

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Обработка текстовых данных

Студент гр. 1304

Заика Т.П.

Преподаватель

Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Заика Т.П.

Группа 1304

Тема работы: Обработка текстовых данных

Исходные данные:

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

Содержание пояснительной записки:

«Содержание», «Введение», «Описание кода программы», «Описание Makefile», «Заключение», «Список использованных источников», «Приложение А: Исходный код программы», «Приложение Б: Тестирование программы»

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 15.10.2021

Дата сдачи реферата: 19.12.2021

Дата защиты реферата: 21.12.2021

Студент

Заика Т.П.

Преподаватель

Чайка К.В.

АННОТАЦИЯ

Курсовая работа представляет из себя программу, предполагающую работу с текстом. На вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой). В программе этот текст сохраняется в виде структуры Text, а составляющие его предложения в виде структуры Sentence. Программа удаляет повторяющиеся предложения и запрашивает у пользователя одно из четырех действий, осуществляющих работу с текстом. Код программы написан на языке C, запуск проводится в операционной системе Linux. Для ввода и вывода текста, удаления повторяющихся предложений, а также выполнения действий с текстом написаны функции, описанные в разделе «Описание кода программы». Исходный код программы и её тестирование представлены в Приложениях А и В соответственно.

СОДЕРЖАНИЕ

	Введение	5
1.	Описание кода программы	6
1.1.	Функции ввода, вывода текста, структуры, отчистка текста из памяти	6
1.2.	Функция удаления повторяющихся предложений	8
1.3.	Функция печати слов, которые встречаются в тексте не более одного раза	9
1.4.	Функция преобразования даты в строке к заданному формату	10
1.5.	Функция сортировки предложений по произведению длин слов	14
1.6.	Функция удаления предложений, содержащих символ «№» или «#», но не содержат ни одной цифры	16
1.7.	Главная функция	17
2.	Описание Makefile	19
	Заключение	20
	Список использованных источников	21
	Приложение А. Исходный код программы	22
	Приложение Б. Тестирование программы	35

ВВЕДЕНИЕ

Кратко описать цель работы, основные задачи и методы их решения.

Цель работы — написание программы для считывания текста и его обработки.

Задание:

Требуется считать текст, провести его отчистку от повторяющихся предложений, а далее по выбору пользователя произвести его обработку:

1. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид “<день> <месяц> <год> г.” заменить на подстроку вида “ДД/ММ/ГГГГ”.
3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ ‘#’ или ‘№’, но не содержат ни одной цифры.

Все функции должны быть реализованы с использованием стандартных библиотек языка C.

Требуется реализовать пользовательский выбор функционала программы, напечатать при этом подсказку, и предусмотреть возможность выхода из программы.

Также должен быть написан Makefile.

1. ОПИСАНИЕ КОДА ПРОГРАММЫ

1.1. Функции ввода, вывода текста, структуры, отчистка текста из памяти

Функции ввода текста:

```
struct Text readText(){
    int size = MEM_STEP;
    struct Sentence** text = malloc(size*sizeof(struct Sentence*));
    if(!text){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    int n = 0;
    struct Sentence* temp;
    int nlcount = 0;
    do{
        temp = readSentence();
        if(temp->str[0] == '\n'){
            nlcount++;
        } else {
            nlcount = 0;
            text[n] = temp;
            n++;
        }
    } while (nlcount<2);
    struct Text txt;
    txt.size = size;
    txt.sents = text;
    txt.n = n;
    return txt;
}

struct Sentence* readSentence(){
    int size = MEM_STEP;
    wchar_t *buf = malloc(size*sizeof(wchar_t));
    if(!buf){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    int n = 0;
    wchar_t temp;
    do{
        if (n >= size-2){
            wchar_t *t = realloc(buf, size+MEM_STEP);
            if(!t){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
            size += MEM_STEP;
            buf = t;
        }
    }
```

```

        temp = getwchar();
        buf[n] = temp;
        n++;
    } while (temp!='\n' && temp!='.' && temp!='!');
    buf[n] = '\0';
    struct Sentence *sentence = malloc(sizeof(struct Sentence));
    if(!sentence){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    sentence -> str = buf;
    sentence -> size = size;
    return sentence;
}

```

Функция readSentence() делает посимвольную запись из stdin в структуру Sentence, каждое новое предложение записывается после символа переноса строки (нажатии клавиши Enter) или точки или восклицательного знака. Функция readText() считывает в себя структуры Sentence, записывая таким образом текст по предложениям. Концом ввода текста считается два символа переноса строки (двойное нажатие клавиши Enter)

Функция вывода текста:

```

void print_text(struct Text *text){
    for (int i=0; i<text->n; i++){
        wprintf(L"%ls", text->sents[i]->str);
    }
}

```

Структуры:

```

struct Sentence{
    wchar_t *str;
    int size;
};

struct Text{
    struct Sentence **sents;
    int n;
    int size;
};

```

Функция отчистки текста из памяти:

```
void free_text(struct Text *text){
    int text_len = text->n;
    for(int i=0; i<text_len; i++){
        free(text->sents[i]);
    }
}
```

1.2. Функция удаления повторяющихся предложений

```
void changed_text(struct Text* text){
    int i = 0;
    int j = i + 1;

    while(i < text->n-1){
        while(j < text->n){
            if (wcscasecmp(text->sents[i]->str, text->sents[j]->str) == 0){
                free(text->sents[j]);
                memmove(&text->sents[j], &text->sents[j+1], (text->n-
j+1)*sizeof(struct Sentence));
                text->n -= 1;
            } else {
                j++;
            }
        }
        i++;
        j = i + 1;
    }
}
```

Функция сравнивает строковое значение указателя на структуру *Sentence*, записанной в указателе на структуру *Text*, со следующим указателем на структуру, также содержащим строковое значение, и при помощи функции *wcscasecmp* сравнивает их идентичность. Если строки равны, то текущая строка удаляется из памяти, а следующая, идентичная ей, записывается на ее место при помощи функции *memmove*.

