Formal Languages and Compiler Design – Lab 2: Symbol Table Implementation

I chose to implement the symbol table using a Hash Table with separate chaining as a collision resolution method. The symbol table is initialized with a given capacity (a prime number is recommended to be used) corresponding to the number of buckets of the inner hash table.

The operations allowed on the symbol table are the following:

- **Adding** a new symbol (If the symbol is already present in the table it will not be added again)
- **Removing** an existing symbol
- **Checking the existence** of a given symbol
- **Retrieving the position** of a given symbol
- **Retrieving the size** (number of entries) from the symbol table

Internally, the symbol table has 3 fields:

- table, which is a list of buckets
- size, the number of entries in the symbol table
- position, a number which is incremented with every new addition to the table

In my implementation I use two utility classes:

- Pair, which allows me to store new entries into the table together with their unique identifier/position. The unique identifier reflects the order in which the symbol was added to the table and it is the number returned whenever we query the position of a given element in the symbol table.
- Bucket, inner class which abstracts a "chain" (linked list) of symbols that hash to the same value i.e. belong to the same "bucket" of the hash table. The class methods are wrappers for accessing the content of a specific bucket.

Link to Github: https://github.com/912-Crismariu-Ioana/FLCD