



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

« МИРЭА Российский технологический университет »

РТУ МИРЭА

Институт Информационных технологий

Кафедра Вычислительной техники

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине

« Объектно-ориентированное программирование »

Наименование задачи:

« КЛ_3_2 Вывод иерархического дерева »

С тудент группы

ИКБО-07-19

Ле Д..

Руководитель практики

Ассистент

Боронников А.С.

Работа представлена

«__»_____2020 г.

(подпись студента)

Оценка

(подпись руководителя)

Москва 2020

Постановка задачи

Вывод иерархического дерева объектов на консоль

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

Построить модель иерархической системы. Реализовать вывод на консоль иерархического дерева объектов в следующем виде:

```
root
    ob_1
    ob_2
    ob_3
    ob_4
    ob
    _5
    ob
    _6
    ob
    _7
```

где: root - наименование корневого объекта (приложения).

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

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

Описание входных данных

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

Описание выходных данных

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

Метод решения

Используя потоки Ввода/Вывода - cin/cout

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

Используя void show_object_state() для показать состояние объекта.

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

Используя int exes_app() для применять.

Описание алгоритма

cl_application(string name)

| № шага | Предикат | Действие |
|--------|----------|------------------------|
| 1 | | set_object_name(name); |
| 2 | | set_state(1); |

void cl_application::bild_tree_objects()

| № шага | Предикат | Действие |
|--------|--|---|
| 1 | | cl_2* ob_2; |
| 2 | | cl_3* ob_3; |
| 3 | | cl_4* ob_4; |
| 4 | | string nameParent, nameChild; |
| 5 | | int selectFamily, state; |
| 6 | while (true) | |
| 7 | | cin >> nameParent; |
| 8 | if (nameParent == text_finish) | break; |
| | else | |
| 9 | | cin >> nameChild >> selectFamily >> state; |
| 10 | if (selectFamily == 2) | |
| | else | |
| 11 | if (this->get_object_name() == nameParent) | |
| | else | |
| 12 | | ob_2 = new cl_2((cl_base*)this); |
| 13 | | ob_2->set_object_name(nameChild); |
| 14 | | ob_2->set_state(state); |
| 15 | | addNewChild(this, nameParent, nameChild, state, |
| 16 | if (selectFamily == 3) | |
| | else | |
| 17 | if (this->get_object_name() == nameParent) | |
| | else | |
| 18 | | ob_3 = new cl_3((cl_base*)this); |
| 19 | | ob_3->set_object_name(nameChild); |
| 20 | | ob_3->set_state(state); |
| 21 | | addNewChild(this, nameParent, nameChild, state, |
| 22 | if (selectFamily == 4) | |
| | else | |
| 23 | if (this->get_object_name() == nameParent) | |
| | else | |

| | | |
|----|--|---|
| 24 | | ob_4 = new cl_4((cl_base*)this); |
| 25 | | ob_4->set_object_name(nameChild); |
| 26 | | ob_4->set_state(state); |
| 27 | | addNewChild(this, nameParent, nameChild, state, |
| 28 | | break; |

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

| № шага | Предикат | Действие |
|--------|---|--|
| 1 | | cl_2* ob_2; cl_3* ob_3; cl_4* ob_4; |
| 2 | if (selectFamily == 2) else | |
| 3 | for (size_t i = 0; i < ob_parent->children.size(); i++) i = ob_parent->children.size() | |
| 4 | if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent) else | ob_2 = new cl_2((cl_base*)ob_parent-> |
| 5 | | ob_2->set_object_name(nameChild); |
| 6 | if (get_state((cl_base*)ob_parent->children.at(i)) > 0) else | ob_2->set_state(state); ob_2->set_state(0); |
| 7 | | return; |
| 8 | if (selectFamily == 3) else | |
| 9 | for (size_t i = 0; i < ob_parent->children.size(); i++) i = ob_parent->children.size() | |
| 10 | if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent) else | ob_3 = new cl_3((cl_base*)ob_parent-> |
| 11 | | ob_3->set_object_name(nameChild); |
| 12 | if (get_state((cl_base*)ob_parent->children.at(i)) > 0) else | ob_3->set_state(state); ob_3->set_state(0); |
| 13 | | return; |
| 14 | if (selectFamily == 4) else | |
| 15 | for (size_t i = 0; i < ob_parent->children.size(); i++) i = ob_parent->children.size() | |

| | | |
|----|--|---|
| 16 | if (get_object_name((cl_base*)ob_parent->children.at(i)) == nameParent) else | ob_4 = new cl_4((cl_base*)ob_parent->children.at(i)); |
| 17 | | ob_4->set_object_name(nameChild); |
| 18 | if (get_state((cl_base*)ob_parent->children.at(i)) > 0) else | ob_4->set_state(state); ob_4->set_state(0); |
| 19 | | return; |
| 20 | | ob_parent->it_child = ob_parent->children.begin(); |
| 21 | while (ob_parent->it_child != ob_parent->children.end()) ob_parent->it_child == ob_parent->children.end() | addNewChild((*ob_parent->it_child), state, selectFamily); |
| 22 | | ob_parent->it_child++; |

int cl_application::exec_app()

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

void cl_application::show_object_state()

| № шага | Предикат | Действие | № перехода |
|--------|----------|---------------------------|------------|
| 1 | | show_state_next(this, 2); | |