1.3. Функция печати слов, которые встречаются в тексте не более одного раза

```
void single_word(struct Text *text){
    wchar_t sep[10] = L" ,!?:\\n";
    wchar_t **word_text = malloc(text->n*MEM_STEP*sizeof(wchar_t*));
    if(!word_text){ wprintf(L"%ls\\n", CRASH_MEMORY_MSG);}
    int count_elem = 0;

    for(int i=0; i<text->n; i++){
        wchar_t *cur_sent = text->sents[i]->str;
        wchar_t temp_sent[wcslen(cur_sent)];
        wcscpy(temp_sent, cur_sent);

        wchar_t *istr;
        wchar_t *pwc;
        istr = wcstok(temp_sent, sep, &pwc);
        while(istr != NULL){
            wchar_t *temp = malloc(sizeof(wchar_t*));
            if(!temp){ wprintf(L"%ls\\n", CRASH_MEMORY_MSG);}
            wcscpy(temp, istr);
            word_text[count_elem] = temp;
            count_elem++;
            istr = wcstok(NULL, sep, &pwc);
        }
    }

    for(int j=0; j<count_elem; j++){
        int flag = 0;
        wchar_t *cmp_item = word_text[j];
        for(int k=0; k<count_elem; k++){
            if (j == k){
                continue;
            }
            if(wcscmp(cmp_item, word_text[k]) == 0){
                flag = 0;
                break;
            } else {
                flag = 1;
            }
        }
        if(flag == 1){
```

```

        wprintf(L"%ls\n", cmp_item);
    }
}

for(int i=0; i<count_elem; i++){
    free(word_text[i]);
}
free(word_text);
}

```

На вход функция принимает указатель на структуру Text, используя находящиеся в ней указатели на структуру Sentence с строковым значением, для разделения каждого предложения на слова при помощи функции wstok и записи их в массив слов из текста. Далее в тексте производится сравнение каждого слова с каждым внутри массива слов с помощью функции wcsstr, и если слово не нашлось в массиве два раза, то функция выводит его в консоль.

1.4. Функция преобразования даты в строке к заданному формату

Сама функция:

```

void change_date(struct Text *text){
    wchar_t sep[10] = L" ,!?:\n";

    for(int i=0; i<text->n; i++){
        int cur_elem = 0;
        wchar_t **work_sent = malloc(MEM_STEP*sizeof(wchar_t*));
        if(!work_sent){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}

        wchar_t *cur_sent = text->sents[i]->str;
        wchar_t temp_sent[wcslen(cur_sent)];
        wcscpy(temp_sent, cur_sent);

        wchar_t *istr;
        wchar_t *pwc;
        istr = wstok(temp_sent, sep, &pwc);
        while(istr != NULL){
            wchar_t *temp = calloc(*istr, sizeof(wchar_t*));
            if(!temp){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
            wcscpy(temp, istr);
            work_sent[cur_elem] = temp;
            cur_elem++;
            istr = wstok(NULL, sep, &pwc);
        }

        wchar_t **temp_date = malloc(3*sizeof(wchar_t*));
        if(!temp_date){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    }
}

```

```

        for(int j=0; j<cur_elem-3; j++){
            if(wcsncmp(work_sent[j+3], L"y.") == 0 || wcsncmp(work_sent[j+3],
L"r.") == 0){
                wchar_t* date = check_date(work_sent[j]);
                if(date != NULL){
                    temp_date[0] = date;
                    wchar_t* num_mounts =
check_month(work_sent[j+1]);
                    if(num_mounts != NULL){
                        temp_date[1] = num_mounts;
                        if(work_sent[j+2][0] > L'0' &&
work_sent[j+2][0] <= L'9'){
                            temp_date[2] = work_sent[j+2];
                        } else {
                            break;
                        }
                    } else {
                        break;
                    }
                } else {
                    break;
                }
            }
        }
    }

```

```

    wchar_t *replace = malloc(30*sizeof(wchar_t*));
    if(!replace){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    swprintf(replace, 30, L"%ls/%ls/%ls", temp_date[0], temp_date[1],
temp_date[2]);

```

```

    int wchar_start = 0;
    int wchar_end = 0;

```

```

    if(temp_date[0] != NULL && temp_date[1] != NULL && temp_date[2]
!= NULL){
        for(int i=0; i<wcslen(cur_sent); i++){
            if(cur_sent[i] == temp_date[0][1] || cur_sent[i] ==
temp_date[0][0]){
                wchar_start = i;
                int cur_num_year = 0;
                for(int j=i+1; j<wcslen(cur_sent); j++){

```

```

        if(cur_sent[j] ==
temp_date[2][cur_num_year]){
            wchar_end = j;
            cur_num_year++;
        }
    }
    if(wchar_end == 0){
        wchar_start = 0;
    } else {
        break;
    }
}
}
}

memmove(cur_sent+wchar_start, replace, ((wchar_end-wchar_start) *
sizeof(wchar_t)));

free(temp_date);
free(replace);
for(int i=0; i<cur_elem; i++){
    free(work_sent[i]);
}
free(work_sent);
}
}

```

Вспомогательные функции:

```

wchar_t* check_month(wchar_t *inp_str){
    wchar_t *rus_months[] = {L"января", L"февраля", L"марта", L"апреля",
L"мая", L"июня", L"июля", L"августа", L"сентября", L"октября", L"ноября",
L"декабря"};
    wchar_t *eng_months[] = {L"january", L"february", L"march", L"april",
L"may", L"june", L"july", L"august", L"september", L"october", L"november",
L"december"};

    for(int i=0; i<12; i++){
        if(wcscasecmp(inp_str, rus_months[i]) == 0 || wcscasecmp(inp_str,
eng_months[i]) == 0){
            wchar_t *buf = malloc(3*sizeof(wchar_t*));
            if(!buf){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
            if(i<10){
                swprintf(buf, 3, L"0%d", i+1);
            }
        }
    }
}

```

```

        } else {
            swprintf(buf, 3, L"%d", i+1);
        }
        return buf;
    }
}

wchar_t* check_date(wchar_t *inp_str){
    wchar_t *dates[] = {L"1", L"2", L"3", L"4", L"5", L"6", L"7", L"8", L"9",
L"10",
                                L"11", L"12", L"13", L"14", L"15", L"16",
L"17", L"18", L"19", L"20",
                                L"21", L"22", L"23", L"24", L"25", L"26",
L"27", L"28", L"29", L"30", L"31"};

    for(int i=0; i<31; i++){
        if(wcscmp(inp_str, dates[i]) == 0){
            wchar_t *buf = malloc(3*sizeof(wchar_t*));
            if(!buf){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
            if(i<10){
                swprintf(buf, 3, L"0%d", i+1);
            } else {
                swprintf(buf, 3, L"%d", i+1);
            }
            return buf;
        }
    }
}

```

В функцию подается указатель на структуру Text, которая используется для получения доступа к строковым значениям указателя на структуру Sentence. Сначала создается массив слов в предложении, затем, следуя определенным условиям (поиск дня и месяца в строке осуществлен при помощи вспомогательных функций), в массив временных данных записываются элементы подстроки, которая содержит дату. Далее при помощи функции swprintf производится приведение временных данных к форматной строке. После в цикле высчитывается начало и конец подстроки, которая в последствии заменяется на форматную строку. В конце функции происходит очистка из памяти всех временных переменных.

