**Задача**

**Вывод иерархического дерева объектов на консоль**

 Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных программах динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

Построить модель иерархической системы. Реализовать вывод на консоль иерархического дерева объектов в следующем виде:

root  
    ob\_1  
        ob\_2  
    ob\_3  
        ob\_4  
            ob\_5  
        ob\_6  
            ob\_7

где: root - наименование корневого объекта (приложения).

Состав и иерархия объектов строиться посредством ввода исходных данных.  Ввод организован как в контрольной работе № 1.

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

**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.

**Выходные данные**

Вывести иерархию объектов в следующем виде: Object tree Наименование корневого объекта Наименование объекта 1 Наименование объекта 2 Наименование объекта 3 . . . . . Отступ каждого уровня иерархии 4 позиции.

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, 2); | Ø |

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

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Предикат | Действе | № перехода |
| 1 |  | cout << endl << setw(i\*4) << right << 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 ); | 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 << "Object tree"; | 6 |
| 6 |  | return ob\_application.exec\_app(); | Ø |

 