**Задача**

**Проверка готовности объектов к работе**

Фрагмент методического указания.

 Создание объектов и построение исходного иерархического дерева объектов.

 Система собирается из объектов, принадлежащих определенным классам. В тексте постановки задачи классу соответствует уникальный номер. Относительно номера класса определяются требования (свойства, функциональность).

Первоначальная сборка системы (дерева иерархии объектов, программы) осуществляется исходя из входных данных. Данные вводятся построчно.

Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Корневой объект объявляется в основной программе (main). Исходное состояние корневого объекта соответствует его функционированию.

Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии. Структура данных в строке:

«Наименование головного объекта» «Наименование очередного объекта» «Номер класса принадлежности очередного объекта» «Номер исходного состояния очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно «endtree» (в данной строке ввода больше ничего не указывается).

Готовность объекта характеризуется значением его состояния. Значение состояния - целое число. Определены правила для значения состояния:

0 – объект выключен;

Отрицательное – объект включен, но не функционирует, обнаружена неисправность. Значение классифицирует характер неисправности.

Положительное – объект включен, функционирует в штатном режиме. Значение определяет текущее состояние объекта.

Подчиненные объекты располагаются слева на право относительно голоного, согласно их следованию в исходных данных. Исходные данные подготовлены таким образом, что любой головной объект предварительно добавлен в качестве подчиненного. Подразумевается, что все объекты имеют уникальные имена.

Для организации исходя из входных данных создания экземпляров объектов и формирования иерархического дерева, необходимо:

1.      В базовом классе реализовать метод поиска объекта на дереве объектов по его наименованию и возврата указателя на него. Если объект не найден, то вернуть нулевой указатель.

2.      В корневом объекте (объект приложения) реализовать метод чтения исходных данных, создания объектов и построения исходного дерева иерархии.

Пример

Ввод

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  
endtree

Построенное дерево

app\_root  
    object\_1  
        object\_7  
    object\_2  
        object\_4  
        object\_5  
        object\_6  
    object\_3

Вывод списка готовности объектов

The object app\_root is ready  
The object object\_1 is ready  
The object object\_7 is ready  
The object object\_2 is ready  
The object object\_4 is not ready  
The object object\_5 is ready  
The object object\_6 is ready  
The object object\_3 is ready

**Постановка задачи**

Все сложные электронные, технические средства разного назначения в момент включения выполняют опрос готовности к работе составных элементов, индицируя соответствующую информацию на табло, панели или иным образом.

Построить модель иерархической системы. Реализовать задачу опроса готовности каждого объекта из ее состава и вывести соответствующее сообщение на консоль.

Объект считается готовым к работе:

1.      Создан и размешен в составе системы (на дереве иерархии объектов) согласно схеме архитектуры;

2.      Имеет свое уникальное наименование;

3.      Свойство, определяющее его готовность к работе, имеет целочисленное положительное значение.

В результате решения задачи опроса готовности объектов, относительно каждого объекта системы на консоль надо вывести соответствующую информацию:

Если свойство определяющее готовность объекта имеет положительное значение:

The object «наименование объекта» is ready

иначе

The object «наименование объекта» is not ready

 Система содержит объекты трех классов, не считая корневого. Номера классов: 2,3,4.

**Входные данные**

Множество объектов, их характеристики и расположение на дереве иерархии. Структура данных для ввода согласно изложенному в фрагменте методического указания.