1.5. Функция сортировки предложений по произведению длин слов

Сама функция:

```
void sort_sents(struct Text *text){
    qsort(text->sents, text->n, sizeof(struct Sentence *), cmp);
}
```

Функция для сравнения структур Sentence:

```
int cmp(const void* a, const void* b){
    struct Sentence **first_sent = (struct Sentence **) a;
    struct Sentence **second_sent = (struct Sentence **) b;

    int first = count_prod(*first_sent);
    int second = count_prod(*second_sent);

    if (first < second) return 1;
    if (first > second) return -1;
    return 0;
}
```

Функция, считающая произведение длин слов в предложении:

```
int count_prod(struct Sentence *sentence){
    wchar_t sep[10] = L" ,!?:\n";
    wchar_t **word_sent = malloc(MEM_STEP*sizeof(wchar_t*));
    if(!word_sent){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    int count_elem = 0;

    wchar_t *cur_sent = sentence->str;
    wchar_t temp_sent[wcslen(cur_sent)];
    wcscpy(temp_sent, cur_sent);

    wchar_t *istr;
    wchar_t *pwc;
    istr = wcstok(temp_sent, sep, &pwc);
    while(istr != NULL){
        wchar_t *temp = malloc(sizeof(wchar_t*));
        if(!temp){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
        wcscpy(temp, istr);
        word_sent[count_elem] = temp;
        count_elem++;
        istr = wcstok(NULL, sep, &pwc);
    }
}
```

```

    }

    int cur_prod = 1;

    for(int i=0; i<count_elem; i++){
        int temp_count_words = 0;
        for(int j=0; j<wcslen(word_sent[i]); j++){
            temp_count_words++;
        }
        if(temp_count_words != 0){
            cur_prod *= temp_count_words;
        }
    }

    for(int i=0; i<count_elem; i++){
        free(word_sent[i]);
    }
    free(word_sent);

    return cur_prod;
}

```

Функция `sort_sents()` принимает в качестве аргумента указатель на структуру `Text`, массив указателей на указатели на структуру `Sentence` которой передает в функцию `qsort()`. В качестве функции сравнения, необходимой для работы функции `qsort()`, написана функция `cmp()`, которая принимает константные указатели на элементы сортируемого массива. Аргументы приводятся к типу сортируемых элементов массива, а в функцию `count_prod()` передается указатель на структуру `Sentence`. В функции `count_prod()` предложение разбивается на слова, которые записываются во временный массив слов, и вычисляется произведение длин слов в предложении. На основании результата функции, происходит сортировка предложений в тексте.

1.6. Функция удаления предложений, содержащих символ «№» или «#», но не содержат ни одной цифры

Сама функция:

```
void remove_symbols(struct Text *text){
    int i = 0;

    while(i < text->n){
        if (check_entry(text->sents[i]) == 1){
            free(text->sents[i]);
            memmove(&text->sents[i], &text->sents[i+1], (text->n-
i+1)*sizeof(struct Sentence));
            text->n -= 1;
        } else {
            i++;
        }
    }
}
```

Вспомогательная функция:

```
int check_entry(struct Sentence *sentence){

    for(int i=0; i<wcslen(sentence->str); i++){
        wchar_t symbol = sentence->str[i];
        int found_symbol = 0;
        if(symbol == L'#' || symbol == L'№'){
            found_symbol = 1;
            for(int j=0; j<wcslen(sentence->str); j++){
                wchar_t num = sentence->str[j];
                if (num < '0' || num > '9'){
                    continue;
                } else {
                    return 0;
                }
            }
            if(found_symbol == 0){
                return 0;
            } else {
                return 1;
            }
        }
    }
}
```


Функция получается в качестве аргумента указатель на структуру Text, содержащий указатель на структуру Sentence, который передается в вспомогательную функцию, в которой происходит проверка предложения на условия содержания символов. В случае удовлетворения условиям, предложение удаляется, а на его место записывается следующее за ним предложение.

1.7. Главная функция

```
int main(){
    setlocale(LC_ALL, "");
    wprintf(L"%ls\n", L"Приветствую!\n\nДанная программа обрабатывает
введенный вами текст.\nТребования к тексту:\n\n1. Текст представляет собой
предложения, разделенные точкой. Предложения - набор слов, разделенные
пробелом или запятой, слова - набор латинских или кириллических букв, цифр
и других символов кроме точки, пробела или запятой.\n2. Для выхода из ввода
текста необходимо дважды нажать клавишу Enter.\n\nПосле ввода текста
автоматически удаляются повторяющиеся предложения.\nПожалуйста, введите
текст:");
    struct Text text = readText();

    changed_text(&text);
    wprintf(L"%ls\n", L"Ваш текст после удаления повторяющихся
предложений:");
    print_text(&text);

    wprintf(L"%ls\n", L"");

    wprintf(L"%ls\n", L"Режим работы с текстом\nКакое действие с текстом
вы хотите выполнить?");
    wprintf(L"%ls\n", L"Введите номер команды:\n1. Распечатать каждое
слово которое встречается не более одного раза в тексте.\n2. Каждую подстроку
в тексте имеющую вид "<день> <месяц> <год> г." заменить на подстроку вида
"ДД/ММ/ГГГГ" (аналогично для английского языка).\n3. Отсортировать
предложения по произведению длин слов в предложении.\n4. Удалить все
предложения, которые содержат символ '#' или '№', но не содержат ни одной
цифры.\n0. Выход из программы");
    wprintf(L"%ls", L">");

    wchar_t command = getwchar();
    switch(command){
        case L'0':
            free_text(&text);
            return 0;
        case L'1':
            wprintf(L"%ls\n", L"Результат работы функции:");
```

```

        single_word(&text);
        break;
    case L'2':
        change_date(&text);
        wprintf(L"%ls\n", L"Результат работы функции:");
        print_text(&text);
        break;
    case L'3':
        sort_sents(&text);
        wprintf(L"%ls\n", L"Результат работы функции:");
        print_text(&text);
        break;
    case L'4':
        remove_symbols(&text);
        wprintf(L"%ls\n", L"Результат работы функции:");
        print_text(&text);
        break;
    default:
        wprintf(L"%ls\n", L"Неверно введена команда!");
        break;

    free_text(&text);
    return 0;
}
}

