Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

Строки, массивы строк, операции над строками
Тема

Преподаватель _____ А. С. Черниговский
подпись, дата — инициалы, фамилия

Студент КИ19-17/1б 031939175 _____ А. Д. Непомнящий
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2020

1 Цель

Цель настоящей работы состоит в том, чтобы продолжить ознакомление с синтаксисом языка СИ, стандартом кодирования, изучить принципы работы строк и массивов строк в СИ.

2 Задачи

Для выполнения лабораторной работы необходимо выполнить следующие задачи:

На оценку 3 балла:

1) написать программу в соответствии с заданием;

2) отформатировать исходный код программы согласно стандарту оформления исходного кода;

На оценку 4 балла:

1) выполнить требования предыдущих пунктов;

2) для хранения строк использовать динамические массивы символов (размер массива определяется в процессе ввода);

3) организовать повтор программы по желанию пользователя;

4) добавить проверку входных аргументов на корректность;

На оценку 5 баллов:

1) выполнить требования предыдущих пунктов;

2) реализовать меню пользователя, состоящее как минимум из 4-х пунктов (ввод данных, обработка данных, вывод результата на экран, выход);

3) для корректной работы меню организовать промежуточное хранение результата;

4) разбить программу на функции;

5) организовать чтение данных и запись результата в файл формата txt.

3 Описание задания

Дана строка с математическим выражением, содержащим цифры, точку, знаки математических операций и скобки. Сопоставить каждому символу свой токен (DIGIT для числа, DOT для точки и т.д.).

4 Ход выполнения

Ниже представлен листинг программы по заданию.

Листинг 1 – Код программы, решающей задачу

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <malloc.h>

enum SymbolCodes
{
    // Перечисление кодов символов для функции ввода строки
    BACKSPACE_KEY = 8,
    START_CHAR_RANGE = 32,
    END_CHAR_RANGE = 126
};

enum OperationsCodes
{
    // Перечисление кодов операций для организации главного меню.
    KEYBOARD_INPUT = 1,
    SHOW = 2,
    TASK = 3,
    CONSOLE_OUTPUT = 4,
    QUIT = 5
};

enum Sizes
{
    // Перечисление наибольших размеров массивов.
    INPUT_SIZE = 100,
    TOKEN_NAME_SIZE = 20
};

char* StrDynInput()
{
    // Функция для ввода строки без указания длины
    // источник - Лекция 3. Строки. Массивы строк. Операции над строками.pdf
    char* userStr = (char*) malloc(1 * sizeof(char));
    userStr[0] = '\0';
    char curChar = 0;
```

Продолжение листинга 1

```
int curSize = 1;
while (curChar != '\n')
{
    curChar = getchar();
    int deltaVal = 0; // Определяет, на сколько изменится длина массива
    int lengthDif = 0;
    // Если мы ситраем символы, а не пишем их,
    if (curChar == BACKSPACE_KEY)
    {
        deltaVal = -1; // то будем уменьшать длину массива
        lengthDif = 1; // и копировать строку до предпоследнего символа
    }
        // Иначе проверяем, входит ли введённый символ в диапазон печатных
    else
    {
        if (curChar >= START_CHAR_RANGE && curChar <=
                                        END_CHAR_RANGE)
        {
            deltaVal = 1; // Если да, то будем увеличиватьдлину на 1
            lengthDif = 2; // Не заполняем последние 2 символа -
            // оставлем мето для введённого символа и 0
        }
        else
        {
            continue;
        } // Если это не печатный символ, то пропускаем его
    }
    // Если стирать больше нечего, но пользователь всё равно жмёт Backspace,
    int newSize = curSize + deltaVal;
    if (newSize == 0)
    {
        continue;
    } // то мы переходим на следующую итерацию - ждём '\n'
    char* tmpStr = (char*) malloc(newSize * sizeof(char));
    if (tmpStr) // Проверяем, выделилась ли память
    {
        // Идём до предпоследнего символа, т.к. надо в конец записать 0
        for (int i = 0; i < newSize - lengthDif; ++i)
        {
            tmpStr[i] = userStr[i];
        }
```

Продолжение листинга 1