**Выходные данные**

В первой строке вывести Test result Далее, построчно, согласно следованию объектов на дереве иерархии слева на право и сверху вниз, относительно каждого объекта в зависимости от состояния готовности выводиться, если объект готов к работе: The object наименование объекта is ready Если не готов, то The object наименование объекта is not ready

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

Bai lam

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\_application(string name)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | set\_object\_name(name); | 2 |
| 2 |  | set\_state(1); | Ø |

void cl\_application::bild\_tree\_objects()

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | cl\_2\* ob\_2; | 2 |
| 2 |  | cl\_3\* ob\_3; | 3 |
| 3 |  | cl\_4\* ob\_4; | 4 |
| 4 |  | string nameParent, nameChild; | 5 |
| 5 |  | int selectFamily, state; | 6 |
| 6 | while (true) |  | 7 |
| 7 |  | cin >> nameParent; | 8 |
| 8 | if (nameParent == text\_finish) | break; | Ø |
| else |  | 9 |
| 9 |  | cin >> nameChild >> selectFamily >> state; | 10 |
| 10 | if (selectFamily == 2) |  | 11 |
| else |  | 16 |
| 11 | if (this->get\_object\_name() == nameParent) |  | 12 |
| else |  | 15 |
| 12 |  | ob\_2 = new cl\_2((cl\_base\*)this); | 13 |
| 13 |  | ob\_2->set\_object\_name(nameChild); | 14 |
| 14 |  | ob\_2->set\_state(state); | 6 |
| 15 |  | addNewChild(this, nameParent, nameChild, state, selectFamily); | 6 |
| 16 | if (selectFamily == 3) |  | 17 |
| else |  | 22 |
| 17 | if (this->get\_object\_name() == nameParent) |  | 18 |
| else |  | 21 |
| 18 |  | ob\_3 = new cl\_3((cl\_base\*)this); | 19 |
| 19 |  | ob\_3->set\_object\_name(nameChild); | 20 |
| 20 |  | ob\_3->set\_state(state); | 6 |
| 21 |  | addNewChild(this, nameParent, nameChild, state, selectFamily); | 6 |
| 22 | if (selectFamily == 4) |  | 23 |
| else |  | 28 |
| 23 | if (this->get\_object\_name() == nameParent) |  | 24 |
| else |  | 27 |
| 24 |  | ob\_4 = new cl\_4((cl\_base\*)this); | 25 |
| 25 |  | ob\_4->set\_object\_name(nameChild); | 26 |
| 26 |  | ob\_4->set\_state(state); | 6 |
| 27 |  | addNewChild(this, nameParent, nameChild, state, selectFamily); | 6 |
| 28 |  | break; | Ø |

void cl\_application::addNewChild(cl\_base\* ob\_parent, string nameParent, string nameChild, int state, int selectFamily)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | cl\_2\* ob\_2; cl\_3\* ob\_3; cl\_4\* ob\_4; | 2 |
| 2 | if (selectFamily == 2) |  | 3 |
| else |  | 8 |
| 3 | for (size\_t i = 0; i < ob\_parent->children.size(); i++) |  | 4 |
| i = ob\_parent->children.size() |  | 20 |
| 4 | if (get\_object\_name((cl\_base\*)ob\_parent->children.at(i)) == nameParent) | ob\_2 = new cl\_2((cl\_base\*)ob\_parent->children.at(i)); | 5 |
| else |  | 3 |
| 5 |  | ob\_2->set\_object\_name(nameChild); | 6 |
| 6 | if (get\_state((cl\_base\*)ob\_parent->children.at(i)) >0) | ob\_2->set\_state(state); | 7 |
| else | ob\_2->set\_state(0); | 7 |
| 7 |  | return; | Ø |
| 8 | if (selectFamily == 3) |  | 9 |
| else |  | 14 |
| 9 | for (size\_t i = 0; i < ob\_parent->children.size(); i++) |  | 10 |
| i = ob\_parent->children.size() |  | 20 |
| 10 | if (get\_object\_name((cl\_base\*)ob\_parent->children.at(i)) == nameParent) | ob\_3 = new cl\_3((cl\_base\*)ob\_parent->children.at(i)); | 11 |
| else |  | 9 |
| 11 |  | ob\_3->set\_object\_name(nameChild); | 12 |
| 12 | if (get\_state((cl\_base\*)ob\_parent->children.at(i)) >0) | ob\_3->set\_state(state); | 13 |
| else | ob\_3->set\_state(0); | 13 |
| 13 |  | return; | Ø |
| 14 | if (selectFamily == 4) |  | 15 |
| else |  | 20 |
| 15 | for (size\_t i = 0; i < ob\_parent->children.size(); i++) |  | 16 |
| i = ob\_parent->children.size() |  | 20 |
| 16 | if (get\_object\_name((cl\_base\*)ob\_parent->children.at(i)) == nameParent) | ob\_4 = new cl\_4((cl\_base\*)ob\_parent->children.at(i)); | 17 |
| else |  | 15 |
| 17 |  | ob\_4->set\_object\_name(nameChild); | 18 |
| 18 | if (get\_state((cl\_base\*)ob\_parent->children.at(i)) >0) | ob\_4->set\_state(state); | 19 |
| else | ob\_4->set\_state(0); | 19 |
| 19 |  | return; | Ø |
| 20 |  | ob\_parent->it\_child = ob\_parent->children.begin(); | 21 |
| 21 | while (ob\_parent->it\_child != ob\_parent->children.end()) | addNewChild((\*(ob\_parent->it\_child)), nameParent, nameChild, state, selectFamily); | 22 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |
| 22 |  | ob\_parent->it\_child++; | 21 |