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

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

cl_base(cl_base* p_parent)

| № шага | Предикат | Действие |
|--------|----------------------|--------------------------------|
| 1 | | set_object_name(" cl_base "); |
| 2 | if(p_parent) else | this->p_parent = p_parent; |
| 3 | | p_parent->add_child(this); |

| | | |
|---|--|---------------------|
| 4 | | this->p_parent = 0; |
|---|--|---------------------|

void cl_base::set_object_name(string object_name)\

| № шага | Предикат | Действие |
|--------|----------|----------------------------------|
| 1 | | this->object_name = object_name; |

string cl_base::get_object_name(cl_base* p_parent)

| № шага | Предикат | Действие |
|--------|----------|-------------------------------|
| 1 | | return p_parent->object_name; |

void cl_base::set_parent(cl_base* p_parent)

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

void cl_base::add_child(cl_base* p_child)

| № шага | Предикат | Действие |
|--------|----------|------------------------------|
| 1 | | children.push_back(p_child); |

cl_base* cl_base::get_child(string object_name)

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

void cl_base::set_state(int c_state)

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

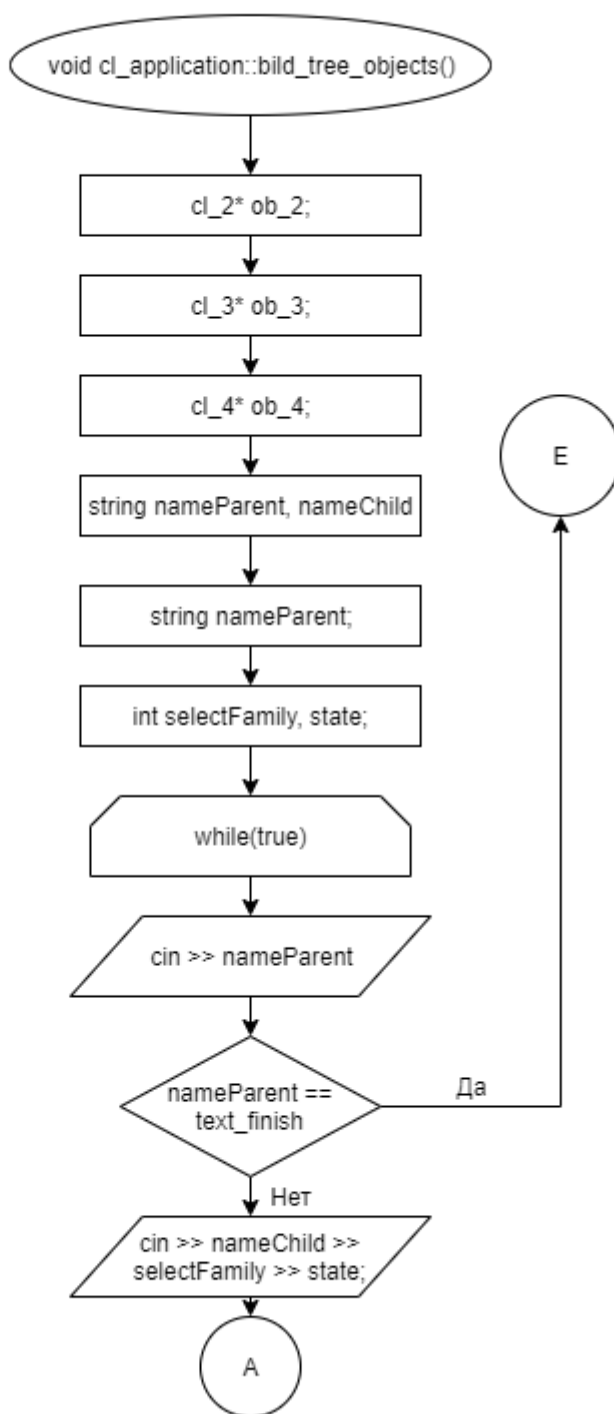
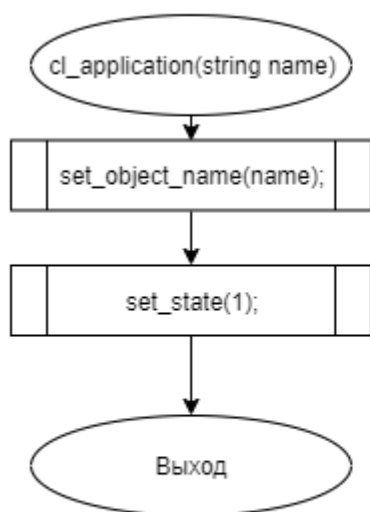
```
int cl_base::get_state(cl_base* p_parent)
```