```
            if (curChar != BACKSPACE_KEY)
            {// Если введён печатный символ,
                tmpStr[newSize - 2] = curChar; // Добавляем его в строку
            }
            tmpStr[newSize - 1] = '\0';
            free(userStr);
            userStr = tmpStr;
            curSize = newSize;
        }
        else
        {
            printf("Couldn't allocate memory!");
            break;
        }
    }

    return userStr;
}

char* CycleInputString(char* stringToOutput, bool(* pChecker)(char*))
{
    // Функция для ввода строки с проверкой ввода.
    //
    // char* stringToOutput - строка, которую нужно выводить
    // ... в запросе ввода;
    // bool(* pChecker)(char*) - указатель на функцию, проверяющую
    // ... дополнительные условия.
    printf("%s\n", stringToOutput);
    char* stringToReturn;
    while (true)
    {
        stringToReturn = StrDynInput();
        if (pChecker(stringToReturn))
        {
            return stringToReturn;
        }
        printf("Wrong format!\n");
        free(stringToReturn);
    }
}
```

Продолжение листинга 1

```
int CycleInputInt(char* stringToOutput, bool(* pChecker)(int))
{
    // Функция для ввода целого числа с проверкой ввода.
    //
    // char* stringToOutput - строка, которую нужно выводить
    // ... в запросе ввода;
    // bool(* pChecker)(int) - указатель на функцию, проверяющую
    // ... дополнительные условия.

    int number;   // Необходимое число
    int position;   // Позиция числа в введенной строке
    char input[INPUT_SIZE];   // Строка для ввода

    // Считывает и проверяет ввод по нескольким условиям, до тех пор,
    // пока не будет введено корректно.
    while (true)
    {
        printf("%s\n", stringToOutput);
        fflush(stdout);
        char* fgetsRet = fgets(input, INPUT_SIZE, stdin);
        if (fgetsRet == NULL)
        {
            printf("Wrong format!\n");
            continue;
        }
        int inputLength = strlen(input) - 1;
        input[inputLength] = '\0';
        int sscanfRet = sscanf(input, "%d%n", &number, &position);
        if (position != inputLength)
        {
            printf("Wrong format!\n");
            continue;
        }
        if (pChecker && !pChecker(number))
        {
            printf("Wrong format!\n");
            continue;
        }
        if (sscanfRet == 1) break;
        printf("Wrong format!\n");
    }
```

Продолжение листинга 1

```
    return number;
}

bool OperationInputChecker(int operationCode)
{
    // Функция для вызова в функциях ввода с проверкой.
    // ... Возвращает true, если введенное значение может быть
    // значением кода операции в меню.
    //
    // int operationCode - число, которое нужно проверить.

    return operationCode >= KEYBOARD_INPUT && operationCode <= QUIT;
}

bool MathCharsInputChecker(char* stringToCheck)
{
    // Функция для вызова в функциях ввода с проверкой.
    // ... Возвращает true, если введенное значение может быть
    // строкой только с числами или с определенными символами.
    //
    // char* stringToCheck - число, которое нужно проверить.
    for (int i = 0; i < strlen(stringToCheck); i++)
    {
        char k = stringToCheck[i];
        if (!(k >= '(' && k <= '9' && k != ','))
        {
            return false;
        }
    }
    return true;
}

typedef struct
{
    // Структура для хранения токенов.
    char tokenName[TOKEN_NAME_SIZE];    // Название токена
    char content;   // Значение токена
} SomeToken;

typedef struct
{
```

Продолжение листинга 1

```
    // Структура для хранения строки и её длины
    char* content;    // Строка
    bool isFilled;    // true, если выделена память
    int length;    // Длина строки
} DynStr;

