Национальный исследовательский ядерный университет
«МИФИ» Институт интеллектуальных кибернетических
систем

Кафедра №12 «Компьютерные системы и технологии»

# ОТЧЕТ

## О выполнении лабораторной работы №4

## «Работа со строками»

**Студент:** Титов Иван Андреевич.

**Преподаватель:** Уваров М.П.

**Группа:** Б23-901

Москва – г .2023

# I.Подготовительная часть

## Вариант №34

### Введение

Из входного потока вводится произвольное количество строк произвольной длины. Каждая строка в общем случае содержит одно или более слов, разделенных пробелами и/или знаками табуляции. Завершение ввода определяется концом файла. Для каждой входной строки формируется новая выходная строка, в которую помещается результат. В полученной строке слова разделяются только одним пробелом, пробелов в её начале и в конце быть не должно. Введённая и сформированная строки выводятся на экран в двойных кавычках.

В ходе выполнения лабораторной работы должны быть разработаны:

1. Программа, использующая функцию `readline()` из состава библиотеки GNU readline для ввода строк и функции стандартной библиотеки для их обработки (`<string.h>`).
2. Программа, идентичная п. 1, за исключением того, что все библиотечные функции заменены на собственную реализацию данных функций, представленную в отдельных файлах (например: `mystring.h`, `mystring.c`).

Отчётность по выполнению лабораторной работы должна включать:

1. Блок-схему алгоритма работы основной программы.
2. Блок-схемы алгоритмов работы функций по обработке строк.
3. Исходные коды всех программ.
4. Тестовые наборы для программ п. 1 и п. 2.
5. Сравнительный анализ времени, потраченного на решение задачи программами п. 1 и п. 2 (на конкретных примерах).

Примечания:

1. Каждая строка представлена на физическом уровне вектором.
2. Использование массивов переменной длины (VLA — variable length arrays) не допускается.
3. Ввод строк в п. 2 должен быть организован с помощью функции `scanf()` со спецификациями для ввода строк. Использование функций семейства `gets()`, `getchar()`, а также спецификаций `%c` и `%m` в `scanf()` не допускается.
4. Целочисленные и строковые константы, используемые в формулировках индивидуальных заданий, должны быть заданы в исходном коде с помощью директив препроцессора `#define`.
5. Программа должна корректным образом завершаться при обнаружении `EOF` — конца файла (в UNIX-подобных ОС инициируется нажатием клавиш `Ctrl + D`, в Windows — `Ctrl + Z`).
6. Логически законченные части алгоритма решения задачи должны быть оформлены в виде отдельных функций с параметрами.
7. Исходные коды программ должны быть логичным образом разбиты на несколько файлов (необходимо использовать как `*.c`-файлы, так и `*.h`-файлы).
8. Использование глобальных переменных не допускается.
9. Программы должна корректным образом работать с памятью. Для проверки необходимо использовать соответствующие программные средства, например, `valgrind` (при тестировании и отладке программ п. 1 и п. 2 необходимо запускать их командой вида `valgrind ./lab4`, а при анализе производительности — `./lab4`).

### Индивидуальное задание

Упорядочить слова в строке по алфавиту.

**Цель**: создать программу, удовлетворяющую требованиям

**Требования к программе**:

1. Требования к структуризации программ:

А) Организовать 2 программы: 1 программа при работе со строками использует функции из библиотеки «string.h», 2 программа при работе со строками использует авторские функции (реализации аналогов из библиотеки «string.h»)

Б) Организовать из логически законченных частей программ функции

В) Организовать заголовочный файл, в котором разместить прототипы функций, макросы, подключение необходимых библиотек

2. Требования к входным данным:

На вход подаются строки (считываемые функцией readline() или её аналогом)

3. Требования к выходным данным:

В терминал последовательно (в порядке их ввода) выводятся строки, заключённые в двойные кавычки без пробелов в начале и конце каждой строки.

4. Требования к обработке:

А) Введённые строки разбиваются на слова (лексемы/токены) функцией (strtok() или её аналогом), полученные слова сортируются в алфавитном порядке, формируются выходные строки из слов в полученном порядке. Все действия осуществляются в рамках массива строк.

Б) Время выполнения двух вариантов программ подсчитываются и сравниваются

## II.Алгоритмическая часть

**Состав программы 1:**

1. Lab4.c – основной файл, содержащий функцию main. Создаёт массив строк, осуществляет последовательный ввод строк, пока на вход не подан EOF.

   Примечания: для подсчёта время работы программы при вызове и обработке строки при их последовательном вводе создаётся переменная begin (время начала) и после конца обработки строки (и занесения её в массив) переменная end записывает время конца. Разность end и begin является временем обработки одной строки. Эта разность в каждой итерации ввода и обработки прибавляется к переменной sum (время работы программы).

