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

Класс cl_base

добавлены поля:

- поле connection (тип данных-структура; private; поле отвечающие за связь между обьектами)

добавлены методы:

- get_level() - возвращает номер класса обьекта public
- set_conection() - созданите новой связи между обьектами pubcli
- delete_conection() - удаление сущ связи public
- emit_signal() - сделать сигнал public

Класс cl_application

добавлены методы:

- connection_input() - ввод и вывод задавыемых связей public
- output() - определение методов сигнала и из вызова Public
- p_signal() - метод сигнала Public

Код программы

Файл cl_1.cpp

```
#include "cl_1.h"
#include <iostream>
#include <string>

using namespace std;
cl_1::cl_1(Base* parent,int state,int classNum, string
name) :Base(parent,state, classNum, name) {}
void cl_1::p_signal(string& s)
{
        s = this->getName() + " -> " + s;
}

void cl_1::p_handler(string s)
{
cout << endl << "Signal to " << this->getName() << " Text: " << s;
}
```

Файл cl_1.h

```
#ifndef CL_1_H
#define CL_1_H

#include "cl_base.h"

class cl_1 : public Base {

public:
        cl_1(Base*,int,int, string);
        void p_signal(string&); //<5B>4 A83=0;0
        void p_handler(string); //<5B>4 >1@01>BG8:0

};

#endif
```

Файл cl_2.cpp

```
#include "cl_2.h"
#include <iostream>
#include <string>
```

```
using namespace std;

cl_2::cl_2(Base* parent, int state,int classNum, string name) :Base(parent,
state,classNum, name) {}

void cl_2::p_signal(string& s)
{
        s = this->getName() + " -> " + s;
}

void cl_2::p_handler(string s)
{
        cout << endl << "Signal to " << this->getName() << " Text: " << s;
}
```

Файл cl_2.h

```
#ifndef CL_2_H
#define CL_2_H
#include "cl_base.h"

class cl_2 : public Base {

public:
        cl_2(Base*, int, int, string);
        void p_signal(string&); //<5B>4 A83=0;0
        void p_handler(string); //<5B>4 >1@01>BG8:0

};

#endif
```

Файл cl_application.cpp

```
#include "cl_application.h"
#include "cl_1.h"
#include "cl_2.h"

#include <iostream>

using namespace std;

Application::Application(Base* parent, int state, int classNum) :
```

```
Base(parent,state, classNum)
{

}

void Application::buildTree()
{
        string parent_name, child_name;
        int classNum, state;
        cin >> parent_name;
        setName(parent_name);
        temp_parent = this;
        root_parent = this;
        do {
                cin >> parent_name;
                if (parent_name == "endtree") return;
                temp_parent = find(parent_name);
                cin >> child_name >> classNum >> state;
                switch (classNum)
                {
                        case 2:
                                temp_child = new cl_1(temp_parent,
state,classNum,child_name);
                                break;
                        case 3:
                                temp_child = new cl_2(temp_parent,
state,classNum,child_name);
                                break;
                }
        } while (true);
}

void Application::connections_input()
{
        Base* sender;
        Base* reciever;
        TYPE_SIGNAL signal;
        TYPE_HANDLER handler;
        int number;
        string emitter, target;
        cout << endl << "Set connects";
        while (true){
                cin >> number;
                if (number == 0){
                        break;
                }
                cin >> emitter >> target;
                cout << endl;
                cout << number << " " << emitter << " " << target;
                sender = this->find(emitter);
                reciever = this->find(target);
                switch (sender->getClass())
                {
                        case 1:
                                signal = SIGNAL_D(Application::p_signal);
                                break;
                        case 2:
                                signal = SIGNAL_D(cl_1::p_signal);
                                break;
                        case 3:
```

```
                                signal = SIGNAL_D(cl_2::p_signal);
                                break;
                        default:
                                signal = 0;
                }
                switch (reciever->getClass())
                {
                        case 1:
                                handler = HANDLER_D(Application::p_handler);
                                break;
                        case 2:
                                handler = HANDLER_D(cl_1::p_handler);
                                break;
                        case 3:
                                handler = HANDLER_D(cl_2::p_handler);
                                break;
                        default:
                                handler = 0;
                }
                sender->set_connection(signal, reciever, handler);
        }

}

void Application::output()
{
        Base* curr_obj;
        string emitter, command;
        TYPE_SIGNAL signal;
        cout << endl << "Emit signals";
        while (true)
        {
                cin >> emitter;
                if (emitter == "endsignals"){
                        break;
                }
                cin >> command;


                curr_obj = find(emitter);
                switch (curr_obj->getClass())
                {
                        case 1:
                                signal = SIGNAL_D(Application::p_signal);
                                break;
                        case 2:
                                signal = SIGNAL_D(cl_1::p_signal);
                                break;
                        case 3:
                                signal = SIGNAL_D(cl_2::p_signal);
                                break;
                        default:
                                signal = 0;
                }
                curr_obj->emit_signal(signal, command);
        }
```

```
}

void Application::p_signal(string& s)
{
        s = this->getName() + " -> " + s;
}

void Application::p_handler(string s)
{
        cout << endl << "Signal to " << this->getName() << " Text: " << s;
}

int Application::execute() {
        cout << "Object tree"<<endl;
        printNames(4);
        this->connections_input();
        this->output();
        return 0;

}
```