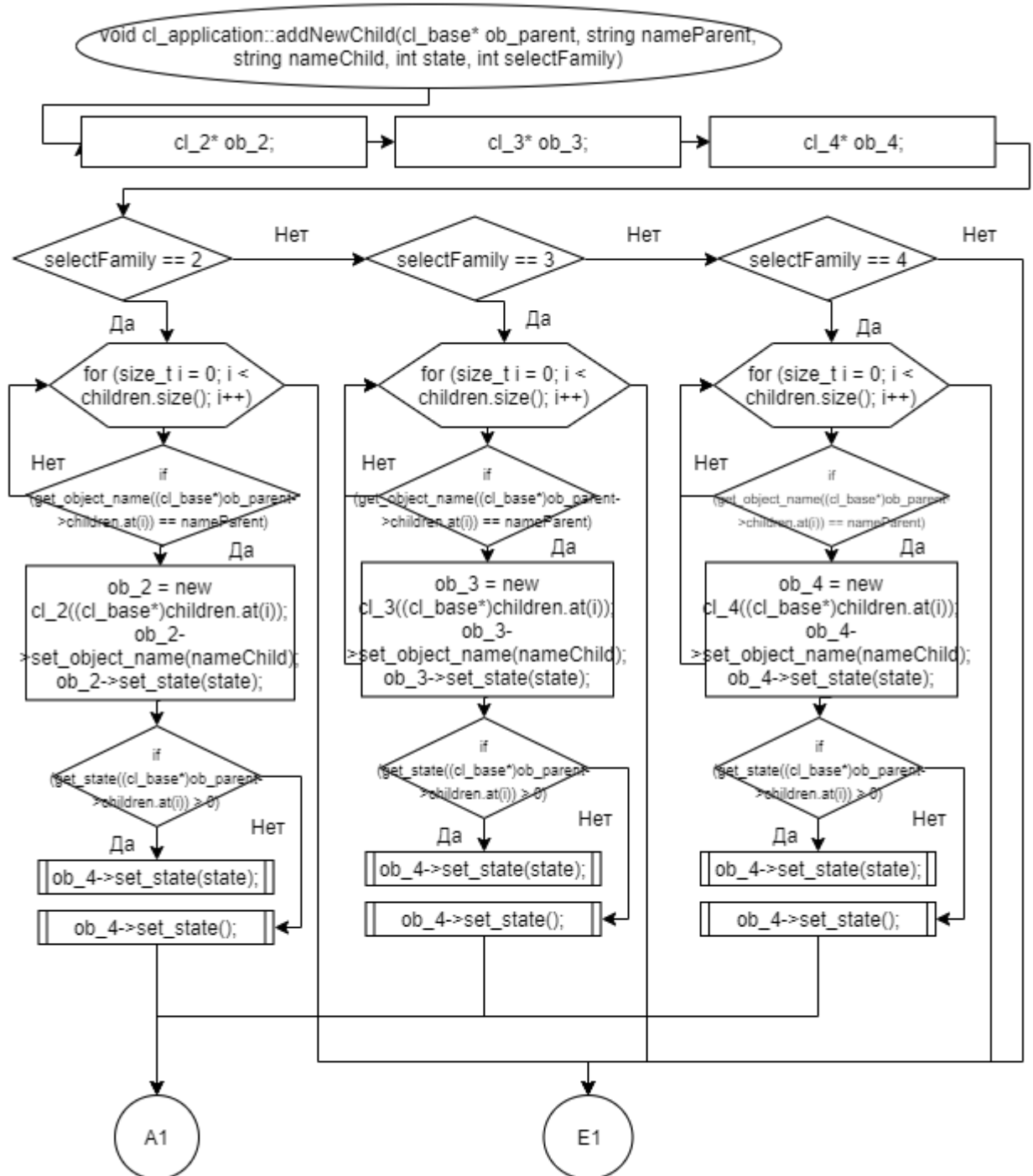
| № шага | Предикат | Действие | № п |
|--------|----------|---------------------------|-----|
| 1 | | return p_parent->c_state; | Ø |