```

В главной функции реализован вывод информации о программе, вызов всех необходимых функций для работы с текстом, а также памятка по функциям программы, выбор действия пользователем, и выход из программы.

2. ОПИСАНИЕ MAKEFILE

В ходе разработки программы, для ее сборки, было необходимо написать

Makefile:

```
all: main.o first_func.o second_func.o third_func.o fourth_func.o textio.o
    gcc main.o first_func.o second_func.o third_func.o fourth_func.o textio.o -o
program
```

```
textio.o: textio.c
    gcc -c textio.c
```

```
first_func.o: first_func.c textio.h
    gcc -c first_func.c
```

```
second_func.o: second_func.c textio.h
    gcc -c second_func.c
```

```
third_func.o: third_func.c textio.h
    gcc -c third_func.c
```

```
fourth_func.o: fourth_func.c textio.h
    gcc -c fourth_func.c
```

```
main.o: main.c textio.h all_funcs.h
    gcc -c main.c
```

```
clean:
    rm -rf *.o program
```

В Makefile происходит описание компиляции как всей программы, так и всех необходимых для этого файлов .o (вынесенных функций в файлы .c, необходимых для компиляции программы). Также реализована инструкция clean, позволяющая отчистить результат работы программы из локальной директории.

ЗАКЛЮЧЕНИЕ

Для успешного написания программы для обработки текстовых данных, соответствующей заданию курсовой работы, были выполнены следующие задачи:

1. Изучен теоретический материал по теме курсовой работы.
2. Разработан и реализован программный код.
3. Проведено тестирование программы.

Исходный код программы представлен в приложении А, результаты тестирования — в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

*1. The C Programming Language / Brian W. Kernigan, Dennis M. Ritchie
Second edition, 1988. 288 с.*

*2. The C++ Resources network [Электронный ресурс]
URL: <http://cplusplus.com>*

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MEM_STEP 100
#define CRASH_MEMORY_MSG L"Ошибка выделения памяти"

#include <wchar.h>
#include <locale.h>

#include "textio.h"
#include "all_funcs.h"

int main(){
    setlocale(LC_ALL, "");
    wprintf(L"%ls\n", L"Приветствую!\n\nДанная программа обрабатывает
введенный вами текст.\nТребования к тексту:\n\n1. Текст представляет собой
предложения, разделенные точкой. Предложения - набор слов, разделенные
пробелом или запятой, слова - набор латинских или кириллических букв, цифр
и других символов кроме точки, пробела или запятой.\n2. Для выхода из ввода
текста необходимо дважды нажать клавишу Enter.\n\nПосле ввода текста
автоматически удаляются повторяющиеся предложения.\nПожалуйста, введите
текст:");
    struct Text text = readText();

    changed_text(&text);
    wprintf(L"%ls\n", L"Ваш текст после удаления повторяющихся
предложений:");
    print_text(&text);

    wprintf(L"%ls\n", L "");

    wprintf(L"%ls\n", L"Режим работы с текстом\nКакое действие с текстом
вы хотите выполнить?");
    wprintf(L"%ls\n", L"Введите номер команды:\n1. Распечатать каждое
слово которое встречается не более одного раза в тексте.\n2. Каждую подстроку
в тексте имеющую вид "<день> <месяц> <год> г." заменить на подстроку вида
"ДД/ММ/ГГГГ" (аналогично для английского языка).\n3. Отсортировать
предложения по произведению длин слов в предложении.\n4. Удалить все
```

предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.\n0. Выход из программы");