   Для адекватного использования памяти начальное количество строк в массиве устанавливается в 10 строк. Размер массива динамически меняется с помощью операции realloc(), благодаря чему массив не будет переполняться.
2. Sort.c – сортировка массива слов (полученных из разбитой строки) методом выбора. Метод сравнения – strcmp(). Таким образом, сортировка происходит в алфавитном порядке
3. Divandsort.c – непосредственное разбиение строки на слова (токены/лексемы) методом strtok(). Полученные слова записываются в массив строк words, который затем сортируется функцией Sort(). Создаётся новая строка, в которую методом strcat() добавляются (присоединяются) слова из words.
4. Out.c – последовательный вывод в терминал массива строк с двойными кавычками

**Состав программы 2:**

1. Изменённые (с вызовами авторских реализаций функций из библиотеки <string.h>) файлы: lab42.c, Divandsort2.c, Sort2.c + Out.c
2. Oureadline.c – функция, аналогичная readline(). Инициализирует массив (который в дальнейшем должен быть очищен) длиной 100, считывает из терминала строку длиной не более 99 (1 символ зарезервирован под терминальный ноль) методом scanf() со спецификатором [^\n], очищает stdin функций __fpurge() для адекватной работы программы с несколькими вызовами oureadline().
3. Ourstrtok.c - функция, аналогичная strtok(). Механизм работы функции подобно описан в комментариях в приведённом далее коде функции
4. Ourstrdup.c - функция, аналогичная strdup(). Инициализирует выходной массив размера входного массива. Пробегается по нему и присваивает ячейкам выходного массива значения входного. Добавляет к выходному массиву терминальный ноль.

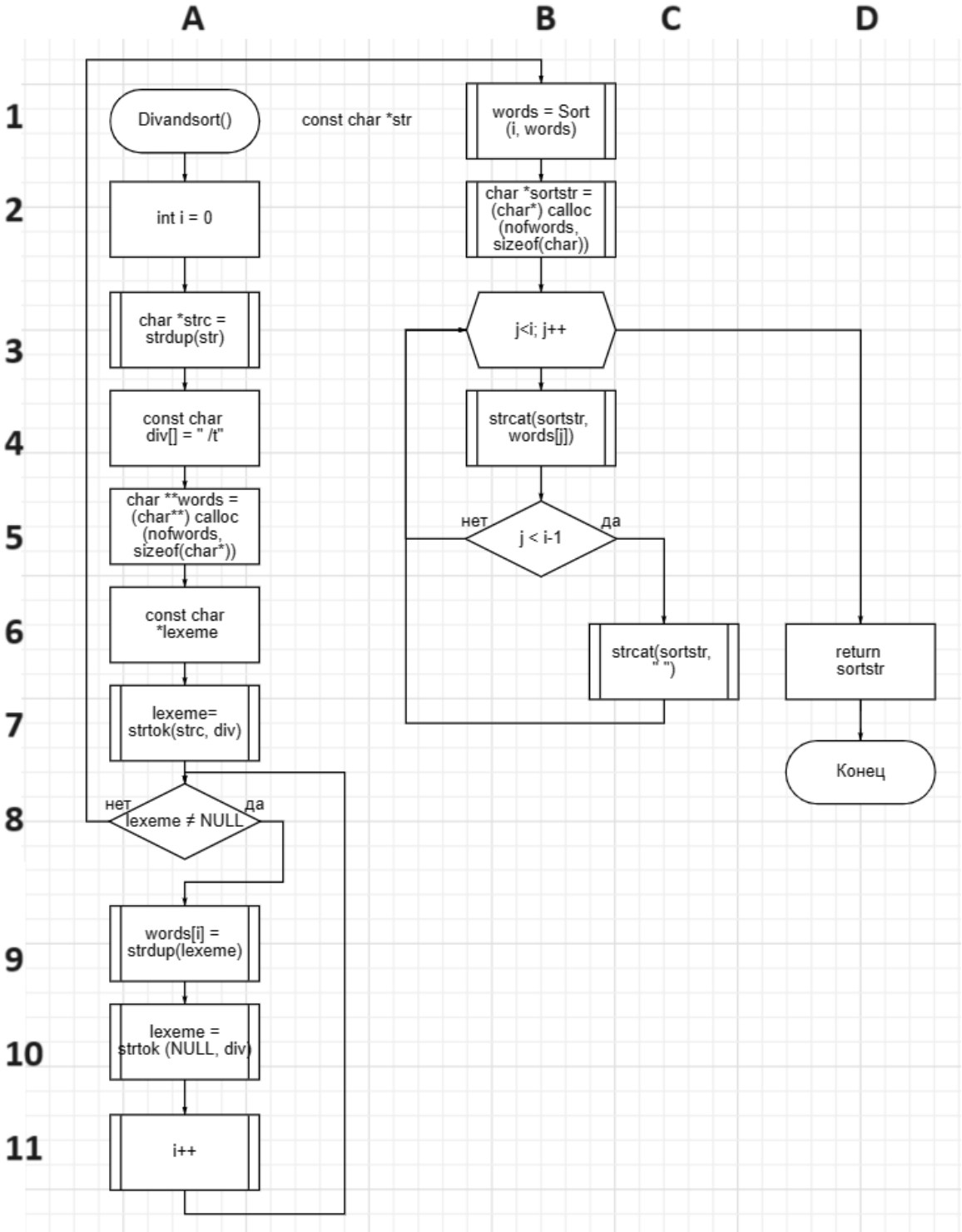
5. Ourstrcat.c - функция, аналогичная strcat(). Склеивает две строки, возвращает одну входную строку. Угрозы переполнения первой строки нет, т.к. в функции Divandsort2, использующей эту функцию, первый входной массив инициализируется всегда с учётом этой склейки.
6. Ourstrcmp.c – функция, аналогичная strcmp(). Возвращает разность между двумя первыми расходящимися символами в строке.

