**Задача**

Для организации иерархического построения объектов необходимо разработать базовый класс, который содержит функционал и свойства для построения иерархии объектов. В последующем, в приложениях использовать этот класс как базовый для всех создаваемых классов. Это позволит включать любой объект в состав дерева иерархии объектов.

Создать базовый класс со следующими элементами:

Свойства:

-       наименование объекта (строкового типа);

-       указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно 0);

-       список указателей на объекты подчиненных к текущему объекту в дереве иерархии.

Функционал:

-       параметризированный конструктор с параметром указателя на головной объект в дереве иерархии;

-       параметризированный конструктор с параметром указателя на головной объект в дереве иерархии и наименованием объекта;

-       метод задания имени объекта;

-       метод получения имени объекта;

-       метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз.

Написать программу, которая последовательно строит дерево иерархии объектов, слева направо и сверху вниз. Переход на новый уровень происходит от правого (последнего) объекта предыдущего уровня. Для построения дерева использовать объекты двух производных классов, наследуемых от базового. Каждый объект имеет уникальное имя.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

**Входные данные**

**Первая строка:**  
«имя корневого объекта»  
Создается корневой объект.  
**Вторая строка и последующие строки:**  
«имя головного объекта» «имя подчиненного объекта»  
Создается подчиненный объект и добавляется в иерархическое дерево.

Если «имя головного объекта» равняется «имени подчиненного объекта», то новый объект не создается и построение дерева объектов завершается.

**Пример ввода**

Object\_root  
Object\_root Object\_1  
Object\_root Object\_2  
Object\_root Object\_3  
Object\_3 Object\_4  
Object\_3 Object\_5  
Object\_6 Object\_6

Дерево объектов, которое будет построено по данному примеру:

Object\_root  
         Object\_1  
         Object\_2  
         Object\_3  
                   Object\_4  
                   Object\_5

**Выходные данные**

**Первая строка:**   
«имя корневого объекта»  
**Вторая строка и последующие строки** имена головного и подчиненных объектов очередного уровня разделенных двумя пробелами.   
«имя головного объекта» «имя подчиненного объекта»[[ «имя подчиненного объекта»] …….]  
  
  
**Пример вывода**   
  
Object\_root  
Object\_root  Object\_1  Object\_2  Object\_3  
Object\_3  Object\_4  Object\_5

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); | Ø |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | cl\_2\* ob\_2; | 2 |
| 2 |  | string nameParent, nameChild; | 3 |
| 3 | while (true) |  | 4 |
| 4 |  | cin >> nameParent, nameChild; | 5 |
| 5 | if (nameParent != nameChild) |  | 6 |
| else | break; | Ø |
| 6 | if (get\_object\_name(this) == nameParent) | ob\_2 = new cl\_2((cl\_base\*)this); | 7 |
| else | add\_new\_child(this, nameParent, nameChild); | 3 |
| 7 |  | ob\_2->set\_object\_name(nameChild); | 3 |

void cl\_application::add\_new\_child(cl\_base\* ob\_parent, string nameParent, string nameChild)

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | cl\_2\* ob\_2; | 2 |
| 2 | for (size\_t i = 0; i < ob\_parent->children.size(); i++) |  | 3 |
| i = ob\_parent->children.size() |  | 6 |
| 3 | if (get\_object\_name((cl\_base\*)ob\_parent->children.at(i)) == nameParent) | ob\_2 = new cl\_2((cl\_base\*)ob\_parent->children.at(i)); | 4 |
| else |  | 2 |
| 4 |  | ob\_2->set\_object\_name(nameChild); | 5 |
| 5 |  | return; | Ø |
| 6 |  | ob\_parent->it\_child = ob\_parent->children.begin(); | 7 |
| 7 | while (ob\_parent->it\_child != ob\_parent->children.end()) | add\_new\_child((\*(ob\_parent->it\_child)), nameParent, nameChild, state, selectFamily); | 8 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |
| 8 |  | ob\_parent->it\_child++; | 7 |

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 |  | cout << get\_object\_name(this); | 2 |
| 2 |  | show\_state\_next(this); | Ø |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | cout << " " << get\_object\_name(ob\_parent); | Ø |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 | if (ob\_parent->children.size() == 0) | return; | Ø |
| else |  | 2 |
| 2 |  | cout << endl; | 3 |
| 3 |  | cout << get\_object\_name(ob\_parent); | 4 |
| 4 |  | ob\_parent->it\_child = ob\_parent->children.begin(); | 5 |
| 5 | while (ob\_parent->it\_child != ob\_parent->children.end()) | show\_child((\*(ob\_parent->it\_child))); | 6 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | 7 |
| 6 |  | ob\_parent->it\_child++; | 5 |
| 7 |  | ob\_parent->it\_child = ob\_parent->children.begin(); | 8 |
| 8 | while (ob\_parent->it\_child != ob\_parent->children.end()) | show\_state\_next((\*(ob\_parent->it\_child))); | 9 |
| ob\_parent->it\_child == ob\_parent->children.end() |  | Ø |
| 9 |  | ob\_parent->it\_child++; | 8 |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | set\_object\_name ( "cl\_base" ); | 2 |
| 2 | if(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 |

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 |  | return ob\_application.exec\_app(); | Ø |