```
wprintf(L"%ls", L">");
```

```
wchar_t command = getwchar();
```

```
switch(command){
```

```
    case L'0':
```

```
        free_text(&text);
```

```
        return 0;
```

```
    case L'1':
```

```
        wprintf(L"%ls\n", L"Результат работы функции:");
```

```
        single_word(&text);
```

```
        break;
```

```
    case L'2':
```

```
        change_date(&text);
```

```
        wprintf(L"%ls\n", L"Результат работы функции:");
```

```
        print_text(&text);
```

```
        break;
```

```
    case L'3':
```

```
        sort_sents(&text);
```

```
        wprintf(L"%ls\n", L"Результат работы функции:");
```

```
        print_text(&text);
```

```
        break;
```

```
    case L'4':
```

```
        remove_symbols(&text);
```

```
        wprintf(L"%ls\n", L"Результат работы функции:");
```

```
        print_text(&text);
```

```
        break;
```

```
    default:
```

```
        wprintf(L"%ls\n", L"Неверно введена команда!");
```

```
        break;
```

```
free_text(&text);
```

```
return 0;
```

```
}
```

```
}
```

Файл: textio.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MEM_STEP 100
#define CRASH_MEMORY_MSG L"Ошибка выделения памяти"

#include <wchar.h>
#include <locale.h>

struct Sentence{
    wchar_t *str;
    int size;
};

struct Text{
    struct Sentence **sents;
    int n;
    int size;
};

struct Sentence* readSentence();

struct Text readText(){
    int size = MEM_STEP;
    struct Sentence** text = malloc(size*sizeof(struct Sentence*));
    if(!text){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    int n = 0;
    struct Sentence* temp;
    int nlcount = 0;
    do{
        temp = readSentence();
        if(temp->str[0] == '\n'){
            nlcount++;
        } else {
            nlcount = 0;
            text[n] = temp;
            n++;
        }
    } while (nlcount<2);
    struct Text txt;
    txt.size = size;
    txt.sents = text;
}
```



```

    txt.n = n;
    return txt;
}

struct Sentence* readSentence(){
    int size = MEM_STEP;
    wchar_t *buf = malloc(size*sizeof(wchar_t));
    if(!buf){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    int n = 0;
    wchar_t temp;
    do{
        if (n >= size-2){
            wchar_t *t = realloc(buf, size+MEM_STEP);
            if(!t){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
            size += MEM_STEP;
            buf = t;
        }
        temp = getwchar();
        buf[n] = temp;
        n++;
    } while (temp!='\n' && temp!='.' && temp!='!');
    buf[n] = '\0';
    struct Sentence *sentence = malloc(sizeof(struct Sentence));
    if(!sentence){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    sentence -> str = buf;
    sentence -> size = size;
    return sentence;
}

void changed_text(struct Text* text){
    int i = 0;
    int j = i + 1;

    while(i < text->n-1){
        while(j < text->n){
            if (wcscasecmp(text->sents[i]->str, text->sents[j]->str) == 0){
                free(text->sents[j]);
                memmove(&text->sents[j], &text->sents[j+1], (text->n-
j+1)*sizeof(struct Sentence));
                text->n -= 1;
            } else {
                j++;
            }
        }
        i++;
    }
}

```

```

        j = i + 1;
    }
}

void print_text(struct Text *text){
    for (int i=0; i<text->n; i++){
        wprintf(L"%ls", text->sents[i]->str);
    }
    wprintf(L"%ls\n", L "");
}

```

```

void free_text(struct Text *text){
    int text_len = text->n;
    for(int i=0; i<text_len; i++){
        free(text->sents[i]);
    }
}

```

Файл: first_func.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MEM_STEP 100
#define CRASH_MEMORY_MSG L"Ошибка выделения памяти"

#include <wchar.h>
#include <locale.h>

#include "textio.h"

void single_word(struct Text *text){
    wchar_t sep[10] = L",!?:\n";
    wchar_t **word_text = malloc(text->n*MEM_STEP*sizeof(wchar_t*));
    if(!word_text){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    int count_elem = 0;

    for(int i=0; i<text->n; i++){
        wchar_t *cur_sent = text->sents[i]->str;
        wchar_t temp_sent[wcslen(cur_sent)];
        wcscpy(temp_sent, cur_sent);

        wchar_t *istr;
        wchar_t *pwc;
    }
}

```

```

        istr = wcstok(temp_sent, sep, &pwc);
        while(istr != NULL){
            wchar_t *temp = malloc(sizeof(wchar_t*));
            if(!temp){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
            wcscpy(temp, istr);
            word_text[count_elem] = temp;
            count_elem++;
            istr = wcstok(NULL, sep, &pwc);
        }
    }

    for(int j=0; j<count_elem; j++){
        int flag = 0;
        wchar_t *cmp_item = word_text[j];
        for(int k=0; k<count_elem; k++){
            if (j == k){
                continue;
            }
            if(wcscmp(cmp_item, word_text[k]) == 0){
                flag = 0;
                break;
            } else {
                flag = 1;
            }
        }
        if(flag == 1){
            wprintf(L"%ls\n", cmp_item);
        }
    }

    for(int i=0; i<count_elem; i++){
        free(word_text[i]);
    }
    free(word_text);
}

```

Файл: second_func.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MEM_STEP 100
#define CRASH_MEMORY_MSG L"Ошибка выделения памяти"

```

```

#include <wchar.h>
#include <locale.h>

#include "textio.h"

wchar_t* check_month(wchar_t *inp_str){
    wchar_t *rus_months[] = {L"января", L"февраля", L"марта", L"апреля",
L"мая", L"июня", L"июля", L"августа", L"сентября", L"октября", L"ноября",
L"декабря"};
    wchar_t *eng_months[] = {L"january", L"february", L"march", L"april",
L"may", L"june", L"july", L"august", L"september", L"october", L"november",
L"december"};

    for(int i=0; i<12; i++){
        if(wcscasecmp(inp_str, rus_months[i]) == 0 || wcscasecmp(inp_str,
eng_months[i]) == 0){
            wchar_t *buf = malloc(3*sizeof(wchar_t*));
            if(!buf){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
            if(i<10){
                swprintf(buf, 3, L"0%d", i+1);
            } else {
                swprintf(buf, 3, L"%d", i+1);
            }
            return buf;
        }
    }
}

wchar_t* check_date(wchar_t *inp_str){
    wchar_t *dates[] = {L"1", L"2", L"3", L"4", L"5", L"6", L"7", L"8", L"9",
L"10",
                                L"11", L"12", L"13", L"14", L"15", L"16",
L"17", L"18", L"19", L"20",
                                L"21", L"22", L"23", L"24", L"25", L"26",
L"27", L"28", L"29", L"30", L"31"};

    for(int i=0; i<31; i++){
        if(wcscmp(inp_str, dates[i]) == 0){
            wchar_t *buf = malloc(3*sizeof(wchar_t*));
            if(!buf){wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
            if(i<10){
                swprintf(buf, 3, L"0%d", i+1);
            } else {
                swprintf(buf, 3, L"%d", i+1);
            }
        }
    }
}

```

```

        }
        return buf;
    }
}

void change_date(struct Text *text){
    wchar_t sep[10] = L" ,!?:\\n";

    for(int i=0; i<text->n; i++){
        int cur_elem = 0;
        wchar_t **work_sent = malloc(MEM_STEP*sizeof(wchar_t*));
        if(!work_sent){wprintf(L"%ls\\n", CRASH_MEMORY_MSG);}

        wchar_t *cur_sent = text->sents[i]->str;
        wchar_t temp_sent[wcslen(cur_sent)];
        wcscpy(temp_sent, cur_sent);

        wchar_t *istr;
        wchar_t *pwc;
        istr = wcstok(temp_sent, sep, &pwc);
        while(istr != NULL){
            wchar_t *temp = calloc(*istr, sizeof(wchar_t*));
            if(!temp){wprintf(L"%ls\\n", CRASH_MEMORY_MSG);}
            wcscpy(temp, istr);
            work_sent[cur_elem] = temp;
            cur_elem++;
            istr = wcstok(NULL, sep, &pwc);
        }

        wchar_t **temp_date = malloc(3*sizeof(wchar_t*));
        if(!temp_date){wprintf(L"%ls\\n", CRASH_MEMORY_MSG);}

        for(int j=0; j<cur_elem-3; j++){
            if(wcscmp(work_sent[j+3], L"y.") == 0 || wcscmp(work_sent[j+3],
L"r.") == 0){
                wchar_t* date = check_date(work_sent[j]);
                if(date != NULL){
                    temp_date[0] = date;
                    wchar_t* num_mounts =
check_month(work_sent[j+1]);
                    if(num_mounts != NULL){
                        temp_date[1] = num_mounts;