В обеих программах есть заголовочные файлы.

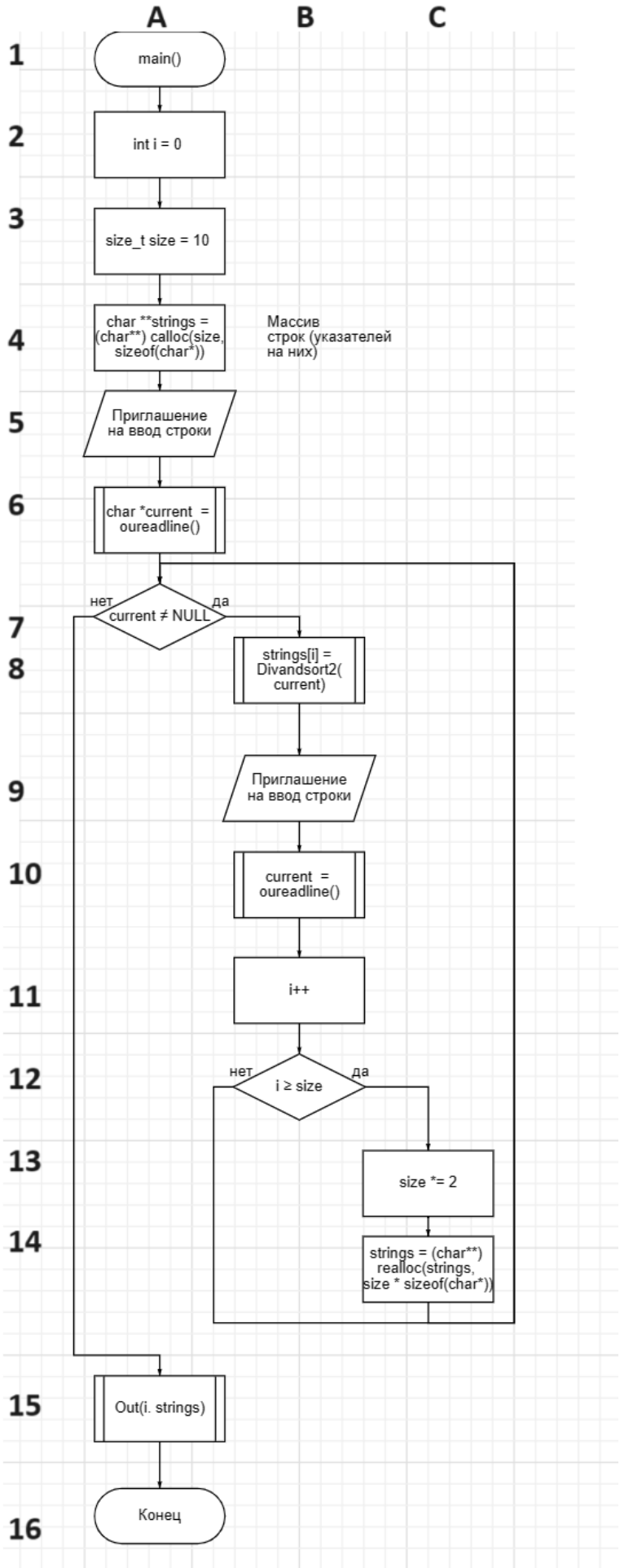
**Блок-схемы:**

1. Алгоритм работы основной программы (первой)

