**Определение указателя на объект по его координате**

 Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

В составе базового класса реализовать метод получения указателя на любой объект в составе дерева иерархии объектов.  
В качестве параметра методу передать путь объекта от корневого. Путь задать в следующем виде:

/root/ob\_1/ob\_2/ob\_3

Уникальность наименования требуется только относительно множества подчиненных объектов для любого головного объекта.

Если система содержит объекты с уникальными именами, то в методе реализовать определение указателя на объект посредством задания координаты в виде:

//«наименование объекта»

Состав и иерархия объектов строиться посредством ввода исходных данных.  Ввод организован как в контрольной работе № 1.  
Единственное различие. В строке ввода первым указать не наименование головного объекта, а путь к головному объекту.  
Подразумевается, что к моменту ввода очередной строки соответствующая ветка на дереве иерархии уже построена.  
Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

**Пример ввода иерархии дерева объектов.**

root  
/root object\_1 3 1  
/root object\_2 2 1  
/root/object\_2 object\_4 3 -1  
/root/object\_2 object\_5 4 1  
/root object\_3 3 1  
/root/object\_2 object\_3 6 1  
/root/object\_1 object\_7 5 1  
/root/object\_2/object\_4 object\_7 3 -1  
endtree

## 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 координаты искомых объектов.  
Ввод завершается при вводе: //

## Выходные данные

**Первая строка:**  
Object tree  
  
**Со второй строки** вывести иерархию построенного дерева.

**Далее, построчно**:  
«координата объекта» Object name: «наименование объекта»  
Разделитель один пробель.

Если объект не найден, то вывести:  
«координата объекта» Object not found  
Разделитель один пробель.

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 | while(true) | char check, checkNext; cin >> check >> checkNext; | 2 |
| false |  | Ø |
| 2 | if (checkNext == '/') | break; | Ø |
| else | scanElementsX(checkNext, this); | 1 |

void cl\_application::scanElementsX(char nextChar, cl\_base\* ob\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | string ancestor; ancestor.push\_back(nextChar); char charNext; charNext = getchar(); | 2 |
| 2 | while(charNext != '/') |  | 3 |
| charNext == '/' |  | 4 |
| 3 | if (charNext != ' ') | ancestor.push\_back(charNext); charNext = getchar(); | 2 |
| else | break | 4 |
| 4 | if (charNext == ' ') | addNewChild(this); | 5 |
| else |  | 5 |
| 5 | if (charNext == '/') | doWithChildLink(this); | Ø |
| else |  | Ø |

void cl\_application::doWithChildLink(cl\_base\* ob\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | string ancestor; char charNext; charNext = getchar(); | 2 |
| 2 | while (charNext != '/') |  | 3 |
| charNext == '/' |  | 4 |
| 3 | if (charNext != ' ') | ancestor.push\_back(charNext); charNext = getchar(); | 2 |
| else | break | 4 |
| 4 |  | ob\_parent->it\_child = ob\_parent->children.begin(); | 5 |
| 5 | while (ob\_parent->it\_child != ob\_parent->children.end()) |  | 6 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |
| 6 | if (get\_object\_name((\*(ob\_parent->it\_child))) == ancestor && charNext == '/') | doWithChildLink((\*(ob\_parent->it\_child))); break; | Ø |
| else |  | 7 |
| 7 | if (get\_object\_name((\*(ob\_parent->it\_child))) == ancestor && charNext == ' ') | addNewChild((\*(ob\_parent->it\_child))); break; | Ø |
| else | ob\_parent->it\_child++; | 5 |

void cl\_application::addNewChild(cl\_base\* ob\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | cl\_2\* ob\_2; cl\_3\* ob\_3; cl\_4\* ob\_4; cl\_5\* ob\_5; cl\_6\* ob\_6; int selectFamily; int state; string nameObject; cin >> nameObject >> selectFamily >> state; | 2 |
| 2 | if (selectFamily == 2) | ob\_2 = new cl\_2((cl\_base\*)ob\_parent); ob\_2->set\_object\_name(nameObject); ob\_2->set\_state(state); | Ø |
| else if (selectFamily == 3) | ob\_3 = new cl\_3((cl\_base\*)ob\_parent); ob\_3->set\_object\_name(nameObject); ob\_3->set\_state(state); | Ø |
| else if (selectFamily == 4) | ob\_4 = new cl\_4((cl\_base\*)ob\_parent); ob\_4->set\_object\_name(nameObject); ob\_4->set\_state(state); | Ø |
| else if (selectFamily == 5) | ob\_5 = new cl\_5((cl\_base\*)ob\_parent); ob\_5->set\_object\_name(nameObject); ob\_5->set\_state(state); | Ø |
| else if (selectFamily == 6) | ob\_6 = new cl\_6((cl\_base\*)ob\_parent); ob\_6->set\_object\_name(nameObject); ob\_6->set\_state(state); } | Ø |
| else | return; | Ø |

int cl\_application::exec\_app()

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | show\_state\_next(this, 0); | Ø |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | if (i == 0) | cout << endl << get\_object\_name(ob\_parent); | 2 |
| else | cout << endl << setw(4 \* i) << " " << get\_object\_name(ob\_parent); | 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)), i + 1); ob\_parent->it\_child++; | 4 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |

void cl\_application::extension(cl\_base\* ob\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | if (ob\_parent->children.size() == 0) | cout << endl << get\_object\_name(ob\_parent) << " " << "Object not found"; return; | Ø |
| else |  | 2 |
| 2 |  | ob\_parent->it\_child = ob\_parent->children.begin(); | 3 |
| 3 | while (ob\_parent->it\_child != ob\_parent->children.end()) | cout << endl << get\_object\_name(ob\_parent) << " Object name: " << get\_object\_name((\*(ob\_parent->it\_child))); extension((\*(ob\_parent->it\_child))); ob\_parent->it\_child++; | 3 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |

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; p\_parent->add\_child(this); | Ø |
| else | 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; p\_parent->add\_child(this); | Ø |
| else |  | Ø |

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

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

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()) | return (\*it\_child); | Ø |
| it\_child == children.end() |  | 4 |
| 4 |  | 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; cin >> name;  cl\_application ob\_application(name);  ob\_application.bild\_tree\_objects();  cout << "Object tree";  return ob\_application.exec\_app(); | Ø |