```

```

        if(work_sent[j+2][0] > L'0' &&
work_sent[j+2][0] <= L'9'){
            temp_date[2] = work_sent[j+2];
            } else {
                break;
            }
        } else {
            break;
        }
    } else {
        break;
    }
}

}

wchar_t *replace = malloc(30*sizeof(wchar_t*));
if(!replace){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
swprintf(replace, 30, L"%ls/%ls/%ls", temp_date[0], temp_date[1],
temp_date[2]);

int wchar_start = 0;
int wchar_end = 0;

if(temp_date[0] != NULL && temp_date[1] != NULL && temp_date[2]
!= NULL){
    for(int i=0; i<wcslen(cur_sent); i++){
        if(cur_sent[i] == temp_date[0][1] || cur_sent[i] ==
temp_date[0][0]){
            wchar_start = i;
            int cur_num_year = 0;
            for(int j=i+1; j<wcslen(cur_sent); j++){
                if(cur_sent[j] ==
temp_date[2][cur_num_year]){
                    wchar_end = j;
                    cur_num_year++;
                }
            }
            if(wchar_end == 0){
                wchar_start = 0;
            } else {
                break;
            }
        }
    }
}

```

```

        }
    }

    memmove(cur_sent+wchar_start, replace, ((wchar_end-wchar_start) *
sizeof(wchar_t)));

    free(temp_date);
    free(replace);
    for(int i=0; i<cur_elem; i++){
        free(work_sent[i]);
    }
    free(work_sent);
}
}

```

Файл: third_func.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MEM_STEP 100
#define CRASH_MEMORY_MSG L"Ошибка выделения памяти"

#include <wchar.h>
#include <locale.h>

#include "textio.h"

int count_prod(struct Sentence *sentence){
    wchar_t sep[10] = L",!?:\n";
    wchar_t **word_sent = malloc(MEM_STEP*sizeof(wchar_t*));
    if(!word_sent){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
    int count_elem = 0;

    wchar_t *cur_sent = sentence->str;
    wchar_t temp_sent[wcslen(cur_sent)];
    wcscpy(temp_sent, cur_sent);

    wchar_t *istr;
    wchar_t *pwc;
    istr = wcstok(temp_sent, sep, &pwc);
    while(istr != NULL){
        wchar_t *temp = malloc(sizeof(wchar_t*));
        if(!temp){ wprintf(L"%ls\n", CRASH_MEMORY_MSG);}
        wcscpy(temp, istr);
    }
}

```

```

        word_sent[count_elem] = temp;
        count_elem++;
        istr = wcstok(NULL, sep, &pwc);
    }

    int cur_prod = 1;

    for(int i=0; i<count_elem; i++){
        int temp_count_words = 0;
        for(int j=0; j<wcslen(word_sent[i]); j++){
            temp_count_words++;
        }
        if(temp_count_words != 0){
            cur_prod *= temp_count_words;
        }
    }

    for(int i=0; i<count_elem; i++){
        free(word_sent[i]);
    }
    free(word_sent);

    return cur_prod;
}

int cmp(const void* a, const void* b){
    struct Sentence **first_sent = (struct Sentence **) a;
    struct Sentence **second_sent = (struct Sentence **) b;

    int first = count_prod(*first_sent);
    int second = count_prod(*second_sent);

    if (first < second) return 1;
    if (first > second) return -1;
    return 0;
}

void sort_sents(struct Text *text){
    qsort(text->sents, text->n, sizeof(struct Sentence *), cmp);
}

```


Файл: fourth_func.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MEM_STEP 100
#define CRASH_MEMORY_MSG L"Ошибка выделения памяти"

#include <wchar.h>
#include <locale.h>

#include "textio.h"

int check_entry(struct Sentence *sentence){

    for(int i=0; i<wcslen(sentence->str); i++){
        wchar_t symbol = sentence->str[i];
        int found_symbol = 0;
        if(symbol == L'#' || symbol == L'№'){
            found_symbol = 1;
            for(int j=0; j<wcslen(sentence->str); j++){
                wchar_t num = sentence->str[j];
                if (num < '0' || num > '9'){
                    continue;
                } else {
                    return 0;
                }
            }
            if(found_symbol == 0){
                return 0;
            } else {
                return 1;
            }
        }
    }
}

void remove_symbols(struct Text *text){
    int i = 0;

    while(i < text->n){
        if (check_entry(text->sents[i]) == 1){
            free(text->sents[i]);
        }
        i++;
    }
}
```

```

        memmove(&text->sents[i], &text->sents[i+1], (text->n-
i+1)*sizeof(struct Sentence));
        text->n -= 1;
    } else {
        i++;
    }
}
}

```

Файл: textio.h

```

struct Sentence{
    wchar_t *str;
    int size;
};

struct Text{
    struct Sentence **sents;
    int n;
    int size;
};

struct Text readText();
struct Sentence readSentence();
void changed_text(struct Text *text);
void print_text(struct Text *text);
void free_text(struct Text *text);