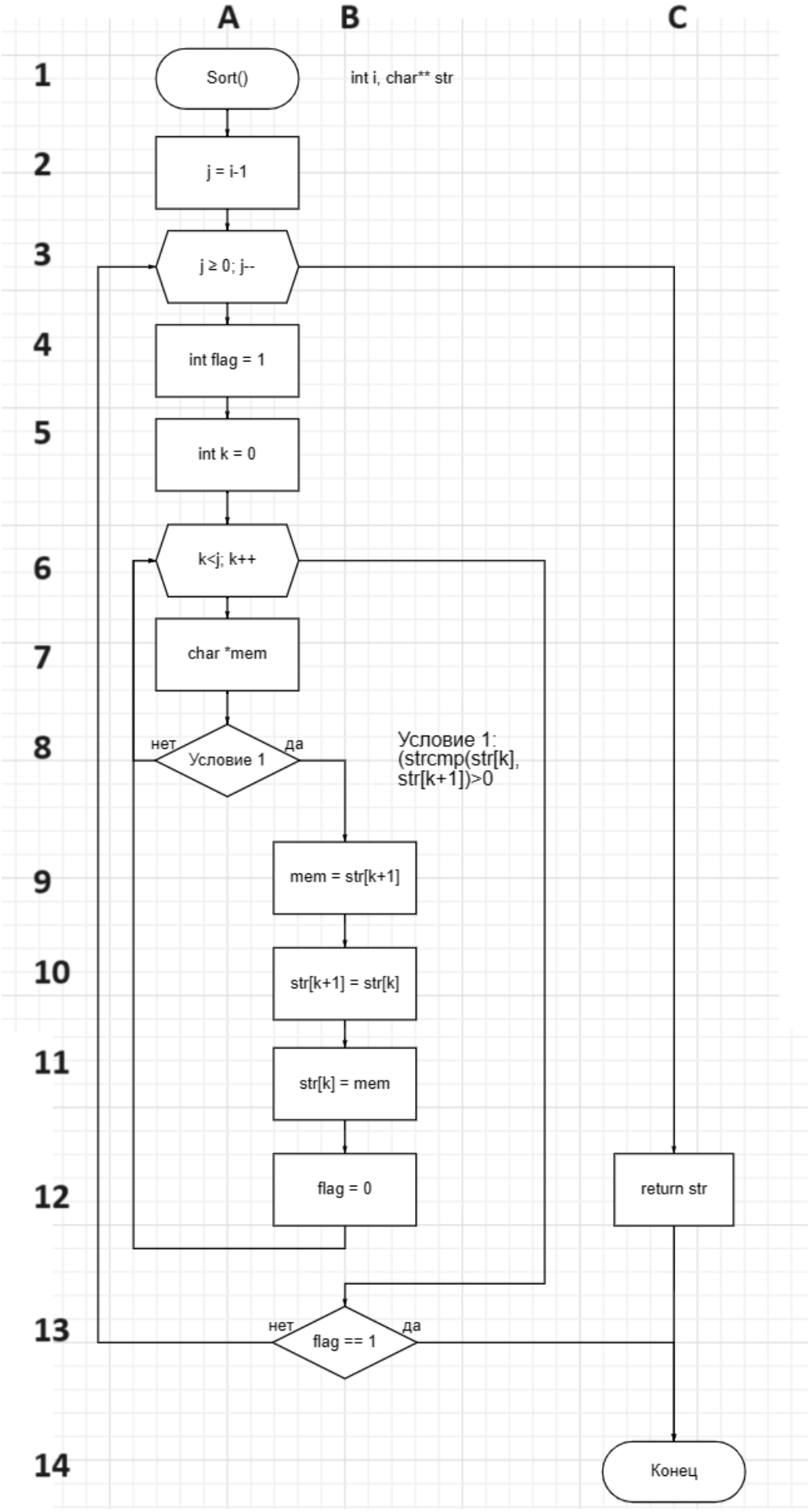
A) Divandsort(const char *str)



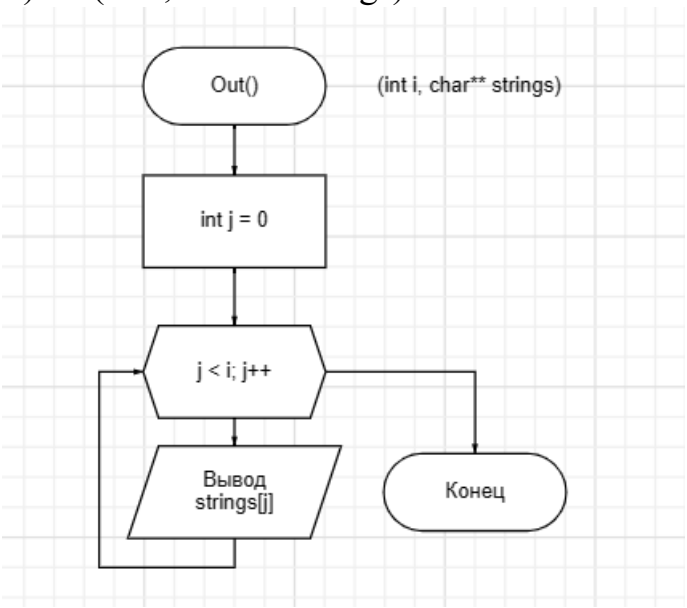
Б)lab4.c


A
B
C
1
main()
2
int i = 0
3
size_t size = 10
4
char **strings = (char**) calloc(size, sizeof(char*))
Массив строк (указателей на них)
5
Приглашение на ввод строки
6
char *current = oureadline()
нет
current ≠ NULL
да
7
8
strings[i] = Divandsort2( current)
9
Приглашение на ввод строки
10
current = oureadline()
11
i++
12
нет
i ≥ size
да
13
size *= 2
14
strings = (char**) realloc(strings, size * sizeof(char*))
15
Out(i. strings)
16
Конец

B)Sort(int i, char** str)

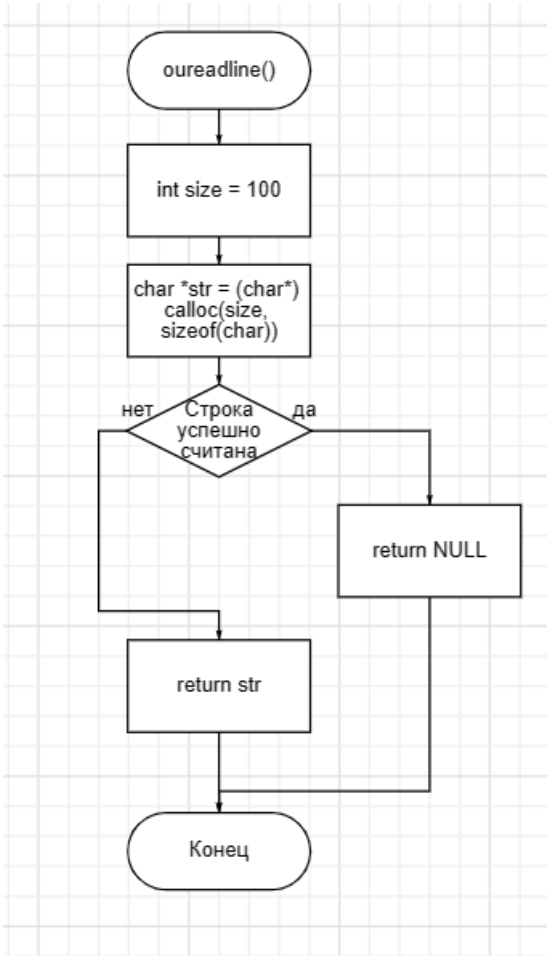


Г)Out(int i, char** strings)