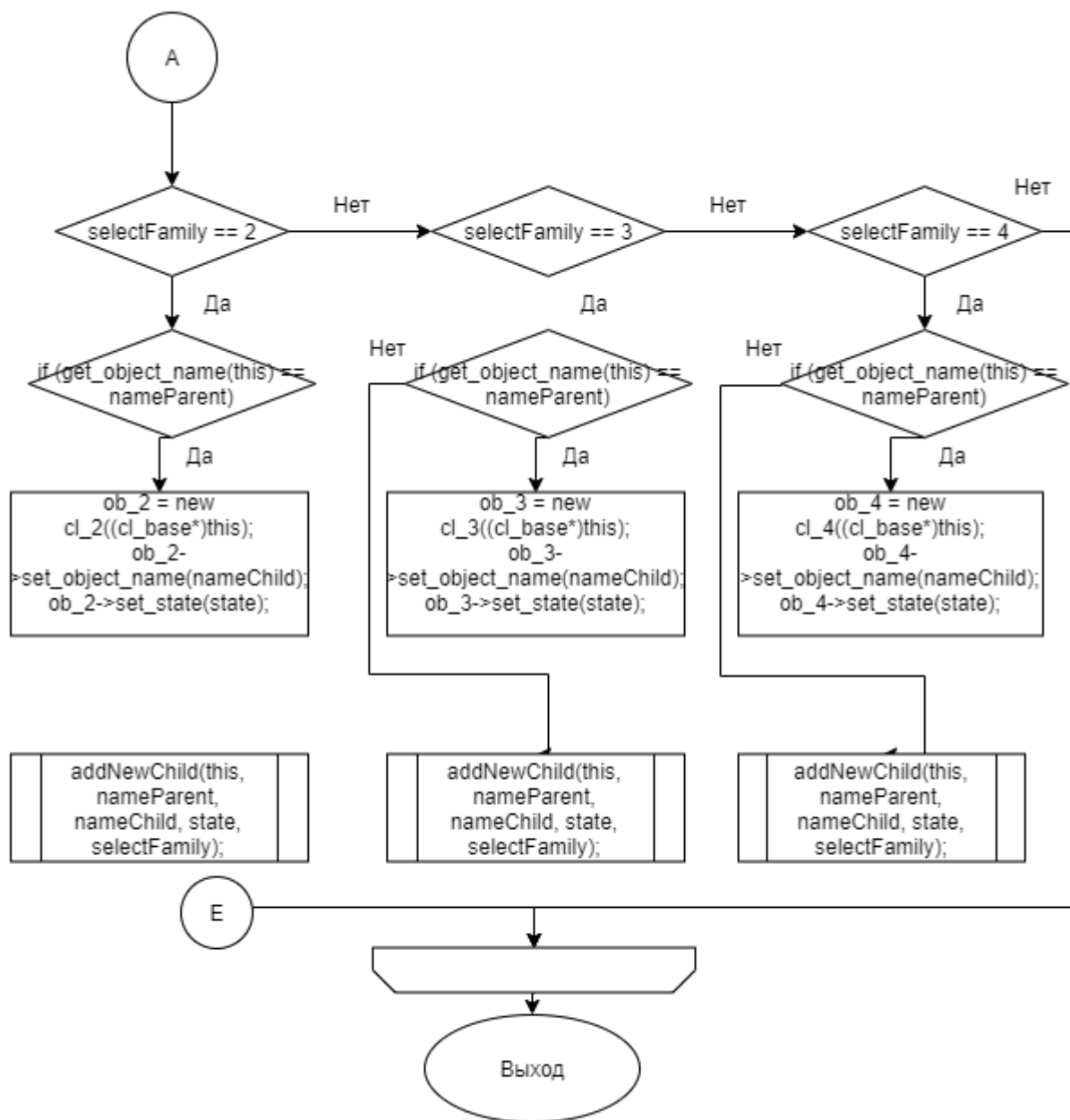
```
int main()
```

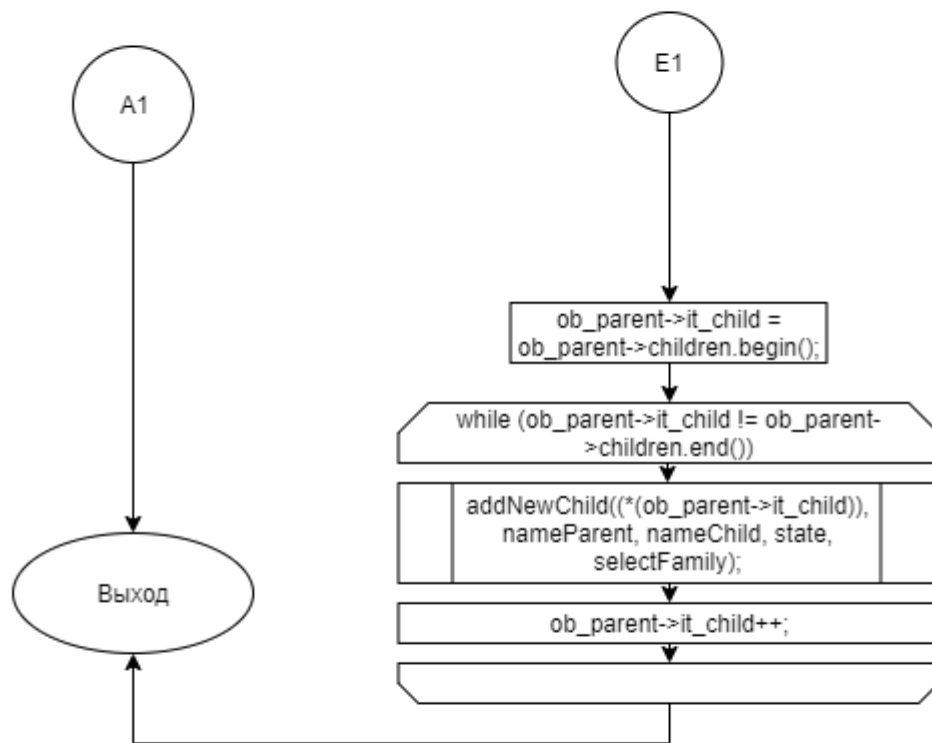
| № шага | Предикат | Действие | № п |
|--------|----------|--------------------------------------|-----|
| 1 | | string name; | 2 |
| 2 | | cin >> name; | 3 |
| 3 | | cl_application ob_application(name); | 4 |
| 4 | | ob_application.bild_tree_objects(); | 5 |
| 5 | | cout << "Object tree"; | 6 |
| 6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

