**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.

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.

New function:

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | while(true) | char check, checkNext; cin >> check >> checkNext; | 2 |
| false |  | 7 |
| 2 | if (checkNext == '/') |  | 3 |
| else | scanElementsX(checkNext, this); | 1 |
| 3 |  | string text; text.push\_back(check); text.push\_back(checkNext); char charNext; charNext = getchar(); | 4 |
| 4 | while (charNext != '\n') |  | 5 |
| charNext == '\n') |  | 6 |
| 5 | if (charNext != ' ') | text.push\_back(charNext); charNext = getchar(); | 4 |
| else | break; | 6 |
| 6 | if (text == text\_finish) | break; | 7 |
| else; |  | 1 |
| 7 |  | processAccess(this); | Ø |

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 |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | show\_state\_next(this, 0); | 2 |
| 2 |  | resultLink(this); | Ø |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | if (ob\_parent->childrenLink.size() == 0) | return; | Ø |
| else |  | 2 |
| 2 |  | ob\_parent->it\_childLink = ob\_parent->childrenLink.begin(); | 3 |
| 3 | while (ob\_parent->it\_childLink != ob\_parent->childrenLink.end()) |  | 4 |
| ob\_parent->it\_childLink == ob\_parent->childrenLink.end() |  | Ø |
| 4 | if (getStateDo((\*(ob\_parent->it\_childLink)))) | cout << endl << getLinkName((\*(ob\_parent->it\_childLink))) << " Object name: " << getCheckName((\*(ob\_parent->it\_childLink))); | 5 |
| else | cout << endl << getLinkName((\*(ob\_parent->it\_childLink))) << " Object not found"; | 5 |
| 5 |  | ob\_parent->it\_childLink++; | 3 |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | while (true) | string linkName; cin >> linkName; | 2 |
| false |  | Ø |
| 2 | if (linkName == text\_finish2) | break; | Ø |
| else | connectWithRoot(0, linkName, ob\_parent, false); | 1 |

void cl\_application::connectWithRoot(int count, string linkName, cl\_base\* ob\_parent, bool notFound)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | cl\_2\* ob\_2L; ob\_2L = new cl\_2((cl\_base\*)ob\_parent, true); ob\_2L->setLinkName(linkName); ob\_2L->setStateDo(false); | 2 |
| 2 | if (linkName[count] == '/') |  | 3 |
| else |  | Ø |
| 3 |  | count++; string name; | 4 |
| 4 | while (linkName[count] != '/') | name.push\_back(linkName[count]); | 5 |
| linkName[count] == '/' |  | 6 |
| 5 | if (count == (linkName.size() - 1)) | break; | 6 |
| else | count++; | 4 |
| 6 | if (linkName[count] == '/') |  | 7 |
| else |  | 17 |
| 7 | if (name.size() == 0) |  | 8 |
| else |  | 13 |
| 8 | if (checkMember(ob\_parent)) | count++; string name; | 9 |
| else |  | Ø |
| 9 | while (linkName[count] != '/') | name.push\_back(linkName[count]); | 10 |
| linkName[count] == '/' |  | 11 |
| 10 | if (count == (linkName.size() - 1)) | break; | 11 |
| else | count++; | 9 |
| 11 | if (checkMemberDeep(this, name)) |  | 12 |
| else |  | Ø |
| 12 | if (getStateDo((cl\_2\*)ob\_2L) == false) | ob\_2L->setCheckName(name); ob\_2L->setStateDo(true); | Ø |
| else |  | Ø |
| 13 | if (name == get\_object\_name(ob\_parent)) |  | 14 |
| else |  | Ø |
| 14 | if (checkMember(ob\_parent)) | ob\_parent->it\_child = ob\_parent->children.begin(); | 15 |
| else |  | Ø |
| 15 | while (ob\_parent->it\_child != ob\_parent->children.end()) | connectAccess(count, linkName, ob\_2L, (\*(ob\_parent->it\_child))); | 16 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |
| 16 |  | ob\_parent->it\_child++; | 15 |
| 17 | if (name == get\_object\_name(ob\_parent)) |  | 18 |
| else |  | Ø |
| 18 | if (getStateDo((cl\_2\*)ob\_2L) == false) | ob\_2L->setCheckName(name); ob\_2L->setStateDo(true); | Ø |
| else |  | Ø |