typedef struct
{
    // Структура для хранения динамического массива токенов
    SomeToken* content;    // Массив
    bool isFilled;    // true, если выделана память
} DynArrTokens;

void FreeDynArrStr(DynStr* arrToClean)
{
    // Функция для очистки памяти, выделенной под arrToClean
    if (arrToClean->isFilled)
    {
        free(arrToClean->content);
        arrToClean->isFilled = false;
    }
    arrToClean->length = 0;
}

void FreeDynArrTokens(DynArrTokens* arrToClean)
{
    // Функция для очистки памяти, выделенной под arrToClean
    if (arrToClean->isFilled)
    {
        free(arrToClean->content);
        arrToClean->isFilled = false;
    }
}


int main()
{
    DynStr objStr;
    objStr.isFilled = false;
    objStr.length = 0;
```

Продолжение листинга 1

```
    DynArrTokens tokens;
    tokens.isFilled = false;

    int operationCode;
    while (true)
    {
        printf("\n1. Enter the string with keyboard.\n"
               "2. Show the object string.\n"
               "3. Perform the task.\n"
               "4. Print result to the console.\n"
               "5. Quit.\n\n");
        operationCode = CycleInputInt(
                "Choose the command and enter its number",
                OperationInputChecker);

        // Ввод строки к анализу с клавиатуры
        if (operationCode == KEYBOARD_INPUT)
        {
            FreeDynArrStr(&objStr);
            FreeDynArrTokens(&tokens);
            objStr.content = CycleInputString("Enter string to analyze",
                                              &MathCharsInputChecker);
            objStr.isFilled = true;
            objStr.length = strlen(objStr.content);
        }

        // Вывод строки к анализу в консоль
        if (operationCode == SHOW)
        {
            if (objStr.isFilled)
            {
                printf("%s", objStr.content);
            }
            else
            {
                printf("No object string!\n");
            }
        }

        // Выполнение анализа
        if (operationCode == TASK)
```

Продолжение листинга 1

```
        {
            FreeDynArrTokens(&tokens);
            if (!objStr.isFilled)
            {
                printf("No object string!\n");
            }
            else
            {
                // Выделение памяти под массив токенов
                tokens.content = (SomeToken*) malloc(
                        objStr.length * sizeof(SomeToken*));
                tokens.isFilled = true;
                for (int i = 0; i < objStr.length; i++)
                {
                    // Непосредственно анализ
                    if (objStr.content[i] >= '0' && objStr.content[i] <= '9')
                    {
                        tokens.content[i].content = objStr.content[i];
                        strcpy(tokens.content[i].tokenName, "DIGIT");
                    }
                    if (objStr.content[i] == '/')
                    {
                        tokens.content[i].content = objStr.content[i];
                        strcpy(tokens.content[i].tokenName, "DIVISION");
                    }
                    if (objStr.content[i] == '.')
                    {
                        tokens.content[i].content = objStr.content[i];
                        strcpy(tokens.content[i].tokenName, "DOT");
                    }
                    if (objStr.content[i] == '+')
                    {
                        tokens.content[i].content = objStr.content[i];
                        strcpy(tokens.content[i].tokenName, "PLUS");
                    }
                    if (objStr.content[i] == '-')
                    {
                        tokens.content[i].content = objStr.content[i];
                        strcpy(tokens.content[i].tokenName, "MINUS");
                    }
                    if (objStr.content[i] == '*')
```

Продолжение листинга 1

```
                {
                    tokens.content[i].content = objStr.content[i];
                    strcpy(tokens.content[i].tokenName, "MULTIPL");
                }
                if (objStr.content[i] == ')')
                {
                    tokens.content[i].content = objStr.content[i];
                    strcpy(tokens.content[i].tokenName, "CLOSE_PARANTH");
                }
                if (objStr.content[i] == '(')
                {
                    tokens.content[i].content = objStr.content[i];
                    strcpy(tokens.content[i].tokenName, "OPEN_PARANTH");
                }
            }
        }