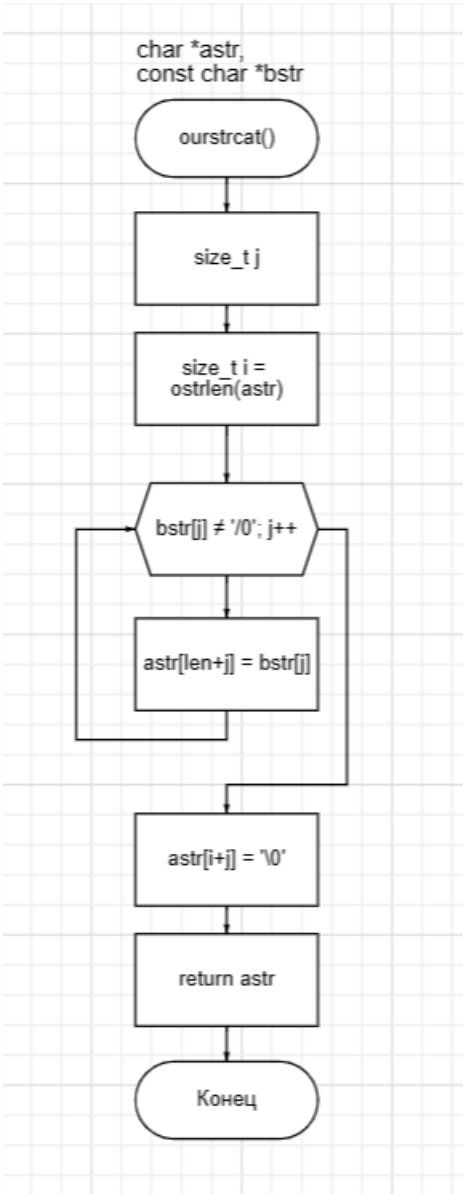



2. Алгоритмы функций по обработке строк

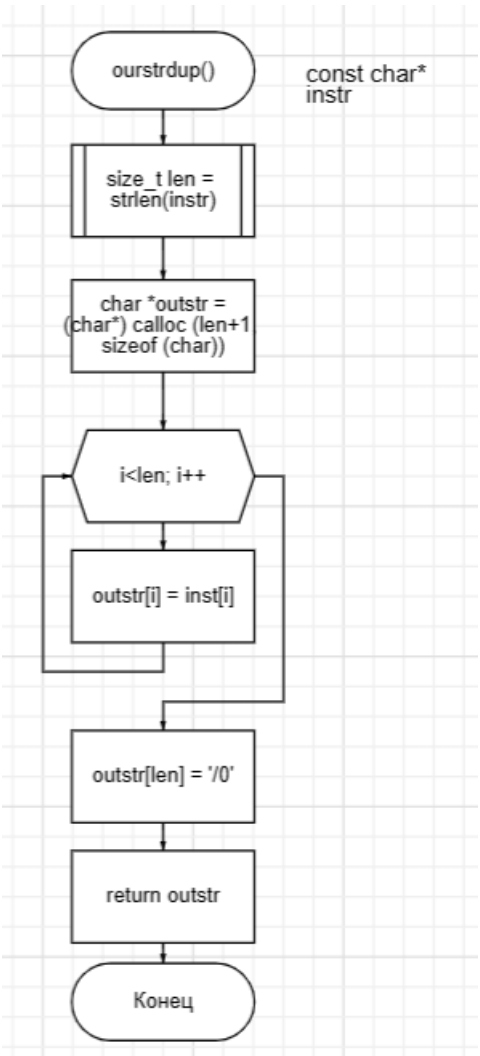
А)oureadline()



