МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка строк на языке С

Студент гр. 0382 _____ Афанасьев Н. С.

Преподаватели _____ Чайка К. В.
Жангиров Т. Р.

Санкт-Петербург
2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент: Афанасьев Н. С.

Группа 0382

Тема работы: Обработка строк на языке С.

Вариант 17

Исходные данные:

Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских букв и цифр. Длина текста и каждого предложения заранее не известна.

Программа должна сохранить этот текст в динамический массив строк и оперировать далее только с ним.

Программа должна найти и удалить все повторно встречающиеся предложения (сравнивать их следует посимвольно, но без учета регистра).

Далее, программа должна запрашивать у пользователя одно из следующих доступных действий (программа должна печатать для этого подсказку. Также следует предусмотреть возможность выхода из программы):

Найти в предложениях все даты записанные в виде “<год> <месяц> <day>” (“1886 Jun 03”) и заменить их на строку показывающую сколько осталось часов до конца года.

Вывести все строки выделив слова на четных позициях красным цветом, а на нечетных зеленым.

Удалить все предложения, которые начинаются и заканчиваются на одно и то же слово.

Отсортировать предложения по увеличению сумме кодов символов первого слова в предложении.

Все сортировки должны осуществляться с использованием функции стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 02.11.2020

Дата сдачи реферата: 19.12.2020

Дата защиты реферата: 22.12.2020

Студент гр. 0382 _____ Афанасьев Н. С.

Преподаватели _____ Чайка К. В.
Жангиров Т. Р.

АННОТАЦИЯ

В процессе выполнения курсовой работы создавалась программа для обработки введённого пользователем текста в зависимости от операции, которую он выберет. Для удобства выводится контекстное меню, при выборе несуществующей операции выводится ошибка. Программа представляет собой CLI, поддерживающий ввод ASCII символов, для хранения теста память выделяется динамично. Разработка велась на операционной системе Windows 10 Home x64 в редакторе исходного кода Visual Studio Code с использованием компилятора MinGW.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – создать консольное приложение для обработки текста согласно запросам пользователя.

Для выполнения работы необходимо выполнить следующие задачи:

- Ввод, хранение и вывод текста
- Реализовать работу с датами для первой операции
- Реализовать выделение текста цветом для второй операции
- Создать возможность сравнивать фрагменты текста для первоначальной обработки и третьей операции
- Создать возможность сортировать текст для четвёртой операции

Для первой задачи, для хранения текста используется динамическая память (методы библиотеки *stdlib.h*), ввод осуществляется посимвольно, вывод осуществляется по предложениям через пробел (методы библиотеки *stdio.h*).

Для второй задачи, для работы с датами используется библиотека *time.h*.

Для третьей задачи, для работы с цветом используются управляющие (ESC) последовательности ANSI.

Для четвёртой задачи, используется посимвольное сравнение без учёта регистра либо реализованные функции для получения первого и последнего слова.

Для пятой задачи используется метод *qsort* из библиотеки *stdlib*.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Ввод текста

Для начала пользователю предлагается ввести текст. Для этого реализованы две функции:

Функция *char* getSentence()* предназначена для чтения одного предложения. Сначала выделяется память (с помощью метода *malloc*) для предложения (*sentence*) первоначального размера *sentenceSize = 40*. При необходимости выделяется дополнительная память (с помощью метода *realloc*). Чтение предложения заканчивается, когда попадается точка. Функция возвращает считанное предложение.

Функция *char** getText(int* size)* предназначена для чтения всего текста. Пока следующий символ не равен символу переводу строки, в переменную text с динамически выделенной памятью добавляется предложение, полученное при помощи *getSentence*. Количество считанных предложений записывается в переменную *size*, переданную в аргументах. Функция возвращает считанный текст.