Файл cl_application.h

```
#ifndef CL_APPLICATION_H
#define CL_APPLICATION_H

#include "cl_base.h"

class Application : public Base {

        Base* temp_parent;
        Base* temp_child;
        Base* root_parent;

public:
        Application(Base* parent = 0, int state = 1,int classNum=1);
        void buildTree();
        void connections_input();
        void output();
        void p_signal(string&);
        void p_handler(string);
        int execute();

};

#endif
```

Файл cl_base.cpp

```
#include "cl_base.h"
#include <iostream>

using namespace std;

Base::Base(Base* parent, int state, int classNum, string name) {
        setParent(parent);
        setName(name);
        setState(state);
        setClass(classNum);
}

void Base::setName(string name)
{
        this->name = name;

}

void Base::setState(int state)
{
        this->state = state;
}

void Base::setClass(int classNum)
{
        this->classNum = classNum;
}

string Base::getName()
{
        return this->name;
}

int Base::getState()
{
        return this->state;
}

void Base::setParent(Base* parent)
{
        this->parent = parent;
        if (parent) {
                parent->children.push_back(this);
        }
}



Base* Base::getParent()
{
        return this->parent;
}
```

```
void Base::printNames(int k)
{
        if (this->parent == 0)
        {
                cout << this->name;
        }
        if (children.empty()) return;
        children_iterator = children.begin();
        while (children_iterator != children.end()) {
                cout << endl;
                for (int i = 0; i < k; i++) cout << " ";
                cout << (*children_iterator)->getName();
                (*children_iterator)->printNames(k + 4);
                children_iterator++;
        }
}

void Base::printState()
{
        if (this->parent == 0){
                cout << "The object " << this->name << " is ready";
        }
        if (children.empty()) return;
        children_iterator = children.begin();
        while (children_iterator != children.end()) {
                if ((*children_iterator)->getState() > 0)
                {
                        cout << endl << "The object " << (*children_iterator)-
>getName() << " is ready";
                        (*children_iterator)->printState();
                }
                else
                {
                        cout << endl << "The object " << (*children_iterator)-
>getName() << " is not ready";
                        (*children_iterator)->printState();
                }
                children_iterator++;
        }
}

Base* Base::find(string parentName)
{
        Base* search = nullptr;
        if (this->name == parentName)
        {
                search = this;
                return search;
        }
        else
        {
                if (!children.empty()) {
                        children_iterator = children.begin();
                        while (children_iterator != children.end())
                        {
                                if (search != 0) return search;
                                search = (*children_iterator)-
>find(parentName);
                                ++children_iterator;
                        }--children_iterator;
```

```
                }
                return(search);
        }
}

Base* Base::findPath(std::string path)
{
        Base* parentName = nullptr;
        if (path[0] == '/' && path[1] == '/'){
                path.erase(0, 2);
                return find(path);
        }
        else
        {
                int cnt = 2;
                std::string name;
                if (path.size() == 0)
                        return this;
                name = path[1];
                while (path[cnt] != '/'&& cnt < path.size())
                {
                        name += path[cnt];
                        cnt++;
                }
                path.erase(0, cnt);
                if (this->getName() == name)
                        parentName = this;
                children_iterator = children.begin();
                while (children_iterator != children.end())
                {
                        if ((*children_iterator)->getName() == name) {
                                parentName = *children_iterator;
                                break;
                        }
                        ++children_iterator;
                }
                if (parentName == nullptr)
                        return nullptr;
                parentName = parentName->findPath(path);
        }
        return parentName;
}

void Base::searchObject()
{
        std::string path;
        cin >> path;
        while (path != "//")
        {
                Base* temp;
                temp = findPath(path);
                if (temp != nullptr) {
                        cout << endl;
                        cout << path << " Object name: " << temp->getName();
                }
                else
```

```
                        cout << endl << path << " Object not found";
                cin >> path;
        }
}

void Base::printPath()
{
        Base* parentPtr = this->getParent();
        if (parentPtr != 0) {
                parentPtr->printPath();
        }
        cout << "/" << this->getName();
}

void Base::set_connection(TYPE_SIGNAL signal, Base* object, TYPE_HANDLER
handler)
{
        if (connects.size() == 0)
        {
                connection* link = new connection(signal, object,handler);
                connects.push_back(link);
                return;
        }
        for (int i = 0; i < connects.size(); i++) {
                if (signal == connects[i]->p_signal && object == connects[i] -
> p_base && handler == connects[i]->p_handler)
                {
                        return;
                }
        }
        connection* link = new connection(signal, object, handler);
        connects.push_back(link);
}

void Base::delete_connection(TYPE_SIGNAL signal, Base* object, TYPE_HANDLER
handler)
{
        for (int i = 0; i < connects.size(); i++)
        {
                if (signal == connects[i]->p_signal && object == connects[i] -
> p_base && handler == connects[i]->p_handler)
                {
                        connects.erase(connects.begin() + i);
                        return;
                }
        }
}

void Base::emit_signal(TYPE_SIGNAL signal, string& command)
{
        TYPE_HANDLER p_handler;
        (this->*signal)(command);
        for (int i = 0; i < connects.size(); i++){
                if (signal == connects[i]->p_signal)
                {
                        p_handler = connects[i]->p_handler;
                        (connects[i]->p_base->*p_handler)(command);
                }
        }
}
```

```
int Base::getClass()
{
	return this->classNum;
}
```

