1. Lecture 8 – C++ Classes – Operator(s) (Pt.1)
   1. Today’s Topics
      1. C++ Classes Cheatsheet
         1. ⮚  Declaration
         2. ⮚  Members, Methods, Interface
         3. ⮚  Implementation–Resolution Operator(**:**☺
         4. ⮚  Instantiation–Objects
         5. ⮚  Object Usage – Dot Operator ( **.** )
         6. ⮚  Object Pointer Usage – Arrow Operator ( **->**)
         7. ⮚  Classes as Function Parameters, Pass-by-Value, by-(**const**)-Reference, by-Address
         8. ⮚  Protection Mechanisms–**const** Method signature
         9. ⮚  Classes – Code File Structure
         10. ⮚ Constructor(s), Initialization List(s), Destructor
         11. ⮚ **static** Members – Variables / Functions
      2. Operator(s)
      3. Operator Overloading
   2. Classes
      1. Delegating Constructor (C++ 11):
         1. ⮚ Can have one **ctor** invoke another **ctor**. ***Car*(char lP[PLT], int fId)** 
            1. ***Car*(lP, DFT\_GLNS,0, fId, DFT\_TIM) { /\* delegating ctor body ... \*/ }**
      2. Default Member Initialization (C++ 11):
         1. ⮚ Can set default Member values in Declaration.
         2. ⮚ Any *Initializer List* appearance of the member will hold precedence over this default.
      3. Static Data Members:
         1. ⮚ Class state properties, not bound to an Object.
         2. ⮚ Manipulated via the Class or an Object (if not **private**).
      4. Static Member Function
         1. ⮚ Can only manipulate & address **static** Data Members
      5. Static Local Variables in Class Methods:
         1. ⮚ Statically allocated data.
         2. ⮚ Initialized the first time Class Function block is entered
         3. ⮚ Lifetime until program exits!
   3. Operator(s)
      1. Operators in Classes – Introduction
      2. Operators (+, -, %, ==, etc.) and Built-in Types (int, double, etc.)
         1. In reality they represent Functions
            1. ⮚ Simply “called” with different syntax: **x 7;**
            2. ( **+** ) is binary operator with x and 7 as operands.  
               ⮚ It’s just a more intuitive notation for humans, instead of:

Add (x,7) or +(x,7)

* + 1. Operator(s) and Custom Types
       1. Useful to have an Operator work with user-defined types?
          1. ⮚Operator (**+**) :

classObjec3 = classObject1 + classObject2

* 1. Operator Overloading
     1. Overloading Operator(s)
        1. Overloading *Binary* Operator ( **==** ):
           1. ⮚ Non-Member Function of Class ***Money***.
           2. ⮚ Like overloading functions, Operator is Function name.
        2. Still like a regular Overloaded Function
           1. Non-member Functions of Class Money
           2. More “involved” than Member-by Member adding
        3. Overloading Operator (=)
           1. ⮚ Must be Member Operator.
           2. ⮚ If not specified, defaults to Member-Copy Assignment.
           3. ⮚ *Remember* **Deep***-Copy* vs **Shallow***-Copy*.
     2. Return by-const-Value
     3. Return by-const-Reference
     4. Return by-Reference
     5. Remember All Operators
        1. Overload just about anything, but be VERY careful...
           1. ⮚**[]**
           2. ⮚ **\*** : Multiplication, Pointer Dereference ⮚ **/** : Division
           3. ⮚ **+** : Addition, Unary Positive
           4. ⮚ **-** : Subtraction, Unary Negative
           5. ⮚ **++** : Increment, Pre-and-Post
           6. ⮚ **--** : Decrement, Pre-and-Post
           7. ⮚ **=** : Assignment
           8. ⮚ **<=**, **>=**, **<**, **>**, **==**, **!=** : Comparisons ⮚ Many, many others...
        2. Some are out, some should be kept untouched...
           1. ⮚ **?** : Ternary Conditional is not Overloadeable.
           2. ⮚ **&&**, **||**, built-in versions are defined for **bool** types. Use “Short-Circuit Evaluation”, also available in C++.
           3. ⮚ When overloaded no longer uses “Short-Circuit”, but “Complete Evaluation”. Generally should not overload these operators,  
              (also Operator Overloading had better “make sense”).