2.2. Первичная обработка

По условию, необходимо удалить повторяющиеся предложения. Для этого реализована функция *char** removeDuplicate(char** text, int* size*), которая выделяет место для обновлённого текста, сравнивает предложения, и добавляет в новый список, если нет совпадений. Сравнение происходит следующим образом: предложение сравнивается посимвольно с другими предложениями, если они совпадают по длине (длина измеряется с помощью метода *strlen* из *string.h*); все символы переводятся в верхний регистр с помощью метода *toupper* из библиотеки *ctype.h* (т.к. сравнение без учёта регистра); при первом отличии сравнение двух предложений заканчивается. Количество оставшихся предложений записывается в переменную *size*, переданную в аргументах. Функция возвращает обновлённый текст.

2.3. Выбор операции

Вызывается функция *void showMenu(),* которая выводит контекстное меню. Кириллица при выводе обеспечена методом *setlocale(LC_ALL, "")* из библиотеки *locale.h* и методом *wprintf(L"...")* для вывода широких символов.

С помощью метода *scanf* считывается номер операции.

Далее идёт конструкция switch определяющая операцию по её номеру. Операции 1 – 4 осуществляются с помощью функции void *solve(short operation, char*** *text, int size, char** (*f)(char**, int*))*, которая выводит информативную строку с номером выбранной операции; выполняет переданную в аргументы функцию по обработке текста, каждая из которых принимает текст и указатель на количество предложений; выводит обновлённый текст и освобождает использованную в процессе память. Операция 5 (Выход) просто завершает выполнение программы. Для всех несуществующих операций выводится ошибка.

2.4. Первая операция

Первая операция заключается в замене даты формата <год месяц день> на число оставшихся до конца года часов. Она выполняется с помощью функции *char** *changeDate(char** text, int* size).* Сначала выделяется память для обновлённого текста. Для каждого предложения выделяется память для обновлённого варианта, затем изначальное предложение посимвольно копируется, пока не попадётся число. Если попалось число, то в новую переменную (позже освобождается) копируется область памяти в 11 символов (метод *strncpy*), начиная с этого числа (т.к. формат даты из условия – всегда 11 символов). С помощью метода *sscanf* из *stdio.h* проверяется соответствие описанного выше фрагмента паттерну вида *<4d 3s 2d>* (формат даты): если нет соответствия, копирование текста продолжается. Если соответствие есть, то из метода *sscanf* получаем записанные переменные *year, month, day*. Месяц через

ряд тернарных операторов переводится в число. Для сравнения дат используется библиотека *time.h*, которая предоставляет структуру *tm*, содержащую компоненты календарного времени. Создаются две таких структуры, содержащие найденную дату и дату начала следующего года. С помощью метода *mktime*, эти структуры переводятся в объекты типа *time_t*. В этом виде можно найти разницу в секундах (потом переводится в часы) между датами с помощью метода *difftime*. Чтобы вставить это значение (*diff*) нужно представить число в виде строки: сначала считается необходимое количество цифр с помощью выражения *floor(log10(diff)) + 1* (методы взяты из *math.h*), создаётся массив символов этого размера, перевод происходит с помощью метода *itoa* из *stdlib.h*. Полученная строка конкатенируется с предложением с помощью метода *strcat* из *string.h*. В результате, функция возвращает обновлённый текст.

2.5. Вторая операция

Вторая операция заключается в выделении слов на чётной позиции красным, а на нечётной – зелёной. Она выполняется с помощью функции *char** colorize(char** text, int* size)*, которая так же выделяет память для обновлённых текста и предложений. Далее копируются символы из оригинального текста следующим образом: если начинается новое слово, то вставляется последовательность *"\033[0;101;37m"* либо *"\033[0;102;37m"* (светло-красный и светло-зелёный фон соответственно) в зависимости от флага *isOdd* (чётность позиции); если слово заканчивается, то после него ставится последовательность *"\033[0;m"*, которая стирает все стили; во всех иных случаях копирование происходит без изменений. Функция выводит полученный текст. Эти последовательности переводятся в цвета, если они поддерживаются (в терминалах Linux – да, в консоли Windows 10 – изначально нет).