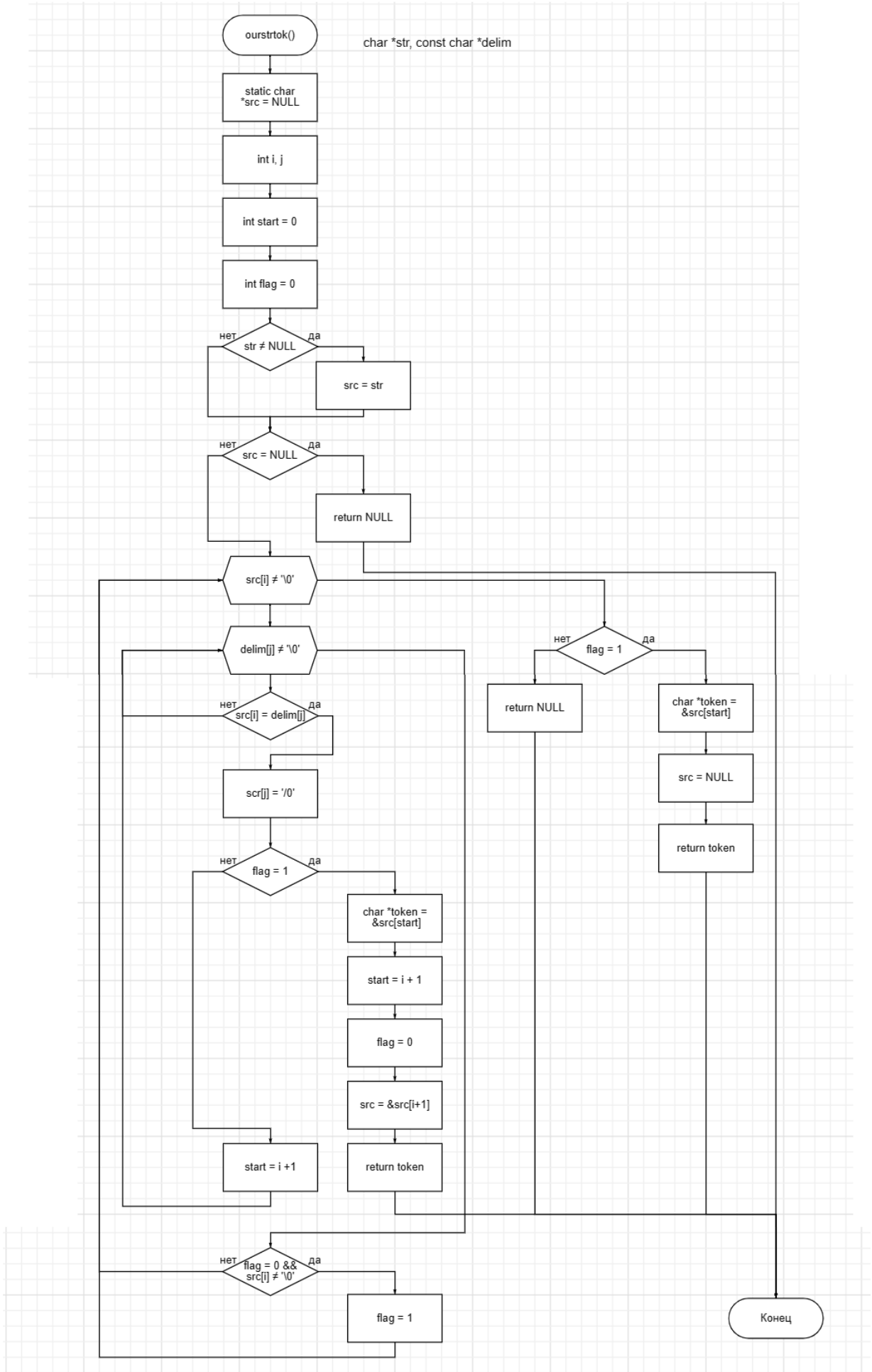
Б)ourstrcat(char *astr, const char *bstr)



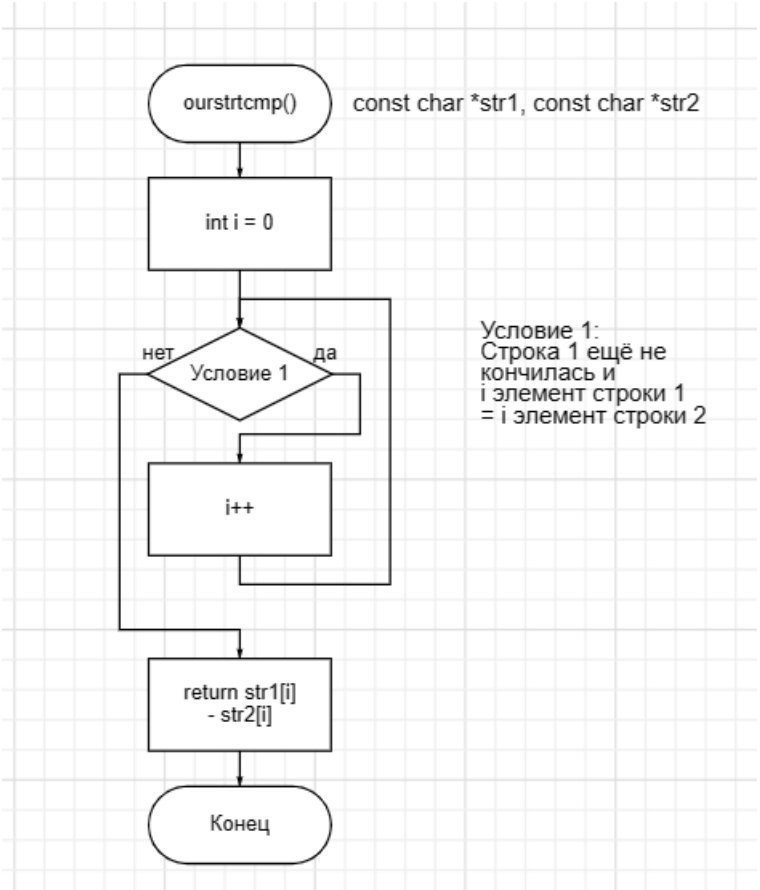

В)ourstrdup(const char* instr)



