ЛАБОРАТОРНА РОБОТА №13.

СТРОКИ.

1. ВИМОГИ
   1.1. **Розробник**
   - Бельчинська Катерина Юріївна;
   - студентка групи КІТ-320;
   - 18 грудня 2020.

   1.2. **Індивідуальне завдання**

   Вирахувати для тексту частотну таблицю: для кожного символу визначити його частоту появи у тексті (число таких символів у тексті ділене на загальне число символів у тексті).

2. ОПИС ПРОГРАМИ
   2.1. **Функціональне призначення**

   Програму доцільно використовувати для розрахування частоти появи у даному тексті конкретного символу.

   2.2. **Опис логічної структури**

Функція 'main' виділяє пам'ять для заданого і результуючого масиву, викликає усі функції для обчислення частоти. Схема алгоритму функції наведена на рис. 1.

Функція countTextLength обчислює довжину заданого масиву. Схема алгоритму функції наведена на рис. 2.

Функція countOfUniqueElements обчислює кількість унікальних елементів. Схема алгоритму функції наведена на рис. 3.

Функція checker перевіряє кожен елемент на повтори. Схема алгоритму функції наведена на рис. 4.

Функція getsymbols переписує унікальні елементи в масив. Схема алгоритму функції наведена на рис. 5.

Функція getSymbolsCounts отримує кількість повторів кожного елементу. Схема алгоритму функції наведена на рис. 6.

Функція fillZeros ініціолізує результуючий масив. Схема алгоритму функції наведена на рис. 7.

Функція getSymbolsFrequencies вираховує та записує в масив частоту появи кожного елементу. Схема алгоритму функції наведена на рис. 8.

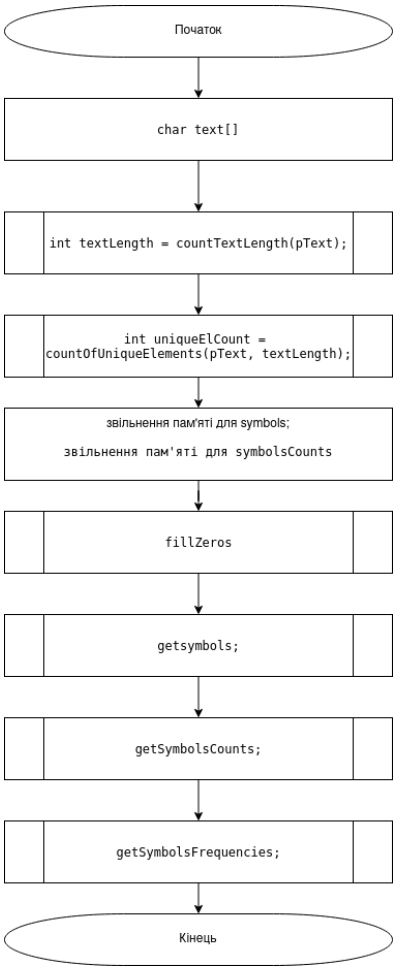

Рис.1. Схема алгоритму функції main

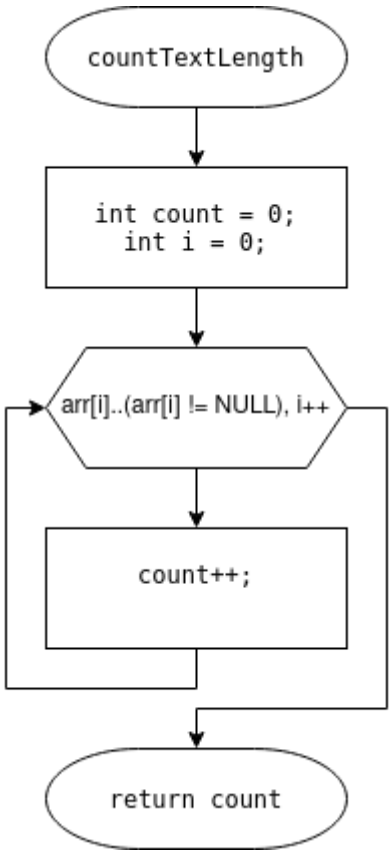

Рис.2. Схема алгоритму функції CountTextLength.