2.6. Третья операция

Третья операция заключается в удалении предложений, начинающихся и заканчивающихся на одно слово. Для начала были реализованы функции *char* firstWord(char* sentence)* и *char* lastWord(char* sentence),* которые копируют предложения до первого пробела (только первая – с начала предложения, а вторая – с конца). Сама операция выполняется с помощью функции *char** removeSimilarities(char** text, int* size),* которая выделяет место для обновлённого текста; для каждого предложения вызывает функции *first-* и *lastWord* (память под эти слова позже освобождается); если слова совпадают (без учёта предложения), то оно не добавляется в обновлённый список. Количество оставшихся предложений записывается в переменную *size*, переданную в аргументах. Функция возвращает обновлённый текст.

2.7. Четвёртая операция

Четвёртая операция заключается в сортировке предложений по возрастанию суммы кодов символов первого предложения. Она выполняется с помощью функции *char** sort(char** text, int* size).* Для удобства сортировки создаётся двумерный массив размера [*size*][*2*] (*size* – это кол-во предложений). В этом массиве хранятся элементы, состоящие из двух чисел: сумма кодов символов первого слова и индекс предложения. Сумма кодов находится просто суммированием через цикл, *char* сам кастуется к *int*. Далее вызывается метод *qsort* из *stdlib*, который принимает сам массив, кол-во элементов (*size*), размер одного элемента (соответствует размеру первого элемента) и функцию для сортировки *int cmpfunc (const void* pa, const void* pb),* которая реализована достаточно просто: если значения суммы кодов разные, сравниваются они; иначе сравниваются индексы. Полученный в результате выполнения *qsort* массив содержит необходимый порядок элементов, который используется для создания обновлённого текста. Функция возвращает отсортированный текст.

2.8. Вывод текста

Как было сказано выше, вывод текста и последующее освобождение памяти происходит в функции *solve*. Так как вводятся и обрабатываются только ASCII символы, то вывод каждого предложения текста осуществляется через обычный *printf*.

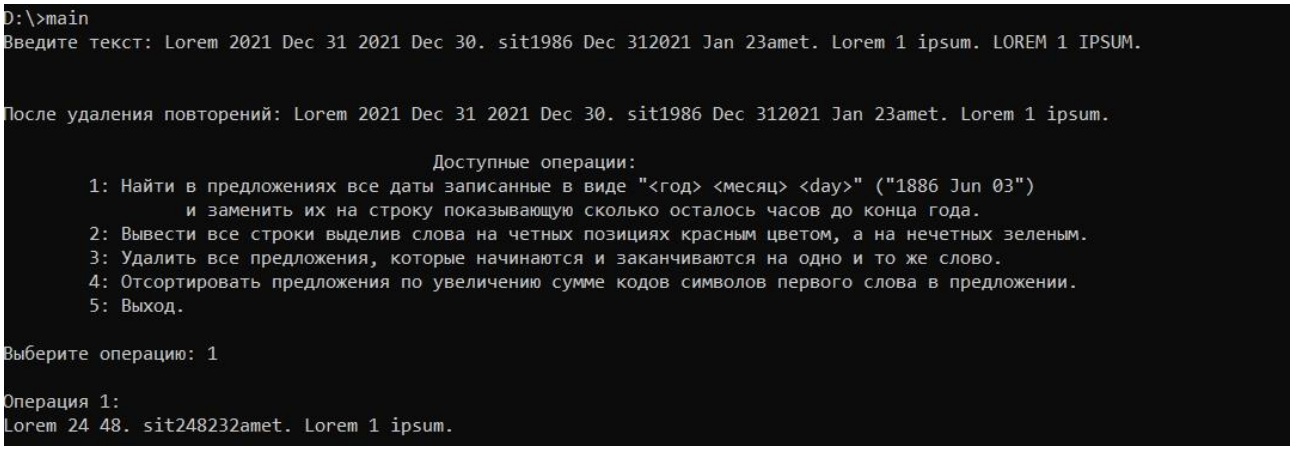
ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было создано консольное предложение для обработки текста согласно запросам пользователя. Все задачи также были успешно выполнены: программа может удалять повторяющиеся предложения; находить в предложениях все даты записанные в виде “<год> <месяц> <день>” и заменять их на строку, показывающую сколько осталось часов до конца года; выводить все строки, выделяя слова на четных позициях красным цветом, а на нечетных зеленым; удалять все предложения, которые начинаются и заканчиваются на одно и то же слово; отсортировывать предложения по увеличению сумме кодов символов первого слова в предложении. Можно сделать вывод о соответствии полученного результата поставленной цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт (онлайн-справочник) www.c-cpp.ru

