Theo Rowlett
CS202-Winter ‘21
Project 3

Overview
The objective of Project 3 is to build an application to manage a list of contacts, where each contact could potentially have three different “devices”. Also required is to demonstrate operator overloading and how it can simplify our code. Another requirement is to organize these contacts within either a binary search tree or a 2-3 tree. I have elected to go with a 2-3 tree design, if for no other reason than I haven’t had as much experience with it and I could use the practice.

Operator Overloading
I’ve chosen to implement operator overloading within the Contact class, as there it makes sense to help clean up our code. Since we are going to be comparing Contact objects during an insertion operation, the equality and not equal to operators seem like it would be a good fit. Also the greater than or less than operators would be extremely useful. Since we’re comparing the name within the Contact object, which is a string type from our custom string class, I will most likely be calling a function from that class to do the comparing. Also required by this program is overloading the +, the +=, and [] operators, although I haven’t decided yet what purpose they should serve.

Contact class
Since each Contact will contain a linearly linked list of devices, run time type identification will be utilized to add the objects to each contact correctly. When adding a device to a contact, we can attempt to dynamically cast to the different derived devices, thereby ensuring that we call the correct copy constructor for the device object.

The 2-3 Tree
The Tree class will contain a single pointer (root) to a Leaf object, and each Leaf object will have 2 Contacts, called the left key and the right key, as well as pointers to left, right, and middle. The Tree class “has a” Leaf, and the Leaf “has a” contact (really it has two, but that’s semantics). The benefit of using a 2-3 tree vice a binary search tree is efficiency. At the worst case, a binary search tree could slow down to O(n) efficiency for insertion, retrieval, and deletion. Since a 2-3 tree is sorted, it will always support insertion, retrieval, and deletion at a O(log(n)) speed.

Devices
Since each contact will contain a linearly linked list of up to 3 devices, it makes sense to tackle this problem with a hierarchical approach. This program is required to support three different means of communication devices. For my project I have chosen a cell phone, discord, and email. Both the cell phone and discord support a “paid tier”, the email service does not because in 2021 that is extremely uncommon.

Our Device class will be an abstract base class, meaning that no Device objects will be created that are only Device Objects. Instead they will always be one of the derived class objects. There are two pure virtual functions within the Device object, the display and change device functions.

Theo Rowlett
CS202-Winter ‘21
Project 3
Since each derived class has its own data members that are different, the change device function will call functions to change the specific data members within that class.

Since each class has a change_device function, we can avoid using getters and setters and make all changes within the class. The change_device function can call functions that are private to that class, so nothing outside of the class can make changes to the data within.

String class
Also implemented for this project will be a custom string class. It will have operator overloading for the equality, does not equal, addition (concatenation), and ostream output operators.