Файл cl_base.h

```
#ifndef CL_BASE_H
#define CL_BASE_H

#include <vector>
#include <string>

using namespace std;

#define SIGNAL_D(signal_f) (TYPE_SIGNAL)(&signal_f)
#define HANDLER_D(handler_f) (TYPE_HANDLER)(&handler_f)

class Base;
typedef void (Base::*TYPE_SIGNAL)(string&);
typedef void (Base::*TYPE_HANDLER)(string);

class Base {
	string name;
	int state,classNum;
	Base* parent;
	vector <Base*> children;
	vector <Base*>::iterator children_iterator;
	int level;
	struct connection
	{
		TYPE_SIGNAL p_signal;
		Base* p_base;
		TYPE_HANDLER p_handler;
		connection(TYPE_SIGNAL p_signal, Base* p_base, TYPE_HANDLER
p_handler)
		{
			this->p_signal = p_signal;
			this->p_base = p_base;
			this->p_handler = p_handler;
		}
	};
	vector <connection*> connects;

public:
	Base(Base*, int,int, string = "base");
	void setName(string);
	string getName();
	void setParent(Base*);
```

```
        Base* getParent();
        void printNames(int);
        Base* find(string);
        void setState(int);
        void setClass(int);
        int getState();
        void printState();
        Base* findPath(string);
        void searchObject();
        void printPath();
        void set_connection(TYPE_SIGNAL, Base*, TYPE_HANDLER);
        void delete_connection(TYPE_SIGNAL, Base*, TYPE_HANDLER);
        void emit_signal(TYPE_SIGNAL, string&);
        int getClass();

};

#endif
```

Файл main.cpp

```
#include "main.h"


int main() {
        Application app;
        app.buildTree();
        return app.execute();
}
```

Файл main.h

```
#ifndef MAIN_H
#define MAIN_H
#include "cl_application.h"

int main();

#endif
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
root root obj1 2 1 root obj6 2 1 root obj7 2 1 obj1 obj2 3 1 obj1 obj5 3 1 obj6 obj8 3 1 endtree 1 obj1 obj2 2 obj1 obj6 3 obj8 obj2 0 obj1 test1 obj8 test2 endsignals	Object tree root obj1 obj2 obj5 obj6 obj8 obj7 Set connects 1 obj1 obj2 2 obj1 obj6 3 obj8 obj2 Emit signals Signal to obj2 Text: obj1 -> test1 Signal to obj6 Text: obj1 -> test1 Signal to obj2 Text: obj8 -> test2	Object tree root obj1 obj2 obj5 obj6 obj8 obj7 Set connects 1 obj1 obj2 2 obj1 obj6 3 obj8 obj2 Emit signals Signal to obj2 Text: obj1 -> test1 Signal to obj6 Text: obj1 -> test1 Signal to obj2 Text: obj8 -> test2