**Объекты в куче, MVC.**

Выделение памяти под объекты в куче точно такое же, как и со встроенными типами int \*pInt = new int [10]; Complex \*pC; Complex com1(5,6); pc=&com1;

Complex \*arrayC = new Complex [100]; Особенность в том, что синтаксис new не предусматривает в таком случае после имени типа никаких конструкторов, поэтому обязательно у такого класса должен быть конструктор по умолчанию. Если нет конструктора по умолчанию (без параметров), то такой код не скомпилируется.

**Большие программы состоят из большого числа классов и объектов. Какие при этом могут возникать соображения по организации? Пусть есть два класса A и B**. A -> B. То есть объект класса A зависит от объекта класса B. Какие есть варианты?

1.Либо класс А имеет внутри себя ссылку на класс B. class A { B& b; };

2.Либо класс А имеет внутри себя поле указатель на класс B. class A { B\* b; };

3.Либо класс A имеет внутри себя объект класса B. class A { B b; };

В таких случаях говорят, что класс A зависит от класса B. Что означает зависит? Если меняется класс B, то придётся скорее всего изменять и класс A (например, изменяться названия методов или полей или их типы).

Если и A -> B и B -> A то классы называются сильно связанными – это плохая ситуация. При изменении любого из классов нужно изменять второй. Если при случае A -> B можно переносить отдельно класс B в другую программу и использовать полезно написанный код повторно. То в случае A < - > B нельзя отдельно переносить какой-то из этих классов и использовать повторно код в новых программах.

Желательно, чтобы программа представляла собой ацикличный граф. Классы – это вершины, если один класс ссылается на другой - между ними рисуется ребро. К сожалению, так получается не всегда. Но такие программы легче сопровождать, в них меньше ошибок.

Макет программы крестики – нолики.

1.Модель. Внутреннее представление, работа с моделью, сделать ход и т.п.

2.View. Интерфейс пользователя, отображение на экране, ввод данных, действия пользователя.

Кто от кого зависит? Model < - > View но так не очень хорошо. Какая из частей скорее всего поменяется когда-либо ? Очевидно View, потому что сначала это будет псевдографика, потом возможно GUI, потом работа в браузере и т.д. Но модель при этом не меняется. Что касается GUI для С++ нет стандарта для графической библиотеки, есть разные реализации, view меняется чаще. Если сделать Model -> View, то при изменении View с большой вероятностью надо переписывать Model. Например, в GTK функция пусть будет называться show, в QT пусть paint, при переходе с одного на другое придётся переписывать Model.

Для этого примера подойдёт Model <- View. Можно делать разные View, а модель останется одной и той же.

class Board

class BoardView