Assignment 2

Muresan Ciprian-Adrian, 938pr

Github link: https://github.com/muresanciprian99/flcd-2021

In this assignment, I implemented a class to represent a Symbol Table. I've used Python as my programming language of choice and used a hash table data structure for internal representation of the ST.

The collision resolution method applied during the hashing process is *open addressing with linear probing.* The hash function h(x) is defined by h(x) = x % size, where x is the element hashed and size is the size of the table, represented by a large prime number in order to avoid rehashing. There is no rehashing mechanism implemented in case the table gets fully occupied.

Open addressing with linear probing is defined as follows. If the slot determine by h(x) is occupied, then the next slot checked is (h(x) + 1) % size. If the new slot is occupied too, the program will check (h(x) + 2) % size and so on, until an open slot is found.

Methods used in the class SymbolTable are the following:

- The constructor, where the size of the hash table is defined and the hash table is initialized. A Python list is used for representation

```
def __init__(self):
    # size of hash table is a large prime number in order to avoid rehashing
    self.__size = 863
    self.__table = []
    for i in range(self.__size):
        self.__table.append(None)
```

- The insert method, which hashes the token, checks for collision and adds the token to the list

```
def insert(self, token):
    # calculate hash value
    hash_value = 0
    if isinstance(token, int):
        hash_value = token % self.__size
    elif isinstance(token, str):
        for c in token:
            hash_value += ord(c)
        hash_value = hash_value % self.__size

    # check for collision and add new token
    if self.__table[hash_value] is not None:
        new_hash = (hash_value + 1) % self.__size
        while new_hash is not hash_value and new_hash < self.__size:
            if self.__table[new_hash] is None:
                hash_value = new_hash
                break
            new_hash += 1
    self.__table[hash_value] = token
```

- The pos method, that for a given token, verifies if it exists in the list, if not, it adds it and then returns its position

```
def pos(self, token):
    if self.__table.count(token) == 0:
        self.insert(token)
    return self.__table.index(token)
```