```

Файл: all_funcs.h

```

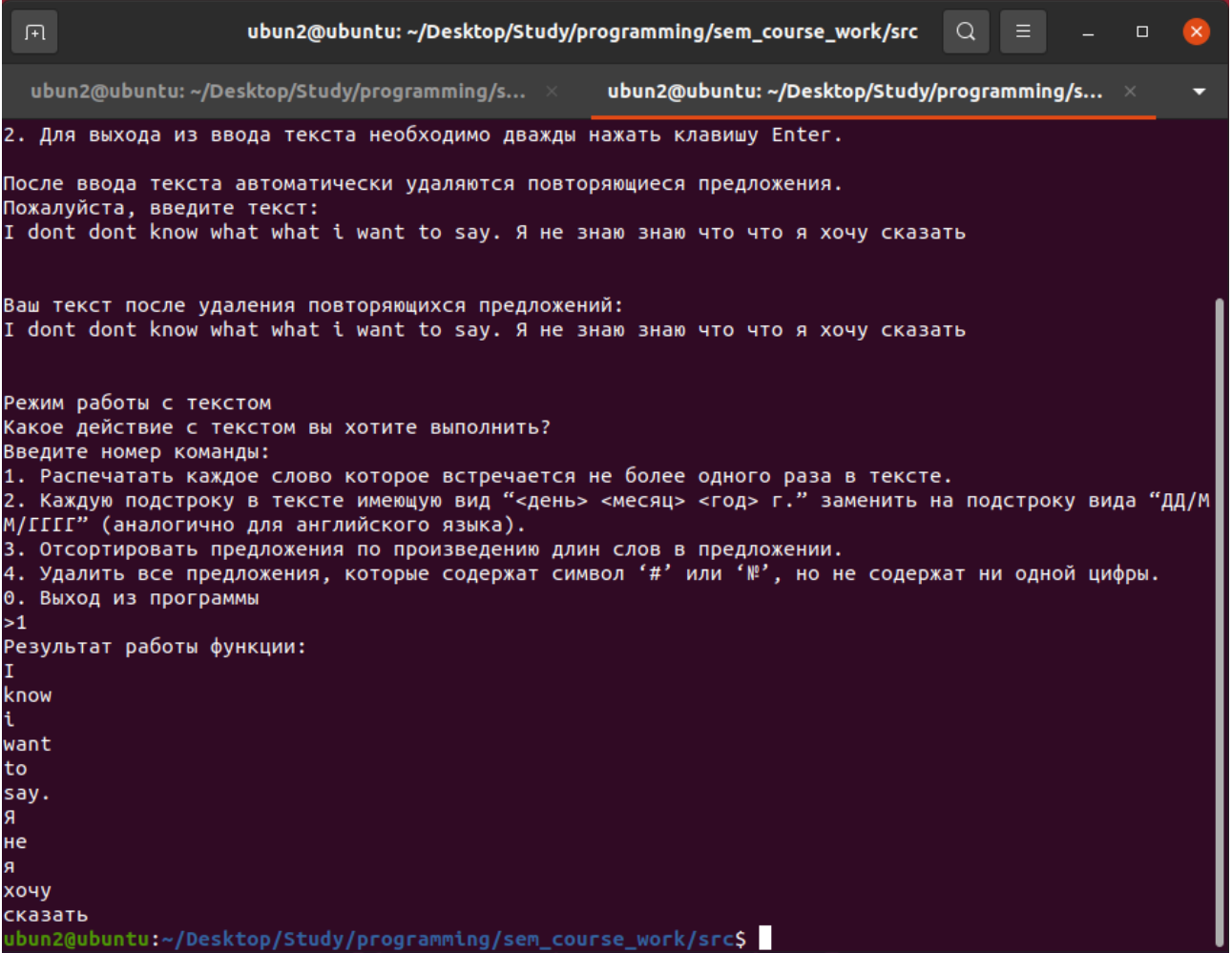
void single_word(struct Text *text);
void change_date(struct Text *text);
void sort_sents(struct Text *text);
void remove_symbols(struct Text *text);

```

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

1. Распечатать каждое слово которое встречается не более одного раза в тексте.



```
ubun2@ubuntu: ~/Desktop/Study/programming/sem_course_work/src
2. Для выхода из ввода текста необходимо дважды нажать клавишу Enter.
После ввода текста автоматически удаляются повторяющиеся предложения.
Пожалуйста, введите текст:
I dont dont know what what i want to say. Я не знаю знаю что что я хочу сказать

Ваш текст после удаления повторяющихся предложений:
I dont dont know what what i want to say. Я не знаю знаю что что я хочу сказать

Режим работы с текстом
Какое действие с текстом вы хотите выполнить?
Введите номер команды:
1. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид "<день> <месяц> <год> г." заменить на подстроку вида "ДД/М
М/ГГГГ" (аналогично для английского языка).
3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.
0. Выход из программы
>1
Результат работы функции:
I
know
i
want
to
say.
я
не
я
хочу
сказать
ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$
```

2. Каждую подстроку в тексте имеющую вид “<день> <месяц> <год> г.” заменить на подстроку вида “ДД/ММ/ТТТТ”.

```
ubun2@ubuntu: ~/Desktop/Study/programming/sem_course_work/src
ubun2@ubuntu: ~/Desktop/Study/programming/s... x ubun2@ubuntu: ~/Desktop/Study/programming/s... x
Данная программа обрабатывает введенный вами текст.
Требования к тексту:
1. Текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой.
2. Для выхода из ввода текста необходимо дважды нажать клавишу Enter.
После ввода текста автоматически удаляются повторяющиеся предложения.
Пожалуйста, введите текст:
Я был здесь 13 марта 2002 г. Это было давно
I was found it 12 november 2021 y. What was it?
Ваш текст после удаления повторяющихся предложений:
Я был здесь 13 марта 2002 г. Это было давно
I was found it 12 november 2021 y. What was it?
Режим работы с текстом
Какое действие с текстом вы хотите выполнить?
Введите номер команды:
1. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид “<день> <месяц> <год> г.” заменить на подстроку вида “ДД/ММ/ТТТТ” (аналогично для английского языка).
3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ ‘#’ или ‘№’, но не содержат ни одной цифры.
0. Выход из программы
>2
Результат работы функции:
Я был здесь 13/03/2002 Это было давно
I was found it 12/11/2021 What was it?
ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$
```

3. Отсортировать предложения по произведению длин слов в предложении.

```
ubun2@ubuntu: ~/Desktop/Study/programming/sem_course_work/src
1. Текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой.
2. Для выхода из ввода текста необходимо дважды нажать клавишу Enter.

После ввода текста автоматически удаляются повторяющиеся предложения.
Пожалуйста, введите текст:
qwe rty fgh
qwe
qwe rty

Ваш текст после удаления повторяющихся предложений:
qwe rty fgh
qwe
qwe rty

Режим работы с текстом
Какое действие с текстом вы хотите выполнить?
Введите номер команды:
1. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид "<день> <месяц> <год> г." заменить на подстроку вида "ДД/ММ/ГГГГ" (аналогично для английского языка).
3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.
0. Выход из программы
>3
Результат работы функции:
qwe rty fgh
qwe rty
qwe

ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$
```

