# CSC-24400 Programming Project #4
**DUE: Friday, December 2, 2022, 11:59 PM**

## 1 Objectives

- writing a C++ class on your own
- working with a binary search tree
- using the const keyword relative to parameters, methods and return values

## 2 Problem Statement

You will be writing a `C++` class called `CandySet` that represents a collection of Candy. You are being given a class called `Candy` - both definition (`Candy.hpp`) and implementation (`Candy.cpp`); please make sure to use this class, as it should make your task *much* easier.

You *must* store the definition for your class (which *must* be called `CandySet`) in a file called `CandySet.hpp`. You may place your implementation in any `.cpp` file you like (although I'd recommend `CandySet.cpp` for sanity's sake.) Your `CandySet` class must contain the following public methods:

- a default constructor that takes no arguments. This should simply build an empty list capabale of storing any number of Candy objects.
- a constructor that takes a single argument of type `istream` . This constructor should read Candy data from the input stream, adding as many Candy objects as there are in the input stream to the collection. Each collection in the input stream will be specified on 1 line, as follows:

  ```
  <name> <calories> <rating> <number-of-ratings>
  ```

  Where:

  - `<name>` is a string (with no spaces in it) representing the name of a Candy bar.
  - `<calories>` is an integer representing the number of calories in a Candy bar.
  - `<rating>` is a positive real number between 1 and 4 representing the current average rating for the Candy in question.
  - `<number-of-ratings>` is a non-negative integer representing the number of times this Candy has been rated.

  Several example data files meeting this format are being provided to you. *Note that both inputting and outputting a* `Candy` *is code that has been provided to you! Do not waste time "re-inventing the wheel!"*

- a copy constructor that takes in as its only parameter an existing `CandySet` and duplicates the existing `CandySet`. The parameter should not be allowed to change and should be passed efficiently. Note that the resulting new CandySet object should be a brand new set that contains the same Candy as the set being copied.

- the `+=` operator should be overloaded so that the right hand side of this operator should be a `Candy` object (which should be guaranteed not to change and should be passed efficiently.) Using this operator should allow the left hand side (CandySet) operand to change. The method should efficiently modify the left hand side operand to be the original CandySet with the right hand side Candy operand now included in the left hand side CandySet. If the left hand side CandySet already contains the Candy named in the right hand side, then the left hand side should not be modified. Finally, the left hand side should be returned as a constant and as efficiently as possible.

- the `-=` operator should be overloaded so that the right hand side of this operator should be a `Candy` object (which should be guaranteed not to change and should be passed efficiently.) Using this operator should allow the left hand side (CandySet) operand to change. The method should efficiently modify the left hand side operand to be the original CandySet with the right hand side Candy operand removed from the left hand side CandySet (unless the named Candy does not exist in the CandySet.) Finally, the value of the left hand side should be returned as a constant and as efficiently as possible.

- the `-=` operator should also be overloaded so that the right hand side operand would be a string (constant and passed efficiently) with the left hand side being a CandySet. Using this operator should allow the left hand side (CandySet) operand to change. The method should efficiently modify the left hand side operand to be the original CandySet with the Candy corresponding to the string on the right hand side removed from the left hand side CandySet (unless the named Candy does not exist in the CandySet.) Finally, the value of the left hand side should be returned as a constant and as efficiently as possible.

- a method called `find` that takes a single parameter of type string (as a constant passed efficiently.) This method should return a (non-constant) pointer to a Candy object. If the Candy named by the parameter is found in this CandySet, then a pointer to that Candy object should be returned; if such is not found in the CandySet, then NULL should be returned.

- the `==` operator should be overloaded so that the right hand side is another CandySet object (passed efficiently as a constant.) This method should return a bool. If all of the Candy objects from the left hand side are found in the right hand side *and* all of the Candy objects from the right hand side are found in the left hand side, then the method should return true; otherwise the method should return false. Note that the values of calories and ratings are irrelevant when considering Candy equality; only the names of the Candy objects are to be considered.

- the `!=` operator should be overloaded so that the right hand side is another CandySet object (passed efficiently as a constant.) This method should return a bool. This should return the opposite value from what the corresponding overloaded `==` operator above would return.

- a method called `size` that takes no parameters and returns an integer. This method should simply return the actual number of different Candy objects in this CandySet. You should ensure that the method guarantees it will not modify the object it was called with.

- the `=` operator should be overloaded. The right hand side of this operator should be another CandySet (which should be guaranteed not to change and should be passed efficiently.) Using this operator should cause the operand on the left hand side to change to be a copy of the one on the right hand side. Be warned that the result should not "share" the same internal array! Finally, this method should return the copied CandySet (efficiently, as a constant.)

- a method called `clear` that simply empties out the set. After calling this, it should appear as if there are no Candy objects in the current set. This method should return nothing.

- the `<<` operator should be overloaded for a CandySet object. Remember that this method should return a reference to an object of type `ostream`. It should print each Candy object found in the set to the stream, one Candy object per line ***IN SORTED ORDER, LOWEST TO HIGHEST.***
- the `>>` operator should be overloaded for a CandySet object. Remember that this method should return a reference to an object of type `istream`.

A few final notes:

- You *MUST* use a *binary search tree* to represent your `CandySet`. Using anything else will result in a *very* poor grade on this project.
- *You may NOT use templates in any way when solving this project.* Using such will result in a *very* poor grade on this project.

## 3 Bonus Points

In addition to submitting the project at least 48 hours early for 5 bonus points, you may also submit a single main function test case file of your own to Dr. Blythe via e-mail (`sblythe@lindenwood.edu`) for an additional 2 points. In order to count for bonus points, your test case must:

- compile successfully.
- not effectively be a duplicate of an existing test case - either one of the test cases that I have provided or one that another student has submitted.
- be submitted no later than 96 hours (4 days) before the project due date.

## 4 What To Hand In

You will be submitting a zip, tar.xz, or tgz file containing your source code and a `read.me` file to Canvas. Make sure that you place the project into a single folder (which may contain sub-folders).

The `read.me` file should include information about your project including (but not limited to):

- your name
- the date
- the platform you developed your code on (Windows, Linux, ...)
- any special steps needed to compile your project
- any bugs your program has
- a brief summary of how you approached the problem

You might also want to consider adding things like a "software engineering log" or anything else you utilized while completing the project.

## 5 Grading Breakdown

| | |
|---|---|
| Correct Submission | 10% |
| Code Compiles | 10% |
| Following Directions | 30% |
| Correct Execution | 40% |
| Code Formatting/Comments/`read.me` | 10% |
| *Early Submission Bonus* | *5%* |

## 6 Notes & Warnings

- For most people, this is not the kind of project that you will be able to start the day before it is due and successfully complete. My recommendation is to start *now*.
- If you have any questions about anything involved with this project, ask me. If you don't ask for help, I will not know that you want it. Do not hesitate to attend my office hours, lab hours, or e-mail me questions!
- *Start now*!
- Projects may *not* be worked on in groups or be copied (either in whole or in part) from *anyone or anywhere*. This also means that you may not allow your code to be copied, either in part or in whole, by anyone - such would mean that you (as a provider of code) have also cheated. Failure to abide by this policy will result in disciplinary action(s). See the course syllabus for details.
- *START NOW*!
- Have you started working on this project yet? If not, then *START NOW !!!!!*