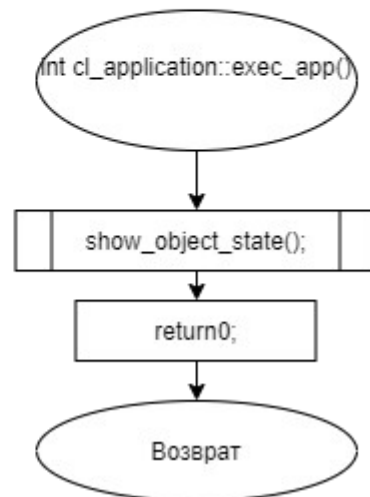
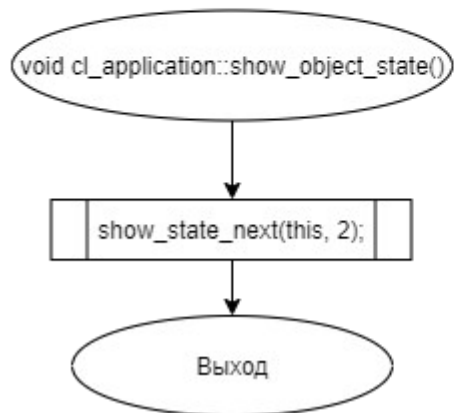
Блок-схема алгоритма