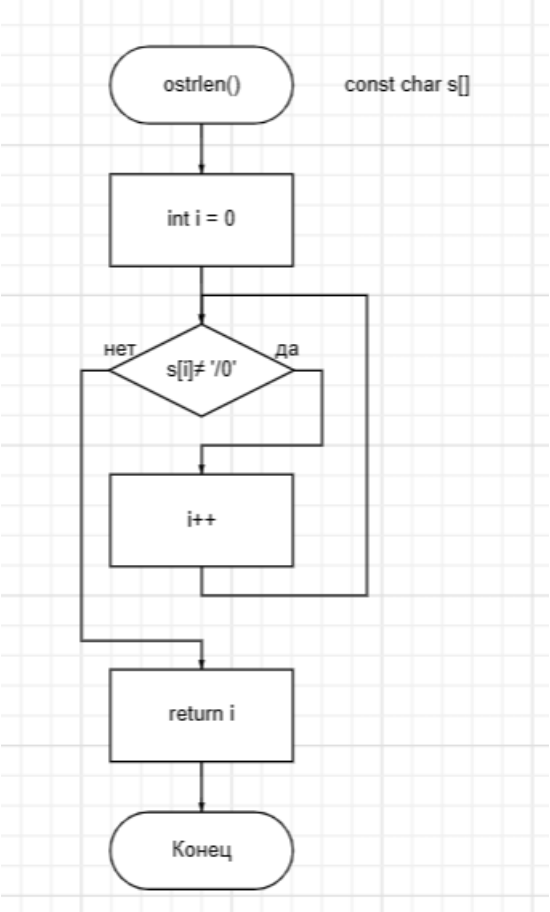
Г)ourstrtok(char *str, const char *delim)



Д)ourstrcmp(const char* str1, const char *str2)




E) ostrlen(const char s[])



## III.Код

1. Первая программа:

A) Divandsort(const char *str)

```
#include "lab4H.h"
#define nofwords 100
char* Divandsort(const char *str){
        int i = 0;
        char *strc = strdup(str);
        const char div[] = " \t";
        char **words = (char**) calloc(nofwords, sizeof(char*));
        const char *lexeme;
        lexeme = strtok(strc, div);
        while (lexeme != NULL){
                words[i] = strdup(lexeme);
                lexeme = strtok (NULL, div);
                i++;
        }
        words = Sort(i, words);
        char *sortstr = (char*) calloc(nofwords, sizeof(char));
        for(int j = 0; j < i; j++){
                strcat(sortstr, words[j]);
                if(j < i-1){
                        strcat(sortstr, " ");
                }
        }
        for(int j = 0; j < i; j++){
                free(words[j]);
        }
        free(words); free(strc);
        return sortstr;
}
```

Б) lab4.c

```
#include "lab4H.h"
#define PROMPT "> "
#include <stdio.h>
#include <time.h>
int main() {
        double sumtime = 0;
        int i = 0; size_t size = 10;
        char **strings = (char**) calloc(size, sizeof(char*));
        printf("Введите строку\n");
        char *current = readline(PROMPT);
        while (current != NULL){
                clock_t begin = clock();
                strings[i] = Divandsort(current);
                free(current);
                printf("Введите строку\n");
                current = readline(PROMPT);
                i++;
                if (i >= size) {
                        size *= 2;
                        strings = (char**) realloc(strings, size *
sizeof(char*));
                }
                clock_t end = clock();
                sumtime += (double)(end - begin)/CLOCKS_PER_SEC;
        }
        free(current);
        Out(i, strings);
```

```
        for(int j = 0; j < i; j++){
                free(strings[j]);
        }
        free(strings);
        printf("Время выполнения программы: %0.11f\n", sumtime);
}
```

В) Sort(int i, char** str)

```
#include "lab4H.h"
char** Sort(int i, char** str){
    for (int j = i-1; j>=0; j--){
        int flag = 1;
        for (int k = 0; k<j; k++){
            char *mem;
            if (strcmp(str[k], str[k+1]) > 0){
                mem = str[k+1];
                str[k+1] = str[k];
                str[k] = mem;
                flag = 0;
            }
        }
        if (flag == 1) break;
    }
    return str;
}
```

Г)Out(int i, char** strings)

```
#include "lab4H.h"
void Out(int i, char** strings){
        for (int j = 0; j < i; j ++){
                printf("\"%s\"\n", strings[j]);
        }
}
```

2. Функции по обработке строк

А)oureadline()

```
#include "lab42H.h"
#include <stdio_ext.h>
char* oureadline(const char* Prompt) {
        size_t size = 100;
        char *str = (char*) calloc(size, sizeof(char));
        if (Prompt != NULL){
                printf("%s", Prompt);
        }
        if (scanf("%99[^\n]", str) <0){
                free(str);
                return NULL;
        }
        __fpurge(stdin);
        return str;
}
```

Б) ourstrcat(char *astr, const char *bstr)

```
#include "lab42H.h"
char *ourstrcat(char *astr, const char *bstr) {
    size_t j;
    size_t i = ostrlen(astr);
    for (j = 0; bstr[j] != '\0'; j++) astr[i+j] = bstr[j];
    astr[i+j] = '\0';
    return astr;
}
```

В)ourstrdup(const char* instr)

```
#include "lab42H.h"
char* ourstrdup(const char* instr){
        size_t len = ostrlen(instr);
        char *outstr = (char*) calloc ((len+1), sizeof (char));
        for (int i = 0; i < len; i++){
                outstr[i] = instr[i];
        }
        outstr[len] = '\0';
        return outstr;
}
```

