cl_base* ob_parent);
void printConnects(cl_base* ob_parent);
void printInfoWithMessage(cl_base* ob_parent);
void returnMessage(cl_base* ob_parent, string sender, string message);
//END NEW 3.4
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;
        }
}

//START NEW 3.4

cl_base::cl_base(cl_base* p_parent, bool infoSender, bool strorageMess)
{
        if (infoSender) {
                set_object_name("cl_base");
                if (p_parent) {
                        this->p_parent = p_parent;
                        p_parent->addChildInfoSender(this);
                }
                else {
                        this->p_parent = 0;
                }
        }
        if (strorageMess) {
                set_object_name("cl_base");
                if (p_parent) {
                        this->p_parent = p_parent;
                        p_parent->addChildstorageMess(this);
                }
                else {
                        this->p_parent = 0;
                }
        }
}


cl_base::cl_base(cl_base* p_parent, bool id)
{
                set_object_name("cl_base");
                if (p_parent) {
                        this->p_parent = p_parent;
                        p_parent->addChildInfoSender(this);
                }
                else {
                        this->p_parent = 0;
                }
}

//END NEW 3.4
```

```
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;
}

//START NEW 3.4
void cl_base::setConnect(int id, string nameSender, string nameReceiver) {
        setID(id);
        setNameSender(nameSender);
        setNameReceiver(nameReceiver);
}
void cl_base::deleteConnect() {

}
void cl_base::signaling(string message, string nameSender) {
```

```
        setNameSender(nameSender);
        setMessageText(message);
}

void cl_base::setNameSender(string nameSender) {
        this->nameSender = nameSender;
}
string cl_base::getNameSender(cl_base* p_parent) {
        return p_parent->nameSender;
}

void cl_base::setNameReceiver(string nameReceiver) {
        this->nameReceiver = nameReceiver;
}
string cl_base::getNameReceiver(cl_base* p_parent) {
        return p_parent->nameReceiver;
}

void cl_base::setMessageText(string messageText) {
        this->messageText = messageText;
}
string cl_base::getMessageText(cl_base* p_parent) {
        return p_parent->messageText;
}

void cl_base::setID(int id) {
        this->id = id;
}
int cl_base::getID(cl_base* p_parent) {
        return p_parent->id;
}


void cl_base::addChildInfoSender(cl_base* p_child) {
        infoSender.push_back(p_child);
}
void cl_base::addChildstorageMess(cl_base* p_child) {
        storageMess.push_back(p_child);
}


//END NEW 3.4
```

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 3.4
cl_base(cl_base* p_parent, bool infoSender, bool strorageMess);
//only add vector ID
cl_base(cl_base* p_parent, bool id);
//End 3.4
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);
//New Start 3.4
void setConnect(int id, string nameSender, string nameReceiver);
void deleteConnect();
void signaling(string message, string nameSender);

void setNameSender(string nameSender);
string getNameSender(cl_base* p_parent);
void setNameReceiver(string nameReceiver);
string getNameReceiver(cl_base* p_parent);
void setMessageText(string messageText);
string getMessageText(cl_base* p_parent);
void setID(int id);
int getID(cl_base* p_parent);

void addChildInfoSender(cl_base* p_child);
void addChildstorageMess(cl_base* p_child);

// -> Vector info Sender
vector <cl_base*> infoSender;
vector <cl_base*> ::iterator it_iS;

// -> Vector Message
vector <cl_base*> storageMess;
vector <cl_base*> ::iterator it_sM;
//End Start 3.4

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;
//New 3.4
int id;
string nameSender;
string nameReceiver;
string messageText;
};

#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 3 1 root obj2 3 1 endtree 1 obj1 obj2 2 obj1 root 0 obj1 TEXT endsignals	Object tree root obj1 obj2 Set connects 1 obj1 obj2 2 obj1 root Emit signals Signal to obj2 Text: obj1 -> TEXT Signal to	Object tree root obj1 obj2 Set connects 1 obj1 obj2 2 obj1 root Emit signals Signal to obj2 Text: obj1 -> TEXT

	root Text: obj1 -> TEXT	Signal to root Text: obj1 -> TEXT
root root ob_1 3 1 root ob_3 3 4 ob_1 ob_2 2 -1 ob_3 ob_6 2 1 ob_3 ob_4 2 1 ob_4 ob_5 3 1 ob_6 ob_7 2 -7 endtree 1 ob_2 ob_5 2 ob_2 ob_3 3 root ob_2 4 root root 0 ob_2 mama ob_2 papa ob_2 TEXT root AVRORA endsignals	Object tree root ob_1 ob_2 ob_3 ob_6 ob_7 ob_4 ob_5 Set connects 1 ob_2 ob_5 2 ob_2 ob_3 3 root ob_2 4 root root Emit signals Signal to ob_5 Text: ob_2 -> mama Signal to ob_3 Text: ob_2 -> mama Signal to ob_5 Text: ob_2 -> papa Signal to ob_3 Text: ob_2 -> papa Signal to ob_5 Text: ob_2 -> TEXT Signal to ob_3 Text: ob_2 -> TEXT Signal to ob_2 Text: root -> AVRORA Signal to root Text: root -> AVRORA	Object tree root ob_1 ob_2 ob_3 ob_6 ob_7 ob_4 ob_5 Set connects 1 ob_2 ob_5 2 ob_2 ob_3 3 root ob_2 4 root root Emit signals Signal to ob_5 Text: ob_2 -> mama Signal to ob_3 Text: ob_2 -> mama Signal to ob_5 Text: ob_2 -> papa Signal to ob_3 Text: ob_2 -> papa Signal to ob_5 Text: ob_2 -> TEXT Signal to ob_3 Text: ob_2 -> TEXT Signal to ob_2 Text: root -> AVRORA Signal to root Text: root -> AVRORA