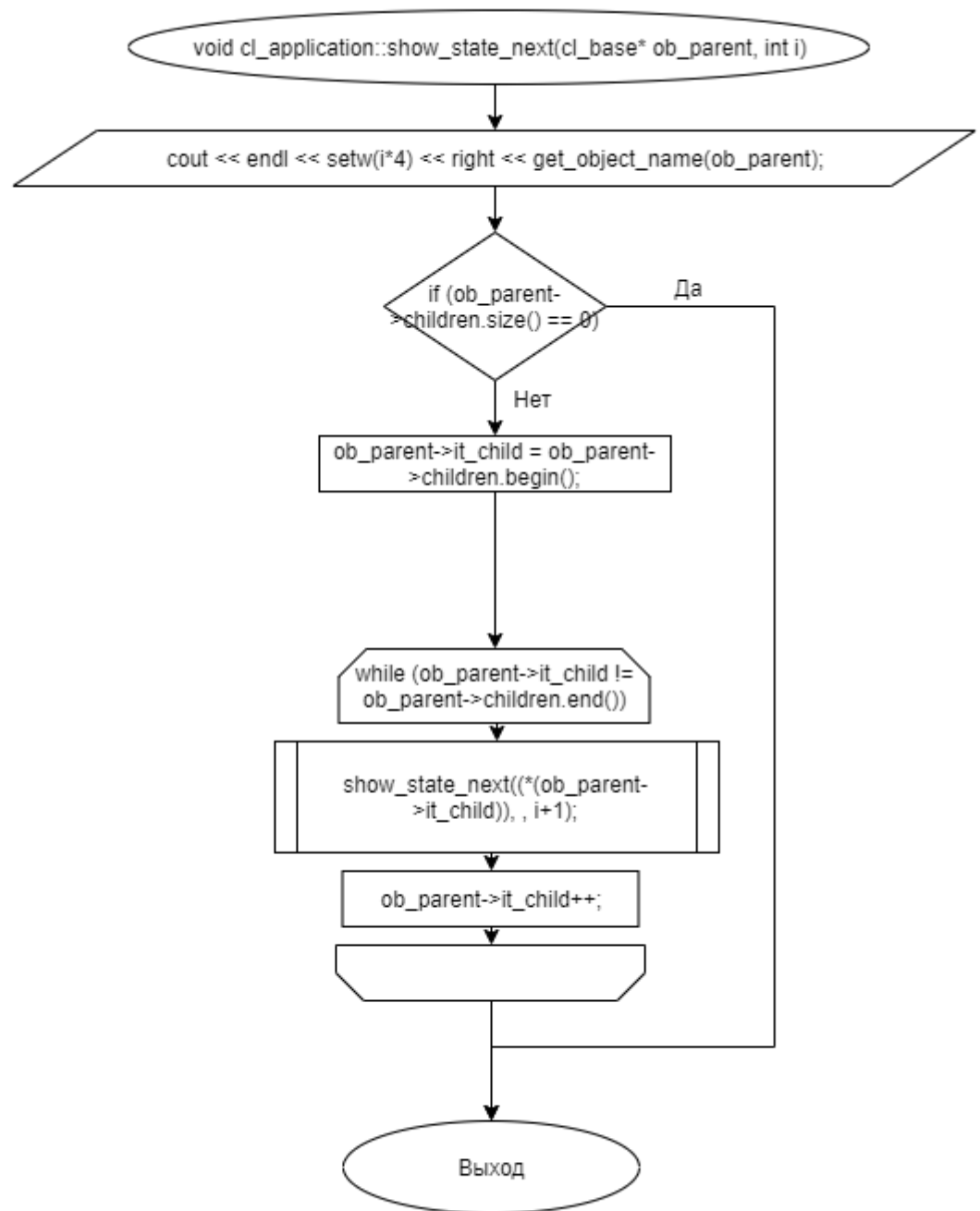


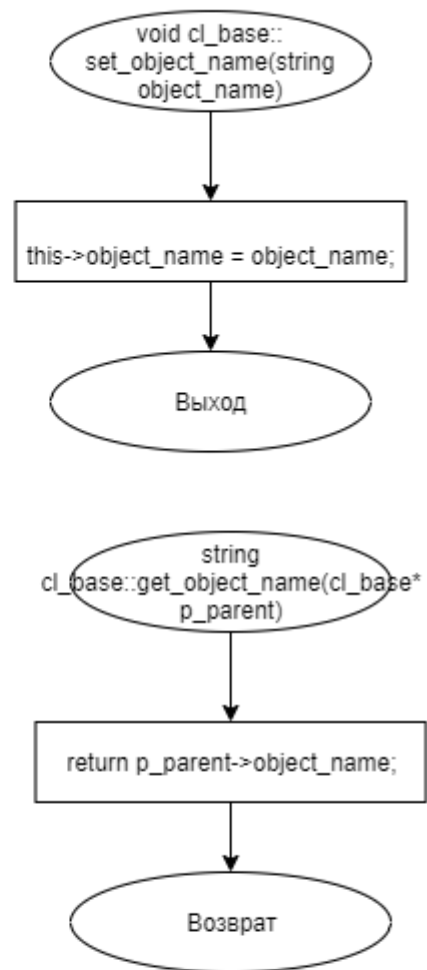
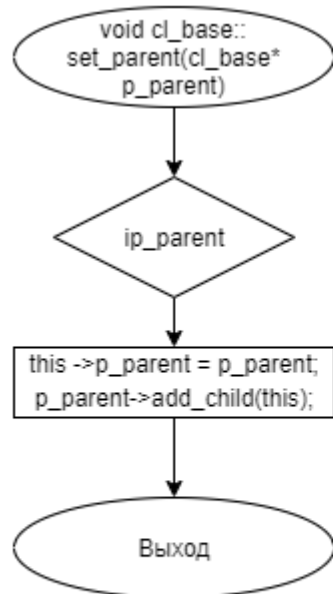
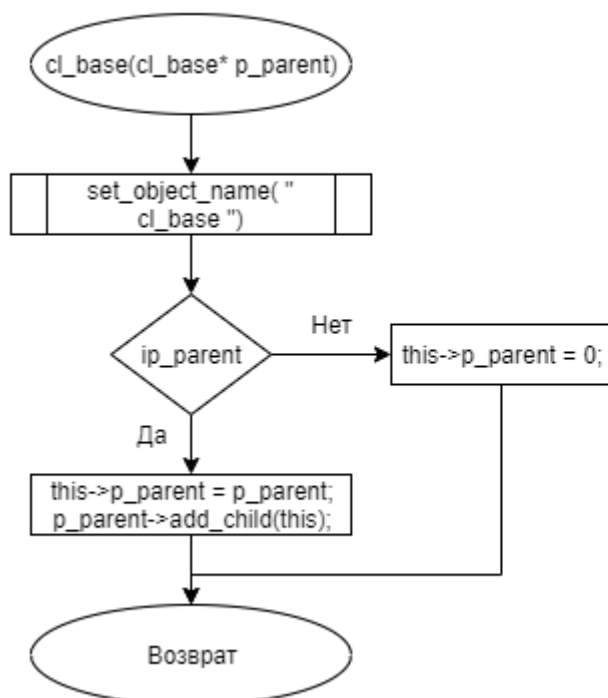


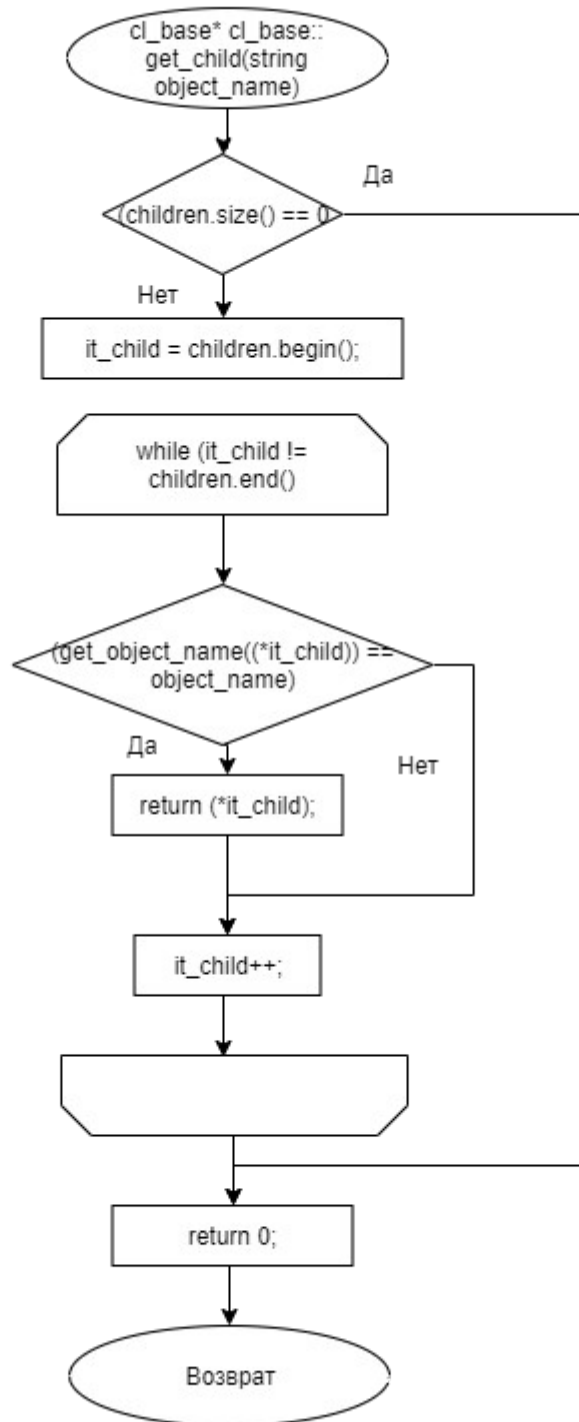
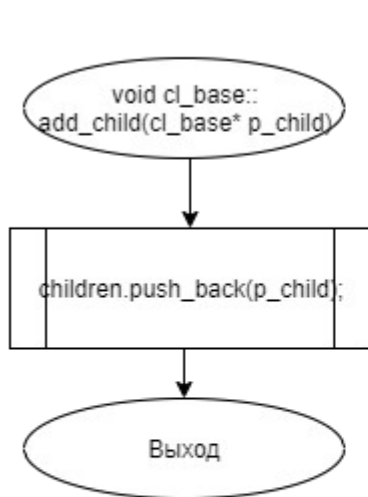


