Оператор копирования списка:

Node \*p = other.H, \*last = nullptr;

While (p!=nullptr)

{

Node \*q = nullptr;

q= new Node;

q->Data = p->Data;

q->next = nullptr;

if(p==other.H)

H=q;

else

Last->next = q;

last=q;

p = p->next;

}

T = last;

Static Node \*clone(const Node \*node)

{

Node \*result = nullptr;

If (node != nullptr)

{

result = new Node;

result->data = node->data;

result->next = clone(node->next);

}

return result;

}

List::List(const List &other) : H(nullptr), T(nullptr)

{

H = clone(other.H);

T = H;

if(T)

while(T->next);

T = T->next

}

Копирование дерево:

Static Node \*clone(const Node \*node)

{

Node \*result = nullptr;

If (node != nullptr)

{

result = new Node;

result->data = node->data;

result->left = clone(node->left);

result->right = clone(node->right);

}

return result;

}

Tree::Tree(const Tree &o)

{

root = clone(o.root);

}

Static Node \*clone(const Node \*node, const Node \*parent)

{

Node \*result = nullptr;

If (node != nullptr)

{

result = new Node;

result->data = node->data;

result->parent = parent;

result->left = clone(node->left, result);

result->right = clone(node->right, result);

}

return result;

}

Tree::Tree(const Tree &o)

{

root = clone(o.root, nullptr);

}

Связь между классами в C++:

-композиция

-наследование

-дружественность

|  |  |  |
| --- | --- | --- |
| **композиция** | **наследование** | **дружественность** |
| Композиция (pattern pimple) когда внутри одного класса используется другой класс (в классе машина используется класс колеса) НО!!! композиция к этому не сводится: может быть доведена до уровня, когда один объект реализуется посредством другого.  Pimpl мутит объект оболочку и объект реализации. Pimpl может содержать экземпляр на др. класс ссылку на др. класс указатель на др. класс | Наследование использует спец. синтаксис.  В С++ сужесвует 4 вида наследования:  -открытое  -защищенное  -закрытое  -виртуальное  В реальности нас интересует только 2 вида наследования открыто и закрытое, это фишка только С++, в других языках наследование - только публичное(открытое).  Далее под наследованием понием открытое наследование.  Наследование обязано реализовать связь «является»(to be)- принцип подстановки лисков (любой класс потомок может быть подставлен вместо класса предка без ущерба для программы).  Наследование одностороннее, как и композиция.  Наследование используется для двух целей:  - повторное использование кода (не злоупотреблять)  - динамический полиморфизм  Соотношение спецификаторов с уровнями доступа:  1. (public, protected, private) B<-(public) D(B public, B protected)<-D1(B public, B protected, D public, D protected) класс наруто видит все public  2. (public, protected, private) B<-(protected) D(B public, B protected)<-D1(B public, B protected, D public, D protected)  Наруто видит ничего  3. (public, protected, private) B<-(private) D(B public, B protected)<-D1() | Класс, объявленный другом какого-то класса, имеет полноценный доступ ко всем полям и методам этого класса.  Дружественность связь односторонняя. Хотя возможно сделать двустороннюю дружественность.  Сделать своим другом может только сам класс.  Кроме дружественных классов бывают дружественные функции.  Дружественная функция членом класса не является, и обычно реализуется с наружи этого класса.  Распространенный дружественный оператор - оператор вывода в поток. Оператор присваивания никогда не может быть дружественным. |
|  | Class Base  {  ----  };  Class Derived : public Base  {  }; | Class A  {  ----  };  Class B  {  Friend A;  };  Объявление класса для совместной дружественности:  сlass B; // - для объявления дружественности, для предварительного объявления указателя на заданный класс  сlass A  {  friend B;  };  class B  {  friend A;  };  friend;  ostream &operator<<(ostream &out const Zuzu &obj)  out <<obj.friend;  return out; |

Уровни доступа С++.

Сущесвует 3 уровня доступа:

-публичный(public) – доступно всем (не следует делать публичными данные), бывают публичными методы.

-защищенный(protected) – доступен наследникам и самому классу, наружу он уже не доступен.

-закрытый(private)

Все методы доступны исключительно классу.

Пример наследования с прямоугольником:

class Rectangle

{

double w, h; //если ничего не написано значит private

public:

Rectangle(double w, double h) : w(w), h(h){}

void set Width(double value)

{

w = value;

}

void setHeight(double value)

{

n = value;

}

double getSquare() const

{

return w\*h;

}

};

class Quadratum : public Rectangle

{

public:

Quadratum(double a) : Rectangle(a, a){};

void setSide (double value)

{

setWidth(value);

setHight(value);

}

}

Допиливаем:

class Quadratum : private Rectangle

{

public:

Quadratum(double a) : Rectangle(a, a){};

void setSide (double value)

{

setWidth(value);

setHight(value);

}

double getSquare()const

{

return Rectangle::getSquare();

}

} //лучше композицией

Через композицию:

class Quadratum

{

class Rectangle rect;

public:

Quadratum(double a):

rect(a,a){}

void setSide(double value);

{

rect.setWights(value);

rect.setHight(value);

}

double getSquare()const //pattern decorator

{

return rect.getSquare();

}

}

pattern – способ что-то запрограммировать.

**pattern pimpl**

h-ка:

class Stack

{

public:

Stack();

Stack(const Stack &o);

----//----

void push(Data data)

private:

class Implementation;

Implementation \*pimpl;

};

cpp-ка:

class Implementation

{

public:

Implementation();

Implementation(const Implementation &o);

----//----

void push(Data data);

private:

struct Node

{

----//----

} \*top;

static Node\* clone(Node \*node);

};

Stack():pimpl(nullptr)

{

pimpl = new Implementation();

}

Stack(const Stack &o):pimpl(nullptr)

{

pimpl = new Implementation (\*(o.pimpl));

}

void Stack::push(Data data)

{

pimpl->push(data);

}

ДЗ: Оператор присваиваня и деструктор и т.д. pimpl