    }

    // Вывод результатов анализа в консоль
    if (operationCode == CONSOLE_OUTPUT)
    {
        if (!tokens.isFilled)
        {
            printf("No results for current string!\n");
        }
        else
        {
            for (int i = 0; i < objStr.length; i++)
            {
                printf("%s, %c\n", tokens.content[i].tokenName,
                       tokens.content[i].content);
            }
        }
    }

    // Выход из меню
    if (operationCode == QUIT)
    {
        break;
```

Окончание листинга 1

```
        }
    }
    // Очистка памяти и завершение программы
    FreeDynArrStr(&objStr);
    FreeDynArrTokens(&tokens);
    return 0;
```

5 Результат

Ниже представлены скриншоты с консольным выводом.

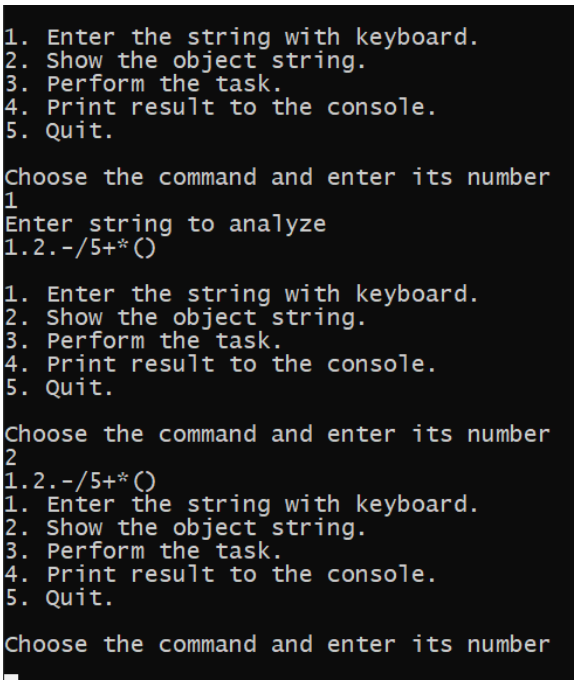

Рисунок 1 – Ввод строки, вывод её в консоль

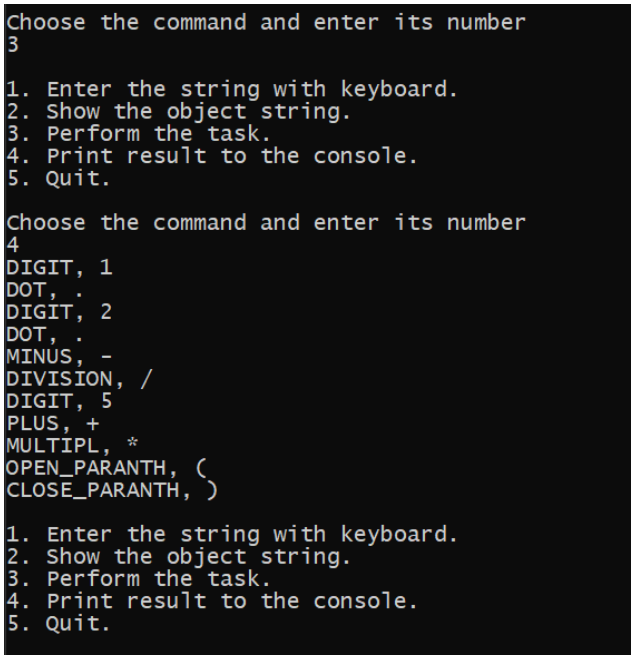

Рисунок 2 – Выполнение обработки, вывод результатов обработки в консоль

6 Выводы

Была написана программа, соответствующая поставленным задачам. В ходе работы были изучены некоторые аспекты синтаксиса языка программирования Си, в частности – принципы работы со строками и с массивами строк. Код был отформатирован в соответствии со стандартом, принятым в учебном заведении.