# Programming Assignment 2
# (Classes, Constructors, & Operator Overloading)
## Due Date: October 04, 2013

---

**Instructions:**

- This assignment is to be turned in via Blackboard before **11:59 PM** on **October 04, 2013**. You should submit '.cpp' files with names mentioned after each problem.

- Make sure your programs compile (without any compiler errors). You will not receive any credit if your program does not compile. If you are unable to complete any of the programs, submit the parts that work (with no compiler errors) for partial credit.

- Your grade will be based on functionality (does the program do what it is suppose to do), readability (is the code nicely formatted), and understandability (are the literals meaningful and is the code modular and well documented with appropriate comments). Try to incorporate all the good programming practices and styles, like **WRITE COMMENTS**, declare all constants and variables at the top of a function, store quantities using the appropriate data types, declare function prototypes, etc.

---

### 1. Student Class (30 points)

Create a class named Student that has three member variables:
- name - A string for the name of the student.
- numClasses - An integer for how many courses the student is currently enrolled in.
- classList - An array of strings for the names of the classes the student is enrolled in.

Write appropriate constructor(s), mutator (set), and accessor (get) methods for the class along with the following:
- A function that inputs all values from the user, including the list of class names.
- A function that outputs the name and list of all courses.
- A function that resets the number of classes to 0 and the classList to an empty array (with null strings).

The main() should test all of these functions. It should perform the following steps in a loop:
- It should ask the user to input a students data.
- Display the entered data back (in a well-formatted way).

- Ask the user if they want to enter the data again (y/n).
- If 'n', then the program ends, otherwise it should clear the student class object and repeat the loop (ask the user to enter new data...).

Note: The statement 'cin >> variable;' leaves a newline character in the buffer. This can be a problem if you are mixing 'cin >> variable;' and the 'getline' function (that reads a line of input). Use 'cin.ignore' function to discard the newline character after using 'cin'.

**Filename:** student.cpp

---

## 2. BoxOfProduce Class (30 points)

Your Community Supported Agriculture (CSA) farm delivers a box of fresh fruits and vegetables to your house once a week. For this programming project, define the class *BoxOfProduce* that contains exactly three bundles of fruits or vegetables. You can represent the fruits or vegetables as an array of type string. Add appropriate constructors and accessor/mutator functions to get or set the fruits or vegetables stored in the array. Also write an output function that displays the complete contents of the box on the console.

Next, write a *main* function that creates a *BoxOfProduce* with three items randomly selected from this list:

- *Broccoli*
- *Tomato*
- *Kiwi*
- *Kale*
- *Tomatillo*

Do not worry if your program randomly selects duplicate produce for the three items. Next, the main function should display the contents of the box and allow the user to substitute any one of the five possible fruits or vegetables for any of the fruits or vegetables selected for the box. After the user is done with substitutions it should output the final contents of the box to be delivered. Then it should ask if the user wants to create another box and if yes, it should repeat the above steps. It should keep doing so until the user chooses not to create another box of produce.

Finally, add a static variable to your class that keeps track of the total number of boxes of produce created and a static function that returns that value. Display this value in the main function at the end of each iteration of the main loop.

**Filename:** produce.cpp

---

### 3. Operator Overloading (40 points)

Define a class for rational numbers. A rational number is a number that can be represented as the quotient of two integers. For example, 1/2, 3/4, 64/2, and so forth are all rarional numbers. (By 1/2 and so on we mean that everyday fractions, not the integer division this expression would produce in a C++ program.) Represent rational numbers as two values of type *int*, one for the numerator and one for the denominator. Call the class *Rational*. Include a constructor with two arguments that can be used to set the member variables of an object to any legitimate values. Also include a constructor that has only a single parameter of type *int*; call this single parameter *wholeNumber* and define the constructor so that the object will be initialized to the rational number *wholeNumber*/1. Include a default constructor that initializes an object to 0 (that is, 0/1). Overload the input and output operators >> and <<. Numbers are to be input and output in the form 1/2, 15/32, 300/401, and so forth. Note that the numerator, the denominator, or both may contain a minus sign, so -1/2, 15/-32, and -300/-401 are also possible inputs. Overload all the following operators so that they correctly apply to the type *Rational*: ==, <, <=, >, >=, +, -, *, and /. The main() should test your class and the functions.

*Hints*: Two rational numbers a/b and c/d are equal if a*d equals c*b. If b and d are positive rational numbers, a/b is less than c/d provided a*d is less than c*b. You should include a function to normalize the values stored so that, after normalization, the deniminator is positive and the numerator and denominator are as small as possible. For example, after normalization 4/-8 would be represented the same as -1/2.

**Filename:** rational.cpp

---