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ



Пример 1 – удаление повторяющихся предложений в начале (удаляется последнее предложение), контекстное меню, первая операция

Пример 2 – вторая операция

```
D:\>main
Введите текст: Lorem ipsum. Lorem ipsum lorem. dolor sit dolor. ut enim UT. Incididunt.


После удаления повторений: Lorem ipsum. Lorem ipsum lorem. dolor sit dolor. ut enim UT. Incididunt.

                                    Доступные операции:
        1: Найти в предложениях все даты записанные в виде "<год> <месяц> <day>" ("1886 Jun 03")
                и заменить их на строку показывающую сколько осталось часов до конца года.
        2: Вывести все строки выделив слова на четных позициях красным цветом, а на нечетных зеленым.
        3: Удалить все предложения, которые начинаются и заканчиваются на одно и то же слово.
        4: Отсортировать предложения по увеличению сумме кодов символов первого слова в предложении.
        5: Выход.

Выберите операцию: 3

Операция 3:
Lorem ipsum.
```

Пример 3 – третья операция

```
D:\>main
Введите текст: Lorem ipsum. Lorem ipsum lorem. dolor sit dolor. ut enim UT. Incididunt.


После удаления повторений: Lorem ipsum. Lorem ipsum lorem. dolor sit dolor. ut enim UT. Incididunt.

                                    Доступные операции:
        1: Найти в предложениях все даты записанные в виде "<год> <месяц> <day>" ("1886 Jun 03")
                и заменить их на строку показывающую сколько осталось часов до конца года.
        2: Вывести все строки выделив слова на четных позициях красным цветом, а на нечетных зеленым.
        3: Удалить все предложения, которые начинаются и заканчиваются на одно и то же слово.
        4: Отсортировать предложения по увеличению сумме кодов символов первого слова в предложении.
        5: Выход.

Выберите операцию: 4

Операция 4:
ut enim UT. Lorem ipsum. Lorem ipsum lorem. dolor sit dolor. Incididunt.
```

Пример 4 – четвёртая операция (суммы кодов первых слов до операции: 511, 511, 544, 233, 1035)

```
Выберите операцию: 5
Выход
```

Пример 5 – выход

```
Выберите операцию: 6
Ошибка: номер операции может быть только от 1 до 5
```

Пример 6 – неверная операция

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <locale.h>
#include <math.h>

#define RED "\033[0;101;37m"
#define GREEN "\033[0;102;37m"
#define NONE  "\033[0;m"

char* getSentence(){
    int sentenceSize = 40;
    char* sentence = malloc(sentenceSize);
    char sym; int count = 0;
    while(1){
        sym = getchar();
        sentence[count++] = sym;
        if(count == sentenceSize){ sentenceSize += 15; sentence =
realloc(sentence, sentenceSize); }
        if(sym == '.') break;
    }
    sentence[count] = '\0';
    return sentence;
}

char** getText(int* size){
    int textSize = 20;
    char** text = calloc(textSize, sizeof(char*));
    char* sentence; int count = 0;
    while(1){
        sentence = getSentence();
        text[count++] = sentence;
        if(getchar() == '\n') break;
        if(count == textSize){ textSize += 10; text = realloc(text, textSize *
sizeof(char*)); }
    }
    *size = count; return text;
}

char* firstWord(char* sentence){
    int wordSize = 20;
    char* word = malloc(wordSize);
    char sym; int count = 0;
    while(1){
        sym = sentence[count];
        if (sym == ' ' || sym == ',' || sym == '.'){
            word[count] = '\0';
            break;
```

