Theo Rowlett
CS 202 Winter ‘21
Project 3

Overview
The purpose of project 3 is to build an address book and display competency when implementing operator overloading. Required operators have to include =,+,+=,[],==,!=. Although they are written, only equality and conditional operators are used in the application. The contact book must be a type of binary tree. Although I attempted to implement a 2-3 tree, it was decided to scale back to a binary search tree.

Overloaded operators
Overloaded operators implemented include =,+,+=,[],==,!=,<, and > within the contact class. By making comparison operators code is streamlined to a much higher degree. For example, it we wanted to compare the names of two different contact it would look something like:

```
bool Contact::compare(const Contact & to_compare)
{
        If (strcmp(name,to_compare->return_name())<0)
                return true;
        return false;
}
```

Can be reduced to:

```
bool val;
If (this < to_compare)
        val = true;
```

This makes the code much more readable and maintainable.

Tree Structure
The tree is made up of individual contacts, sorted by comparing the first names of the contacts. No attempts were made to balance the tree, but full and individual removal is supported. Sorting makes use of the ==,!=,<, and > operators within the contact class. These overloaded operators use the overloaded operators from within the custom string class implemented for this project.

For removal of a single node within a binary search tree 3 cases have to be considered.

1. Node to be deleted is a leaf. The easiest situation requires to extra traversals. Although if the node is not the root, another traversal is necessary to set the parents pointer to null. This could have been avoided by using a look ahead approach, but by the time I realized it I was too far along with what I had already written.
2. Node to be deleted only has a left child. Copy the data from the child, remove the pointer to the child.
3. Node has children in the right subtree. The node to be removed has to be replaced with the inorder successor. This requires traversing further through the subtree.

Device Structure

Theo Rowlett
CS 202 Winter ‘21
Project 3
Each contact can have 3 different types of “devices” (email and discord aren’t actually devices, but it works within the context of the application). To solve for that, the device class is an abstract base class with 3 inherited classes to represent email, discord, and a cell phone. Each function has a display and change device function that is common. We can use downcasting to populate a linearly linked list of devices within each contact.