https://github.com/dragoscrisan7/Univeristy/tree/ff9939055871cf8617263db08aca1a4bb3af1a47/Year3/Semester%201/Limbaje%20Formale%20si%20Tehnici%20de%20Compilare/Projects/Lab2

```
class HashTable:
    def __init__(self, size=100, load_factor=0.7):
        self.size = size
        self.load_factor = load_factor
        self.table = [None] * self.size
        self.count = 0

    def resize(self, new_size):
        old_table = self.table
        self.size = new_size
        self.table = [None] * self.size
        self.count = 0
        for entry in old_table:
            if entry is not None:
                for key, value in entry:
                    self.add(key, value)

    def hash(self, key):
        if isinstance(key, int):
            return key % self.size
        elif isinstance(key, str):
            return sum(ord(char) for char in key) % self.size
        else:
            raise ValueError("Unsupported key type")

    def getSize(self):
        return self.size

    def getHashValue(self, key):
        return self.hash(key)

    def add(self, key, value=None):
        if self.count / self.size >= self.load_factor:
            self.resize(2 * self.size)

        index = self.hash(key)
        if self.table[index] is None:
            self.table[index] = []
        for i, entry in enumerate(self.table[index]):
            if entry[0] == key:
                if value is not None:
                    self.table[index][i] = (key, value)
                return (index, i)
        self.table[index].append((key, value))
        self.count += 1
        return (index, len(self.table[index]) - 1)

    def contains(self, key):
        index = self.hash(key)
        if self.table[index] is not None:
            for entry in self.table[index]:
                if entry[0] == key:
                    return True
```

```
        return False

    def getPosition(self, key):
        index = self.hash(key)
        if self.table[index] is not None:
            for i, entry in enumerate(self.table[index]):
                if entry[0] == key:
                    return (index, i)
        return (-1, -1)

    def __str__(self):
        return str(self.table)


class SymbolTable:
    def __init__(self, size=100):
        self.table = HashTable(size)

    def add(self, key, value=None):
        return self.table.add(key, value)

    def has(self, key):
        return self.table.contains(key)

    def getPosition(self, key):
        return self.table.getPosition(key)

    def __str__(self):
        return f"Symbol Table: {self.table}"


if __name__ == "__main__":
    symbol_table = SymbolTable(size=77)
    pos = (-1, -1)
    pos2 = (-1, -1)
    pos3 = (-1, -1)
    try:
        pos = symbol_table.add("apple")
        print(f"apple {pos}")
        print(f"banana {symbol_table.add('banana')}")
        print(f"cherry {symbol_table.add('cherry')}")
        print(f"date {symbol_table.add('date')}")
        print(f"elderberry {symbol_table.add('elderberry')}")

        print(f"2 {symbol_table.add(2)}")
        print(f"5 {symbol_table.add(5)}")
        pos2 = symbol_table.add(50)
        print(f"50 {pos2}")
        print(f"30 {symbol_table.add(30)}")
        print(f"111 {symbol_table.add(111)}")
        print(f"77 {symbol_table.add(77)}")
        print(f"95 {symbol_table.add(95)}")

        print(f"str1 {symbol_table.add('str1')}")
        print(f"str2 {symbol_table.add('str2')}")
        print(f"str3 {symbol_table.add('str3')}")
        print(f"str4 {symbol_table.add('str4')}")
```

```
        pos3 = symbol_table.add("str5")
        print(f"str5 {pos3}")

        print(f"apple {symbol_table.add('apple')}")
    except Exception as e:
        print(e)

    print(symbol_table)

    try:
        assert symbol_table.has("apple") == pos, f"apple does not have
position {pos}"
        assert symbol_table.getPosition(50) == pos2, f"50 does not have
position {pos2}"
        assert symbol_table.getPosition("str5") == pos3, f"str5 does not have
position {pos3}"
        assert symbol_table.has("aaaa") == (-1, -1), "aaaa exists in the
table"
        assert symbol_table.getPosition(95) == (2, 2), "95 does not have
position (2, 2)"
    except AssertionError as e:
        print(e)

    try:
        print(f"77 {symbol_table.getPosition(77)}")
        assert symbol_table.getPosition(77) == (2, 2), "77 does not have
position (2, 2)"
    except AssertionError as e:
        print(e)

    try:
        print(f"elderberry {symbol_table.has('elderberry')}")
        assert symbol_table.has('elderberry') == pos, f"elderberry does not
have position {pos}"
    except AssertionError as e:
        print(e)
```