ALL OF THIS INFORMATION CAN ALSO BE FOUND IN THE README.md

## Compiling commands
In order to compile all of the required java files, make sure that you are in the parent directory of bin and uga (It should be CSCI4370 Project 1 for example).
 - Once in the main project directory, enter the following command:
```
    javac -d bin .\uga\cs4370\mydb\impl\*.java
```

## Running command
In order to run our Project 3, make sure that you are in the main project directory (this should be the parent directory of bin and uga)
 - Once in the main project directory enter the following command:
```
   java -cp bin uga.cs4370.mydb.impl.Main
```

## Project Explanation
Our project implements indexing on tables with periodic rehashing using the polynomial rolling hash function.
Each hash table tracks the average chain length, and if it reaches or exceeds .75, then the hash table rehashes.

### Insertion:
To store values in the hash table, the Relation will insert a cell into the hashtable.
A hash is calculated from a string stringified from the cell's indexed attribute.
This hash is then modulus'd by the length of the bucket array to get an index to store the cell's information at.
The cell's index and hash key is then stored at a linked list which is pointed to at the index previously calculated.
If the average chain length meets or exceeds .75, the hash table rehashes by creating a new array of buckets twice the
size of the previous one, and restores all the elements by using the larger array size as the value to be modulus'd by.

### Searching:
When searching, the program hashes the specified value and then searches for it at the index where it would be.
With no to minimal collisions, as is the goal, the value can be searched for much more quickly than searching an entire table.

### Joining:
When joining, the program will iterate through every row in one table and search for a matching value for each same attribute
present within table 2.
If there is an index on the current attribute being searched for, the program will use the search method on the index for that

attribute, which allows much faster retrieval.
After all matching rows are found, they are then checked for any remaining present non indexed attributes in table 2, and subsequently
added to the joined relation if every present attribute in table 2 matches.

# Observations
We completed 2 tests to compare the results of joining indexed table vs non indexed tables.
The first test was comparing the performance of joining a short (5 values in table 1) non indexed vs indexed table.
Each join was performed 10 times.

These are the results:

```
Short Non-Indexed Join
-------------------------------------------
Minimum Time: 8.09E+04 ns
Maximum Time: 1.10E+05 ns
Average Time: 8.63E+04 ns

Short Indexed Join
-------------------------------------------
Minimum Time: 7.78E+04 ns
Maximum Time: 3.08E+05 ns
Average Time: 1.38E+05 ns
```

As we can see, the non-indexed join performed considerably better than the indexed join. This is because although searching is faster
in the indexed table, the overhead created by indexing outweighed the marginal added search time in the non indexed join, due to not many rows
being searched through.

The second test was comparing the performance of joining a long (1000 values in table 1) non indexed vs indexed table.

These are the results:

```
Long Non-Indexed Join
-------------------------------------------
Minimum Time: 2.15E+07 ns
Maximum Time: 8.07E+07 ns
Average Time: 2.87E+07 ns

Long Indexed Join
-------------------------------------------
```

Minimum Time: 1.76E+06 ns
Maximum Time: 3.05E+06 ns
Average Time: 2.12E+06 ns

As we can see, the indexed join completed on average 10 times better than the non-indexed join. This is because in a join between two
considerably large tables, the much more efficient search time gained from indexing far outweighs the overhead cost of indexing. Comparing every row
from table 2 with table 1 greatly increases the search time compared to indexing.

## Conclusion
If the sizes of the tables are relatively small, comparing every row in table 1 with every row in table 2 usually outperforms
indexing due to the overhead cost of indexing. However, if they are relatively large, indexing is vastly superior and only becomes more efficient
with added data.