Chis Sergiu
group 932/1

Lab2 documentation

Github: https://github.com/SergiuChis/FLCD

For the implementation of the symbol table, a hash table is used. We can see in it's representation that it has a size an array of strings.

```
class SymbolTable
{
private:
    std::string* hash_table;
    int size;
```

```
SymbolTable::SymbolTable(int Size)
{
    size = Size;
    hash_table = new std::string[size];

    for (int i = 0; i < size; ++i)
    {
        hash_table[i] = "";
    }
}
```

Each element is added by running it through a simple hash function. The function sums up the ascii values of each character of the string, and returns the remainder of dividing the sum and the size of the table (sum % size).

```
int SymbolTable::hash(std::string Element)
{
    int sum_of_ascii_codes = 0;

    for (int i = 0; i < Element.size(); ++i)
    {
        sum_of_ascii_codes += (int)Element[i];
    }

    return sum_of_ascii_codes % size;
}
```

The add method uses the hash function to obtain the index at which we should place the element. If that place in memory is occupied, we try the next one. If all are occupied, we fail to add the element, because the table is full.

```
int SymbolTable::add(std::string Element)
{
    // std::cout << "Adding: " << Element << std::endl;
    int hash_value = hash(Element);
    // std::cout << "Hash value starting: " << hash_value << std::endl;
    int hv_copy = hash_value;
    int cycle = 0;

    while (hash_table[hash_value] != "")
    {
        if (hash_table[hash_value] == Element)
        {
            std::cout << "Already existing" << std::endl;
            return hash_value;
        }
        if (hash_value == hv_copy)
        {
            cycle++;
        }
        if (cycle == 2)
        {
            std::cout << "Table full" << std::endl;
            return hash_value;
        }
        hash_value = (hash_value + 1) % size;
    }

    hash_table[hash_value] = Element;
    // std::cout << "Added at index: " << hash_value << std::endl;
    return hash_value;
}
```