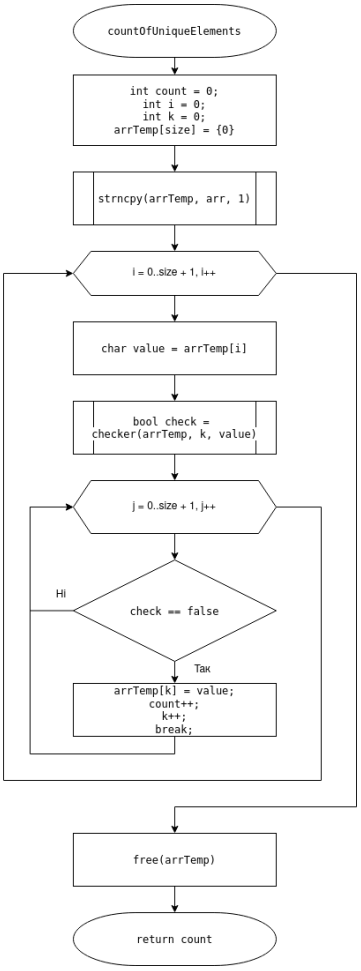

Рис.3. Схема алгоритму функції CountOfUniqueElements.

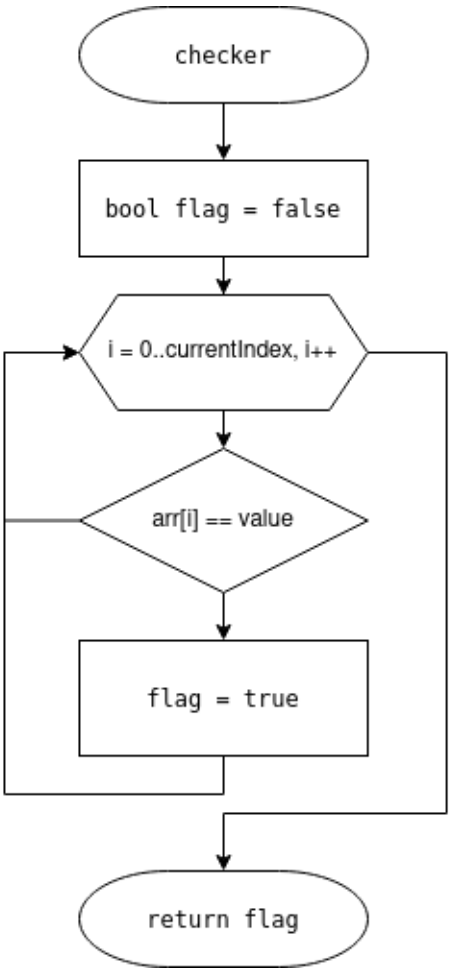

Рис.4. Схема алгоритму функції checker.

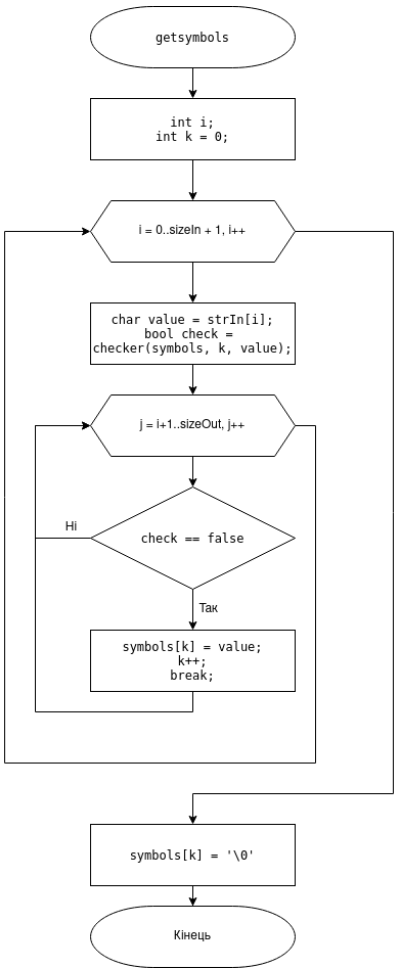

Рис.5. Схема алгоритму функції getsymbols.

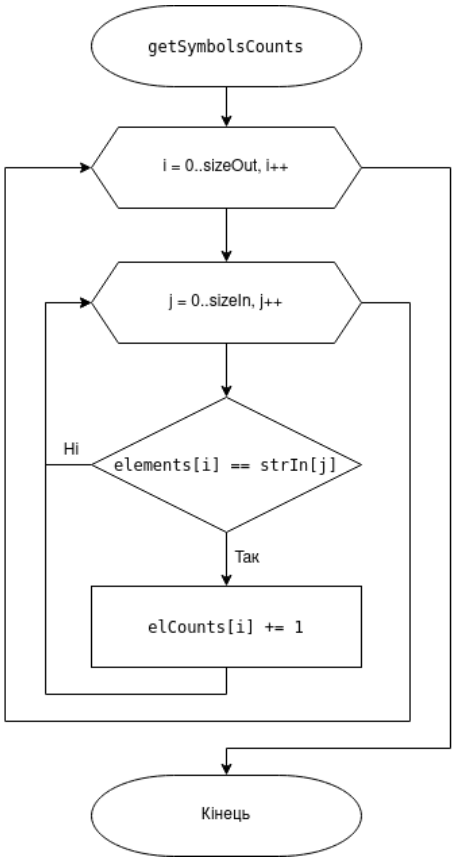

Рис.6. Схема алгоритму функції getSymbolsCounts.