Г)ourstrtok(char *str, const char *delim)

```
#include "lab42H.h"
char *ourstrtok(char *str, const char *delim) {
    static char *src = NULL; // указатель на начало следующего токена
    int i, j;
    int start = 0; // индекс начала текущего токена
    int flag = 0; // флаг указывающий начался ли новый токен

    if(str != NULL) src = str; // первый вызов функции
    if(src == NULL) return NULL; // если все токены были извлечены,
возвращаем NULL

    for(i = 0; src[i] != '\0'; i++) {
        for(j = 0; delim[j] != '\0'; j++) {
             if(src[i] == delim[j]) {
                src[i] = '\0'; // разделитель -> терминальный нуль
                if(flag == 1) { // если новый токен уже начался
                    char *token = &src[start]; // сохраняем текущий токен
                    start = i + 1;
                    flag = 0;
                    src = &src[start]; // обновляем src, чтобы указывать на
начало следующего токена
                    return token;
                }
                start = i + 1; // обновляем индекс начала токена
            }
        }
        if(flag == 0 && src[i] != '\0') flag = 1; // если новый токен еще не
начался и сейчас не разделитель
    }
    if(flag == 1) { //конец строки достигнут
        char *token = &src[start]; // сохраняем текущий(последний) токен
        src = NULL; // все токены считаны
        return token; // возвращаем последний токен
    }
    else return NULL;//нет ни одного токена
}
```

Д) ourstrcmp(const char* str1, const char *str2)

```
#include "lab42H.h"
int ourstrcmp(const char *str1, const char *str2) {
        int i = 0;
        while(str1[i] && (str1[i] == str2[i])) {
                i++;
        }
        return str1[i] - str2[i];
}
```

## IV.Тестирование

```
$ gcc -o program2 lab4.c Sort.c Divandsort.c Out.c -lreadline
$ gcc -o program1 lab42.c Sort2.c Divandsort2.c Out.c oureadline.c ourstrdup.c ourstrtok.c ourstrcat.c ourstrcmp.c
```

1. Ввод: «c d a»

Первая программа (свои строки):

```
Введите строку
> c d a
Введите строку
>
Ваши строки:
"a c d"
Время выполнения операций: 0.00018100000
```

Вторая программа:

```
[titov.ia@unix lab4]$ ./program2
Введите строку
> c d a
Введите строку
>

Ваши строки:
"a c d"
Время выполнения программы: 0.00029200000
```

2. Ввод: «c d a», «man no cap test», «  fear i open joke  i»

Первая программа (свои строки):

```
Введите строку
> c d a
Введите строку
> man no cap test
Введите строку
>    fear i open joke    i
Введите строку
>
Ваши строки:
"a c d"

"cap man no test"

"fear i i joke open"
Время выполнения операций: 0.00047900000
```

Вторая программа:

```
Введите строку
> c d a
Введите строку
> man no cap test
Введите строку
>    fear i open joke    i
Введите строку
>

Ваши строки:
"a c d"

"cap man no test"

"fear i i joke open"
Время выполнения программы: 0.00146300000
```

3. Ввод: «I hate you», «harmony random words for sure», «wwwwwwwwwwwda sdweq we da fderefweasda weqw sda dqweqsad»

Первая программа (свои строки):

```
Введите строку
> I hate you
Введите строку
> harmony  random words       for     sure
Введите строку
> wwwwwwwwwwda sdweq we da fderefweasda weqw sda  dqweqsad
Введите строку
>
Ваши строки:
"I hate you"

"for harmony random sure words"

"da dqweqsad fderefweasda sda sdweq we weqw wwwwwwwwwwda"
Время выполнения операций: 0.00043700000
```

Вторая программа:

```
Введите строку
> I hate you
Введите строку
> harmony  random words       for     sure
Введите строку
> wwwwwwwwwwwda sdweq we da fderefweasda weqw sda  dqweqsad
Введите строку
>

Ваши строки:
"I hate you"

"for harmony random sure words"

"da dqweqsad fderefweasda sda sdweq we weqw wwwwwwwwwwda"
Время выполнения программы: 0.00180500000
```

4. Ввод: «Hello world»*1000

Первая программа (свои строки):


Вторая программа:

## V.Выводы

Работая над лабораторной работой №4, автор освоил работу со строками: осознал, что такое строка в языке С, научился их создавать, копировать, разбивать на лексемы, сравнивать и т.п.

В ходе анализа полученных при тестировании программ 1 и 2 был сделан вывод о том, что авторские реализации функций библиотеки <string.h> работают с переменным успехом (в смысле эффективности, а не работоспособности в целом). Так, при вводе небольших строк, авторские реализации работают быстрее, но когда дело доходит до более крупных объёмов информации, библиотечные функции быстрее в разы.

Помимо того, немаловажным является разработка алгоритма, т.к. в силу всё возрастающих размеров программ, сознаётся в полной мере необходимость первоначального планирования и некой базовой формализации мыслей для решения поставленных задач.

Большую роль в освоенном материале также стала сборка многофайлового проекта, разбиение проекта на заголовочный файл, файлы с исходным кодом.

Конечно, реализация библиотечных функций – дело нетривиальное, если к нему подходить всерьёз. Ведь оптимизация и совершенствование (в плане безопасности, быстродействия) могут занять существенное количество времени/усилий.