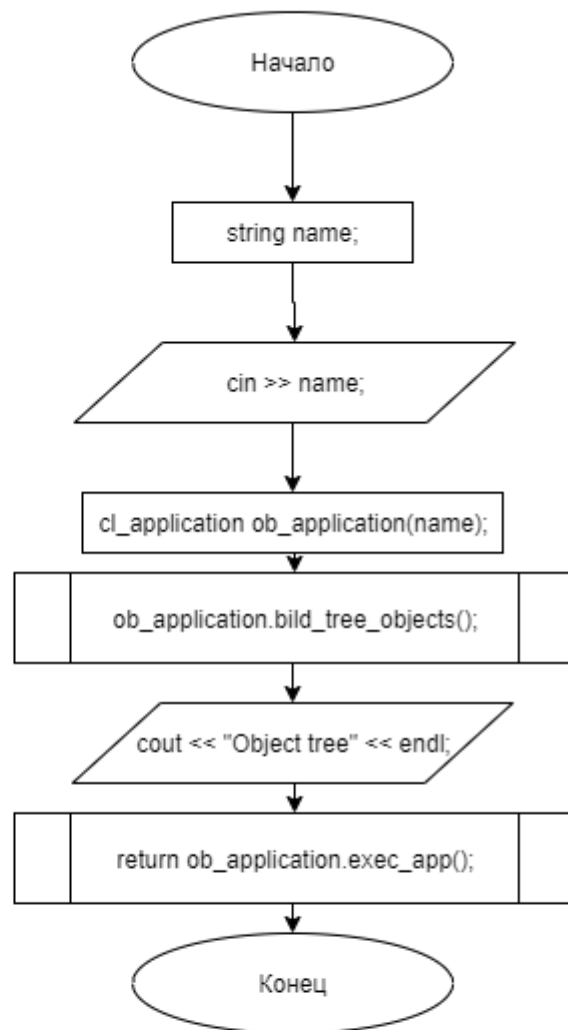
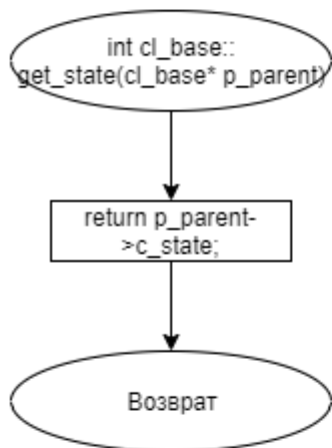
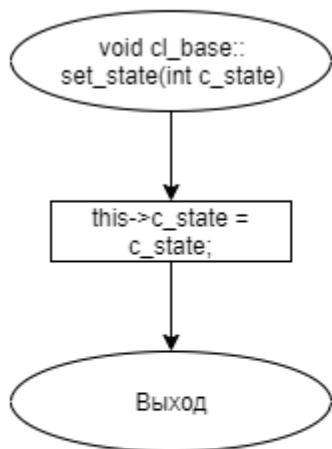












Код программы

Файл cl_2.cpp

```
#include "cl_2.h"

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

Файл cl_2.h

```
#ifndef CL_2_H
#define CL_2_H

#include "cl_base.h"

class cl_2 : public cl_base {
public:
    cl_2(cl_base* p_parent = 0);
};

#endif // CL_2_H
```

Файл cl_3.cpp

```
#include "cl_3.h"

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

Файл cl_3.h

```
#ifndef CL_3_H
#define CL_3_H

#include "cl_base.h"

class cl_3 : public cl_base {
public:
    cl_3(cl_base* p_parent = 0);
};
```

```
};  
#endif // CL_3_H
```

Файл cl_4.cpp

```
#include "cl_4.h"  
cl_4::cl_4(cl_base* p_parent) : cl_base(p_parent) {}
```

Файл cl_4.h

```
#ifndef CL_4_H  
#define CL_4_H  
  
#include "cl_base.h"  
  
class cl_4 : public cl_base {  
public:  
    cl_4(cl_base* p_parent = 0);  
};  
  
#endif // CL_4_H
```

Файл cl_5.cpp

```
#include "cl_5.h"  
cl_5::cl_5(cl_base* p_parent) : cl_base(p_parent) {}
```

Файл cl_5.h

```
#ifndef CL_5_H
```

```
#define CL_5_H

#include "cl_base.h"

class cl_5 : public cl_base {
public:
    cl_5(cl_base* p_parent = 0);
};

#endif // CL_5_H
```

