**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.

**Входные данные**

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

**Выходные данные**

**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.

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.

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;  cl\_5\* ob\_5;  cl\_6\* ob\_6;  string nameParent, nameChild;  int selectFamily;  int state; | 2 |
| 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);  ob\_3->set\_state(state); | 2 |
| !(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);  scanSignals(this); | Ø |

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;  cl\_2\* messageStart;  messageStart = new cl\_2((cl\_base\*)ob\_parent, false, true);  messageStart->signaling(message, sender); | 1 |

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 |
| 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() | returnMessage(this, 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::returnMessage(cl\_base\* ob\_parent, string sender, string receiver)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действие | № перехода |
| 1 | ob\_parent->storageMess.size() == 0 | return; | Ø |
|  | !(ob\_parent->storageMess.size() == 0) |  | 2 |
| 2 |  | ob\_parent->it\_sM = ob\_parent->storageMess.begin(); | 3 |
| 3 | ob\_parent->it\_sM != ob\_parent->storageMess.end() |  | 4 |
| !(ob\_parent->it\_sM != ob\_parent->storageMess.end()) |  | Ø |
| 4 | sender == getNameSender((\*(ob\_parent->it\_sM))) | cout << endl << "Signal to " << receiver << " Text: " << sender << " -> " << getMessageText((\*(ob\_parent->it\_sM))); | 5 |
| !(sender == getNameSender((\*(ob\_parent->it\_sM)))) |  | 5 |
| 5 |  | ob\_parent->it\_sM++; | 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->children.size(); i++) |  | 14 |
| 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->children.at(i)) > 0 | ob\_5->set\_state(state); | 21 |
| !(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);  ob\_parent->it\_child++; | 29 |
| !(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 );  ob\_parent->it\_child++; | 4 |
|  | !(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 |