```
        }
        else word[count++] = sym;
        if(count == wordSize){ wordSize += 30; word = realloc(word, wordSize);
}
    }
    return word;
}

char* lastWord(char* sentence){
    int size = -1;
    while(sentence[size+1] != '.') size++;

    short wordSize = 30;
    char* word = malloc(wordSize);
    char sym; short count = 0;
    while(1){
        sym = sentence[size-count];
        if (sym == ' ' || sym == ','){
            word[count] = '\0';
            break;
        }
        else word[count++] = sym;
        if(size-count == -1){
            word[count] = '\0';
            break;
        }
        if(count == wordSize){
            wordSize += 30;
            word = realloc(word, wordSize);
        }
    }
    strrev(word);
    return word;
}

char** removeDuplicate(char** text, int* size){
    char** updatedText = calloc(*size, sizeof(char*));
    int count = 0; short areSame;
    for(int i = 0; i < *size; i++){
        short isDuplicate = 0;
        for(int j = 0; j < i; j++){
            if(strlen(text[i]) == strlen(text[j])){
                areSame = 1;
                for(int n = 0; n < strlen(text[i]); n++){
                    if(toupper(text[i][n]) != toupper(text[j][n])){
                        areSame = 0; break;
                    }
                }
                if(areSame){
                    isDuplicate = 1; break;
                }
            }
        }
        if(!isDuplicate) updatedText[count++] = text[i];
    }
    *size = count; return updatedText;
}
```

```
char** changeDate(char** text, int* size){
    char** updatedText = calloc(*size, sizeof(char*));
    char* updatedSentence; char sym; int count;
    for(int i = 0; i < *size; i++) {
        updatedSentence = malloc(sizeof(text[i])); count = 0;
        for(int j = 0;;j++){
            sym = text[i][j];
            if(isdigit(sym)){
                char* data = malloc(11);
                strncpy(data, &text[i][j], 11);

                int year, day; char month[4];
                if((sscanf(data, "%4d %3s %2d", &year, month, &day)) == 3){
                    month[3] = '\0';
                    short month_i =  !strcmp(month, "Jan") ? 0 : !strcmp(month,
"Feb") ? 1 : !strcmp(month, "Mar") ? 2 : !strcmp(month, "Apr") ? 3 :
!strcmp(month, "May") ? 4 : !strcmp(month, "Jun") ? 5 : !strcmp(month, "Jul") ?
6 : !strcmp(month, "Aug") ? 7 : !strcmp(month, "Sep") ? 8 : !strcmp(month,
"Oct") ? 9 : !strcmp(month, "Nov") ? 10 : 11;

                    struct tm time1 = {0};          struct tm time2 = {0};
                    time1.tm_year = year-1900;      time2.tm_year = year-1899;
                    time1.tm_mon = month_i;
                    time1.tm_mday = day;            time2.tm_mday = 1;
                    time_t t1 = mktime(&time1);     time_t t2 = mktime(&time2);

                    int diff = (int) difftime(t2, t1) / 3600;
                    short numCount = diff > 0 ? floor(log10(diff)) + 1 : 1;
                    char diff_s[numCount];
                    itoa(diff, diff_s, 10);

                    diff_s[numCount] = '\0';
                    updatedSentence[count] = '\0';
                    strcat(updatedSentence, diff_s);

                    count += numCount;
                    j+=10; continue;
                }
                 free(data);
            }
            updatedSentence[count++] = sym;
            if(sym == '.') {
                updatedSentence[count] = '\0';
                break;
            }
        }
        updatedText[i] = updatedSentence;
    }
    return updatedText;
}

char** colorize(char** text, int* size){
    char** updatedText = calloc(*size, sizeof(char*));
    int sentenceSize; char* updatedSentence;
    int count; char sym; short isOdd; short noWord;
```