void cl\_application::connectAccess(int count, string linkName, cl\_2\* ob\_2L, cl\_base\* ob\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | count++; string name; | 2 |
| 2 | while (linkName[count] != '/') | name.push\_back(linkName[count]); | 3 |
| linkName[count] == '/' |  | 4 |
| 3 | if (count == (linkName.size() - 1)) | break; | 4 |
| else | count++; | 2 |
| 4 | if (linkName[count] == '/') |  | 5 |
| else |  | 15 |
| 5 | if (name.size() == 0) |  | 6 |
| else |  | 11 |
| 6 | if (checkMember(ob\_parent)) | count++; string name; | 7 |
| else |  | Ø |
| 7 | while (linkName[count] != '/') | name.push\_back(linkName[count]); | 8 |
| linkName[count] == '/' |  | 9 |
| 8 | if (count == (linkName.size() - 1)) | break; | 9 |
| else | count++; | 7 |
| 9 | if (checkMemberDeep(this, name)) |  | 10 |
| else |  | Ø |
| 10 | if (getStateDo((cl\_2\*)ob\_2L) == false) | ob\_2L->setCheckName(name); ob\_2L->setStateDo(true); | Ø |
| else |  | Ø |
| 11 | if (name == get\_object\_name(ob\_parent)) |  | 12 |
| else |  | Ø |
| 12 | if (checkMember(ob\_parent)) | ob\_parent->it\_child = ob\_parent->children.begin(); | 13 |
| else |  | Ø |
| 13 | while (ob\_parent->it\_child != ob\_parent->children.end()) | connectAccess(count, linkName, ob\_2L, (\*(ob\_parent->it\_child))); | 14 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |
| 14 |  | ob\_parent->it\_child++; | 13 |
| 15 | if (name == get\_object\_name(ob\_parent)) |  | 16 |
| else |  | Ø |
| 16 | if (getStateDo((cl\_2\*)ob\_2L) == false) | ob\_2L->setCheckName(name); ob\_2L->setStateDo(true); | Ø |
| else |  | Ø |

 bool cl\_application::checkMemberDeep(cl\_base\* ob\_parent, string name)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | bool result = false;  ob\_parent->it\_child = ob\_parent->children.begin(); | 2 |
| 2 | while (ob\_parent->it\_child != ob\_parent->children.end()) |  | 3 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | 6 |
| 3 | if (name == get\_object\_name(\*(ob\_parent->it\_child))) | result = true; | 4 |
| else |  | 4 |
| 4 | if (checkMemberDeep((\*(ob\_parent->it\_child)), name)) | result = true; | 5 |
| else |  | 5 |
| 5 |  | ob\_parent->it\_child++; | 2 |
| 6 |  | return result; | Ø |

bool cl\_application::checkMember(cl\_base\* ob\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | if (ob\_parent->children.size() == 0) | return false; | Ø |
| else | return true; | Ø |

   
   
void cl\_base::setLinkName(string linkName)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | this->linkName = linkName; | Ø |

string cl\_base::getLinkName(cl\_base\* p\_parent)

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

void cl\_base::setCheckName(string checkName)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | this->checkName = checkName; | Ø |

string cl\_base::getCheckName(cl\_base\* p\_parent)

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

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

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

void cl\_base::setStateDo(bool stateDo)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | this->stateDo = stateDo; | Ø |

bool cl\_base::getStateDo(cl\_base\* p\_parent)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | return p\_parent->stateDo; | 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 |