int cl\_application::exec\_app()

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | show\_object\_state(); | 2 |
| 2 |  | return 0; | Ø |

void cl\_application::show\_object\_state()

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | show\_state\_next(this); | Ø |

void cl\_application::show\_state\_next(cl\_base\* ob\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | if (get\_state(ob\_parent) > 0) | cout << endl << "The object " << ob\_parent->get\_object\_name() << " is ready”; | 2 |
| else | cout << endl << "The object " << ob\_parent->get\_object\_name() << " is not ready"; | 2 |
| 2 | if (ob\_parent->children.size() == 0) | return; | Ø |
| else |  | 3 |
| 3 |  | ob\_parent->it\_child = ob\_parent->children.begin(); | 4 |
| 4 | while (ob\_parent->it\_child != ob\_parent->children.end()) | show\_state\_next((\*(ob\_parent->it\_child))); | 5 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |
| 5 |  | ob\_parent->it\_child++; | 4 |

cl\_base(cl\_base\* p\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | set\_object\_name( " cl\_base "); | 2 |
| 2 | if(p\_parent) | this->p\_parent = p\_parent; | 3 |
| else |  | 4 |
| 3 |  | p\_parent->add\_child(this); | Ø |
| 4 |  | this->p\_parent = 0; | Ø |

void cl\_base::set\_object\_name(string object\_name)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | this->object\_name = object\_name; | Ø |

string cl\_base::get\_object\_name(cl\_base\* p\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | return p\_parent->object\_name; | Ø |

void cl\_base::set\_parent(cl\_base\* p\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | if (p\_parent) | this ->p\_parent = p\_parent; | 2 |
| else |  | Ø |
| 2 |  | p\_parent->add\_child(this); | Ø |

void cl\_base::add\_child(cl\_base\* p\_child)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | children.push\_back(p\_child); | Ø |

cl\_base\* cl\_base::get\_child(string object\_name)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № 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() |  | 6 |
| 4 | if (get\_object\_name((\*it\_child)) == object\_name) | return (\*it\_child); | 5 |
| else |  | 5 |
| 5 |  | it\_child++; | 4 |
| 6 |  | return 0; | Ø |

void cl\_base::set\_state(int c\_state)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | this->c\_state = c\_state; | Ø |

int cl\_base::get\_state(cl\_base\* p\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | return p\_parent->c\_state; | Ø |

int main()

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | string name; | 2 |
| 2 |  | cin >> name; | 3 |
| 3 |  | cl\_application ob\_application(name); | 4 |
| 4 |  | ob\_application.bild\_tree\_objects(); | 5 |
| 5 |  | cout << "Test result"; | 6 |
| 6 |  | return ob\_application.exec\_app(); | Ø |