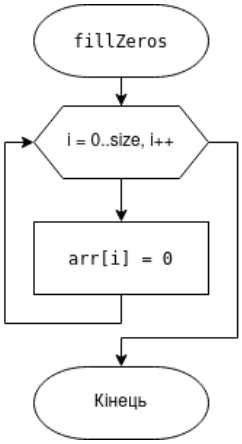

Рис.7. Схема алгоритму функції filZeros.

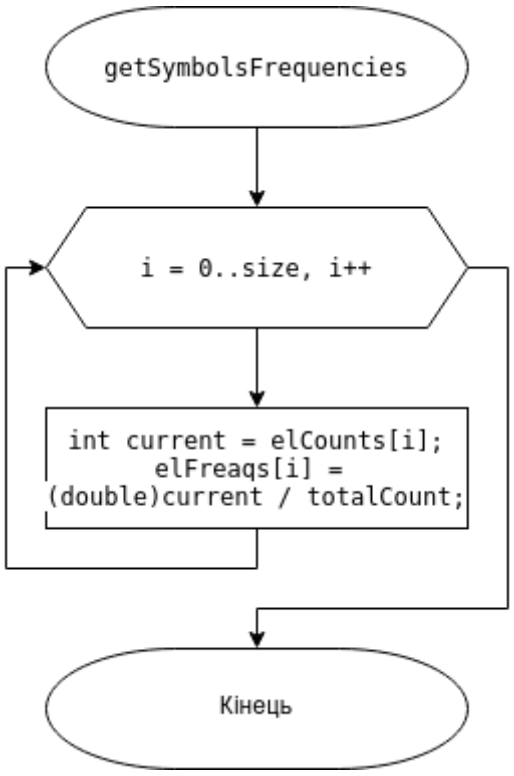

Рис.8. Схема алгоритму функції getSymbolFrequencies.

2.3. Структура проекту

```
.
├── doc
│   ├── assets
│   │   ├── checker.png
│   │   ├── countOfUniqueElements.png
│   │   ├── countTextLength.png
│   │   ├── debugger.png
│   │   ├── Doxygen1.png
│   │   ├── Doxygen2.png
│   │   ├── fillZeros.png
│   │   ├── getSymbolsCounts.png
│   │   ├── getSymbolsFrequencies.png
│   │   ├── getsymbols.png
│   │   ├── main.c.png
│   │   └── Valgrind.png
│   ├── lab13.docx
│   ├── lab13.md
```

```
│   └── lab13.pdf
├── Doxyfile
├── Makefile
├── README.md
├── task1
│   ├── README.md
│   └── src
│       ├── lib.c
│       ├── lib.h
│       └── main.c
├── task2
│   ├── README.md
│   └── src
│       ├── lib.c
│       ├── lib.h
│       └── main.c
├── task3
│   ├── README.md
│   └── src
│       ├── lib.c
│       ├── lib.h
│       └── main.c
└── task4
    ├── README.md
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

2.4. Генерування Doxygen-документації

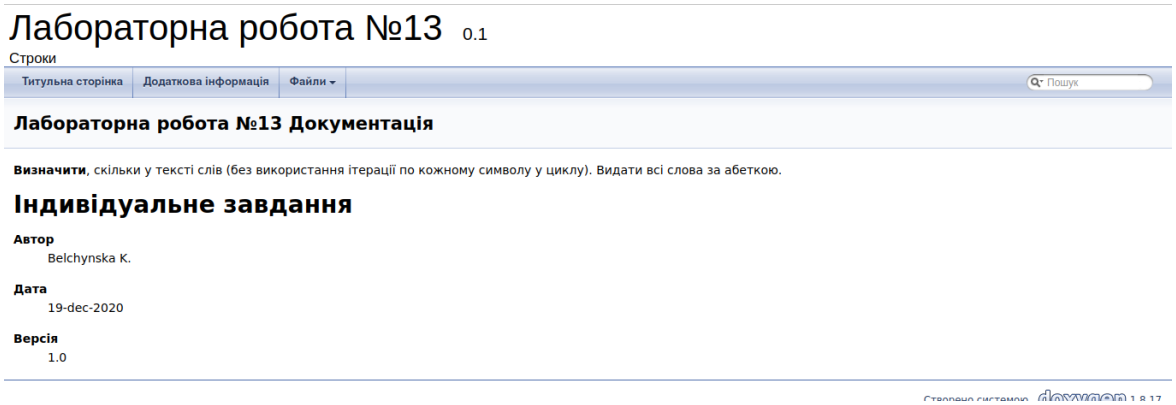

Рис. 9. Титульна сторінка Doxygen

doc	
task1	
src	
lib.c	Файл з реалізацією функцій
lib.h	Файл з прототипами функцій
main.c	Головний файл з викликом функцій, виділенням пам'яті для динамічних масивів та заданням вхідного тексту
task2	
src	
lib.c	Файл з реалізацією функцій
lib.h	Файл з прототипами функцій
main.c	Головний файл з викликом функцій, виділенням пам'яті для динамічних масивів та заданням вхідного тексту
task3	
src	
lib.c	
lib.h	Файл з прототипами функцій
main.c	Головний файл з викликом функцій, виділенням пам'яті для динамічних масивів та заданням вхідного тексту
task4	
src	
lib.c	Файл з реалізацією функцій
lib.h	Файл з прототипами функцій
main.c	Головний файл з викликом функцій, виділенням пам'яті для динамічних масивів та заданням вхідного тексту

Рис. 10.

Структура файлів в Doxygen.

2.5. Перевірка на утечки пам’яті за допомогою Valgrind:

Рис. 11 Перевірка на утечки пам’яті

