Test Cases

*For each test file, I tested instances where the hash table was empty, partially filled, and completely filled.

Double Hashing:

- Tested to create a hash table of size x
- Tested inserting multiple elements, including inserting when the hash table is full (expect failure), inserting when there is a collision using first hash function and collisions using secondary hash function, and inserting the same student number that has already been inserted (expect failure).
- Tested searching for students that were in the hashtable (expect success) and those that weren't (expect failure). As well, tested searching for a student that was inserted and has since been deleted (expect failure).
- Tested deleting students that were in the hashtable (expect success) and those that weren't (expect failure). As well, I tested deleting the same student twice (expect success on the first attempt and failure on the second).

Chaining:

- Tested to create a hash table of size x.
- Tested inserting multiple elements, including inserting when there is a collision (expect chaining) and inserting the same student number that has already been inserted (expect failure).
- Tested searching for students that were in the hashtable (expect success) and those that weren't (expect failure). As well, tested searching for a student that was inserted and has since been deleted (expect failure).
- Tested deleting students that were in the hashtable (expect success) and those that weren't (expect failure). As well, I tested deleting the same student twice (expect success on the first attempt and failure on the second).
- Tested print, including indexes that have no students, and ones that have multiple. Tested to make sure they are printing in descending order. Tested if you delete a student that it prints out the chain without that student.

Performance Consideration

Double Hashing: For insert, delete, and search, in the best case no collisions occur, therefore, the time complexity will be O(1). Worst case, we have n iterations of the hash function, therefore the time complexity would be O(n). However, the worst case is rare, therefore the average insertion time is O(1).

Chaining: For insert, delete, and search, 1 is the constant running time to reach the slot location, m is the number of elements stored, and n is the number of slots available in the hash table. Measuring when to decide to increase the hash table size is m/n. Therefore, the time complexity is O(1+m/n) -> O(1+1) -> O(1).