```
    isOdd = 1;
    for(int i = 0; i < *size; i++) {
        sentenceSize = 30;
        updatedSentence = malloc(sentenceSize);
        count = 0; noWord = 1;
        for(int j = 0;;j++){
            sym = text[i][j];
            if ((sym == ' ' || sym == ',' || sym == '.')){
                if(!noWord){
                    updatedSentence[count] = '\0';
                    strcat(updatedSentence, NONE);
                    count += strlen(NONE);
                    noWord = 1;
                    isOdd = !isOdd;
                }
            }else if(noWord){
                noWord = 0;
                updatedSentence[count] = '\0';
                strcat(updatedSentence, isOdd ? GREEN : RED);
                count += strlen(GREEN);
            }
            updatedSentence[count++] = sym;

            if(count >= sentenceSize - 20){ sentenceSize += 30; updatedSentence
= realloc(updatedSentence, sentenceSize); }
            if(sym == '.') {
                updatedSentence[count] = '\0';
                break;
            }
        }
        updatedText[i] = updatedSentence;
    }
    return updatedText;
}

char** removeSimilarities(char** text, int* size){
    char** updatedText = calloc(*size, sizeof(char*));
    int count = 0;
    for(int i = 0; i < *size; i++) {
        char* first = firstWord(text[i]);
        char* last = lastWord(text[i]);
        for(int j = 0; j < strlen(first); j++) first[j] = tolower(first[j]);
        for(int j = 0; j < strlen(last); j++) last[j] = tolower(last[j]);

        if(strcmp(first, last)) updatedText[count++] = text[i];
      free(first); free(last);
    }
    *size = count; return updatedText;
}

int cmpfunc (const void* pa, const void* pb) {
    int* arr1 = (int*)pa;
    int* arr2 = (int*)pb;
    int diff1 = arr1[0] - arr2[0];
    if (diff1) return diff1;
    return arr1[1] - arr2[1];
}
```

```
char** sort(char** text, int* size){
    int array[*size][2]; int sum; char* word;
    for(int i = 0; i < *size; i++){
        sum = 0; word = firstWord(text[i]);
        for(int j = 0; j < strlen(word); j++) sum += word[j];
        array[i][0] = sum; array[i][1] = i;
    }
    qsort(array, *size, sizeof(array[0]), cmpfunc);

    char** updatedText = calloc(*size, sizeof(char*));
    for(int i = 0; i < *size; i++){
        updatedText[i] = text[array[i][1]];
    }
    return updatedText;
}

void solve(short operation, char*** text, int size, char** (*f)(char**, int*)){
    wprintf(L"\nОперация %hd:\n", operation);
    int newSize = size;
    char** temp = f(*text, &newSize);
    for(int i = 0; i < newSize; i++) {
        printf("%s ", temp[i]);
        free(temp[i]);
    }
    for(int i = 0; i < size; i++) free((*text)[i]);
    free(temp); free(*text);
    printf("\n");
}

void showMenu(){
    wprintf(L"\n\t\t\t\t\tДоступные операции:\n");
    wprintf(L"\t1: Найти в предложениях все даты записанные в виде “<год>
<месяц> <day>” (“1886 Jun 03”)\n\t\t и заменить их на строку показывающую
сколько осталось часов до конца года.\n");
    wprintf(L"\t2: Вывести все строки выделив слова на четных позициях красным
цветом, а на нечетных зеленым.\n");
    wprintf(L"\t3: Удалить все предложения, которые начинаются и заканчиваются
на одно и то же слово.\n");
    wprintf(L"\t4: Отсортировать предложения по увеличению сумме кодов символов
первого слова в предложении.\n");
    wprintf(L"\t5: Выход.\n\n");
}

int main(){
    setlocale(LC_ALL, "");
    wprintf(L"Введите текст: ");
    int size; char** text;

    text = getText(&size);
    char** tmp = removeDuplicate(text, &size);
    free(text); text = tmp;

    wprintf(L"\n\nПосле удаления повторений: ");
    for(int i = 0; i < size; i++) printf("%s ", text[i]);
    printf("\n");
```

```
    showMenu();
    wprintf(L"Выберите операцию: ");
    short operation; scanf("%hd", &operation);

    switch (operation)
    {
        case 1: solve(operation, &text, size, changeDate); break;
        case 2: solve(operation, &text, size, colorize); break;
        case 3: solve(operation, &text, size, removeSimilarities); break;
        case 4: solve(operation, &text, size, sort); break;
        case 5: wprintf(L"Выход\n"); break;
        default: wprintf(L"Ошибка: номер операции может быть только от 1 до
5\n"); break;
    }
    return 0;
}
```