```
kate@kate-HP-ProBook-440-G3:~/Programming-Belchynska/lab13$ valgrind --leak-check=yes dist/main3.bin
==10881== Memcheck, a memory error detector
==10881== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10881== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10881== Command: dist/main3.bin
==10881==
==10881== Invalid free() / delete / delete[] / realloc()
==10881==    at 0x483CA3F: free (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==10881==    by 0x109DFC: main (main.c:64)
==10881==  Address 0x1ffefffcec is on thread 1's stack
==10881==  in frame #1, created by main (main.c:43)
==10881==
==10881==
==10881== HEAP SUMMARY:
==10881==     in use at exit: 0 bytes in 0 blocks
==10881==   total heap usage: 4 allocs, 5 frees, 77 bytes allocated
==10881==
==10881== All heap blocks were freed -- no leaks are possible
==10881==
==10881== For lists of detected and suppressed errors, rerun with: -s
==10881== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

3. ВАРІАНТИ ВИКОРИСТАННЯ

Хід роботи та результат виконання програми доцільно спостерігати у відлагоднику.

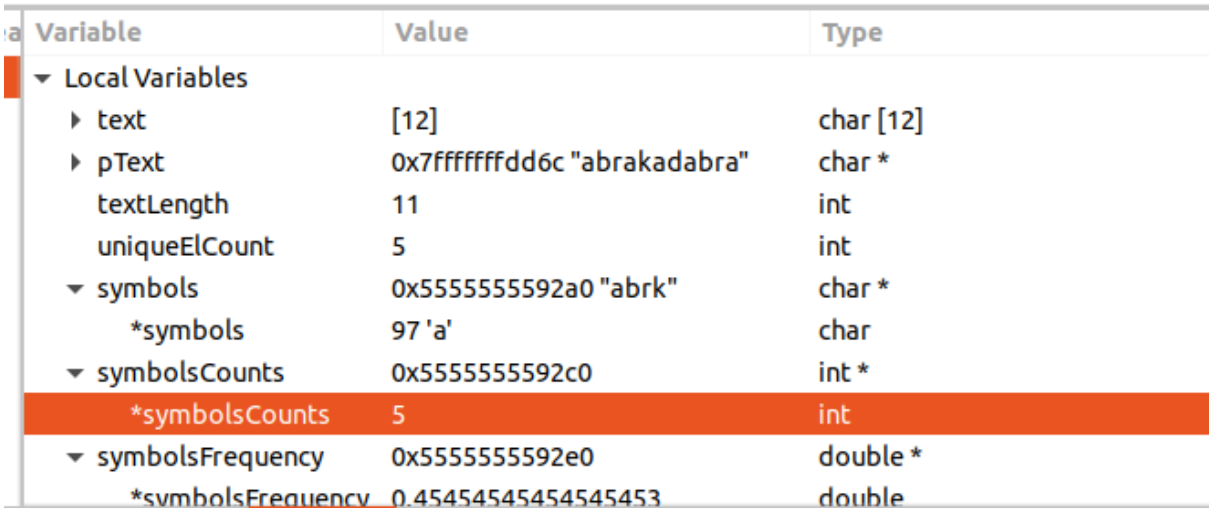

Рис. 12. Спостерігання за перебігом програми у відлагоднику.

ВИСНОВКИ

В ході даної лабораторної роботи я навчилася проводити дії над строками за допомогою бібліотечних функцій.