Файл cl_6.cpp

```
#include "cl_6.h"

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

Файл cl_6.h

```
#ifndef CL_6_H
#define CL_6_H

#include "cl_base.h"

class cl_6 : public cl_base {
public:
    cl_6(cl_base* p_parent = 0);
};

#endif // CL_6_H
```

Файл cl_application.cpp

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

```

cl_application::cl_application(string name) {
    set_object_name(name);
    set_state(1);
}

void cl_application::bild_tree_objects() {

    cl_2* ob_2;
    cl_3* ob_3;
    cl_4* ob_4;
    string nameParent, nameChild;
    int selectFamily;
    int state;

    while (true) {
        cin >> nameParent;
        if (nameParent == text_finish)
            break;
        cin >> nameChild >> selectFamily >> state;
        if (selectFamily == 2) {
            if (get_object_name(this) == nameParent) {
                ob_2 = new cl_2((cl_base*)this);
                ob_2->set_object_name(nameChild);
                ob_2->set_state(state);
            }
            else {
                addNewChild(this, nameParent, nameChild,
state, selectFamily);
            }
        }
        else if (selectFamily == 3) {
            if (get_object_name(this) == nameParent) {
                ob_3 = new cl_3((cl_base*)this);
                ob_3->set_object_name(nameChild);
                ob_3->set_state(state);
            }
            else {
                addNewChild(this, nameParent, nameChild,
state, selectFamily);
            }
        }
        else if (selectFamily == 4) {
            if (get_object_name(this) == nameParent) {
                ob_4 = new cl_4((cl_base*)this);
                ob_4->set_object_name(nameChild);
                ob_4->set_state(state);
            }
            else {
                addNewChild(this, nameParent, nameChild,
state, selectFamily);
            }
        }
        else
            break;
    };
}

```

```

}
void cl_application::addNewChild(cl_base* ob_parent, string nameParent, string
nameChild, int state, int selectFamily) {
    cl_2* ob_2;
    cl_3* ob_3;
    cl_4* ob_4;
    if (selectFamily == 2) {
        for (size_t i = 0; i < ob_parent->children.size(); i++) {
            if (get_object_name((cl_base*)ob_parent->children.at(i)) ==
nameParent) {
                ob_2 = new cl_2((cl_base*)ob_parent->children.at(i));
                ob_2->set_object_name(nameChild);
                if (get_state((cl_base*)ob_parent->children.at(i)) >
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;
            }
        }
    }
}

```

```

        }
    }
    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, 2);
}

void cl_application::show_state_next(cl_base* ob_parent, int i) {
    cout << endl << setw(i*4) << right << 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_base.h"

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

```



```

private:

void show_state_next(cl_base* ob_parent, int i);

};

#endif // CL_APPLICATION_H

```

Файл cl_base.cpp

```

#include "cl_base.h"

cl_base::cl_base(cl_base* p_parent)
{
    set_object_name("cl_base");
    if (p_parent) {
        this->p_parent = p_parent;
        p_parent->add_child(this);
    }
    else {
        this->p_parent = 0;
    }
}

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

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

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

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

cl_base* cl_base::get_child(string object_name) {
    if (children.size() == 0) return 0;
    it_child = children.begin();
    while (it_child != children.end()) {

```

```

        if (get_object_name((*it_child)) == object_name) {
            return (*it_child);
        }
        it_child++;
    }
    return 0;
}

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

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

```

Файл cl_base.h

```

#ifndef CL_BASE_H
#define CL_BASE_H

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

using namespace std;

class cl_base
{
public:
    cl_base(cl_base* p_parent = 0);
    void set_object_name(string object_name);
    string get_object_name(cl_base* p_parent);
    void set_parent(cl_base* p_parent);
    void add_child(cl_base* p_child);
    cl_base* get_child(string object_name);
    void set_state(int c_state);
    int get_state(cl_base* p_parent);

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

private:
    string object_name;

    cl_base* p_parent;

```

```

int c_state;

};

#endif // CL_BASE_H

```

Файл main.cpp

```

#include <iostream>
using namespace std;

#include "cl_application.h"

int main()
{
    string name;
    cin >> name;
    cl_application ob_application(name);
    ob_application.bild_tree_objects();
    cout << "Object tree";
    return ob_application.exec_app();
}

```

Тестирование

| Входные данные | Ожидаемые выходные данные | Фактические выходные данные |
|--|---|---|
| app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 3 -1 object_2 object_5 3 1 app_root object_3 3 1 object_2 object_6 2 1 object_1 | Object tree app_root object_1 object_7 object_2 object_4 object_5 object_6 object_3 | Object tree app_root object_1 object_7 object_2 object_4 object_5 object_6 object_3 |

| | | |
|---|---|---|
| object_7 2 1 endtree | | |
| app_root app_root object_1 3 1 app_root object_2 2 1 object_2 object_4 3 -1 object_2 object_5 3 1 app_root object_3 3 1 object_2 object_6 2 1 object_1 object_7 2 1 object_7 object_8 2 1 endtree | Object tree app_root object_1 object_7 object_8 object_2 object_4 object_5 object_6 object_3 | Object tree app_root object_1 object_7 object_8 object_2 object_4 object_5 object_6 object_3 |