4. Удалить все предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.

```
ubun2@ubuntu: ~/Desktop/Study/programming/sem_course_work/src
Предложение с №
Предложение с # и еще чем то
Предложение с № 1 2 3
Предложение #4
Предложение 1 2 3
Предложение

Ваш текст после удаления повторяющихся предложений:
Предложение с №
Предложение с # и еще чем то
Предложение с № 1 2 3
Предложение #4
Предложение 1 2 3
Предложение

Режим работы с текстом
Какое действие с текстом вы хотите выполнить?
Введите номер команды:
1. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид "<день> <месяц> <год> г." заменить на подстроку вида "ДД/М
М/ГГГГ" (аналогично для английского языка).
3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.
0. Выход из программы
>4
Результат работы функции:
Предложение с № 1 2 3
Предложение #4
Предложение 1 2 3
Предложение

ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$
```

Удаление повторяющихся предложений:

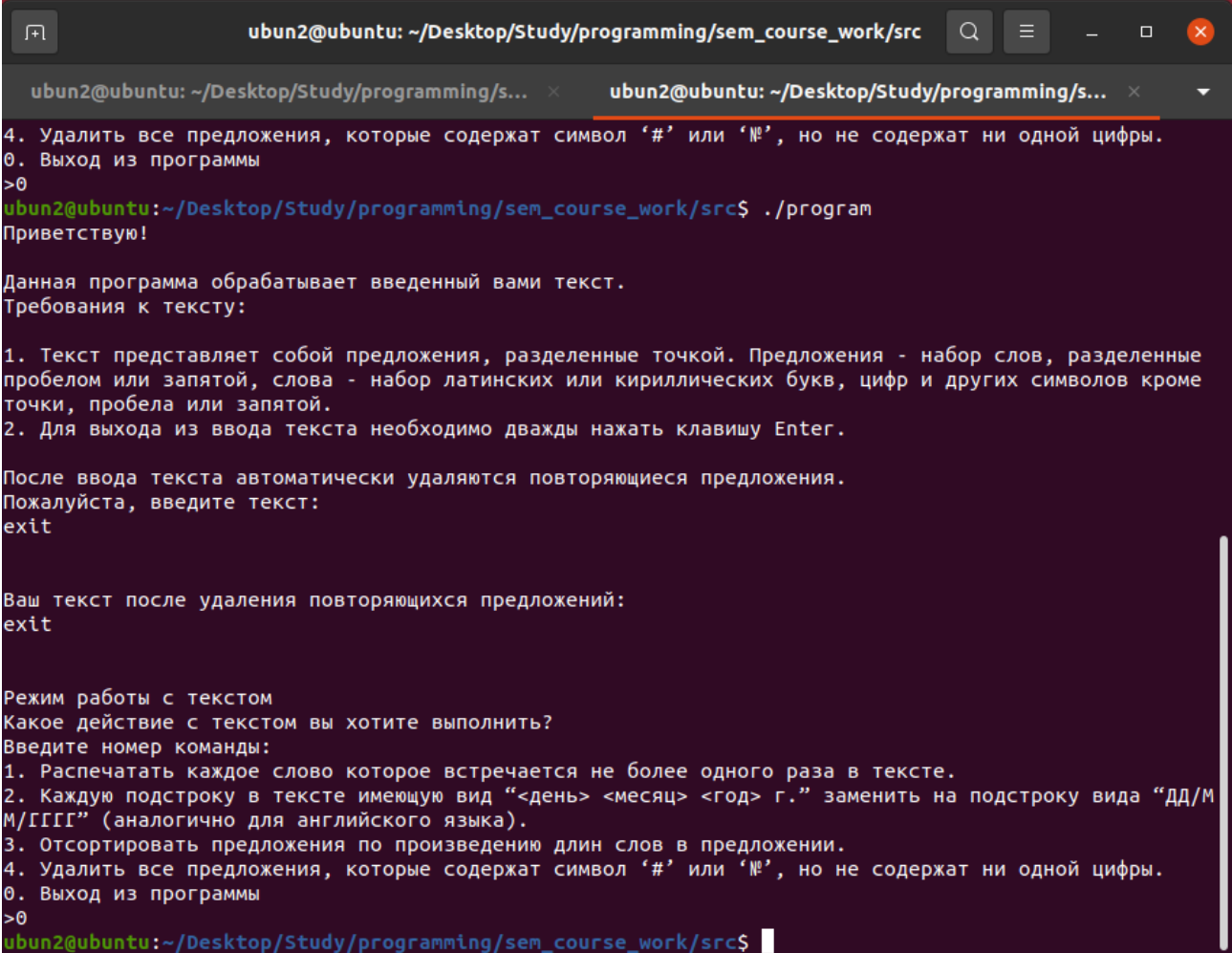
```
ubun2@ubuntu: ~/Desktop/Study/programming/sem_course_work/src
Данная программа обрабатывает введенный вами текст.
Требования к тексту:
1. Текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой.
2. Для выхода из ввода текста необходимо дважды нажать клавишу Enter.

После ввода текста автоматически удаляются повторяющиеся предложения.
Пожалуйста, введите текст:
Новое предложение
Новое предложение
1st one
1st one
last

Ваш текст после удаления повторяющихся предложений:
Новое предложение
1st one
last

Режим работы с текстом
Какое действие с текстом вы хотите выполнить?
Введите номер команды:
1. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид "<день> <месяц> <год> г." заменить на подстроку вида "ДД/ММ/ГГГГ" (аналогично для английского языка).
3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.
0. Выход из программы
>0
ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$
```

Выход из программы:



```
ubun2@ubuntu: ~/Desktop/Study/programming/sem_course_work/src
ubun2@ubuntu: ~/Desktop/Study/programming/s... x ubun2@ubuntu: ~/Desktop/Study/programming/s... x
4. Удалить все предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.
0. Выход из программы
>0
ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$ ./program
Приветствую!

Данная программа обрабатывает введенный вами текст.
Требования к тексту:

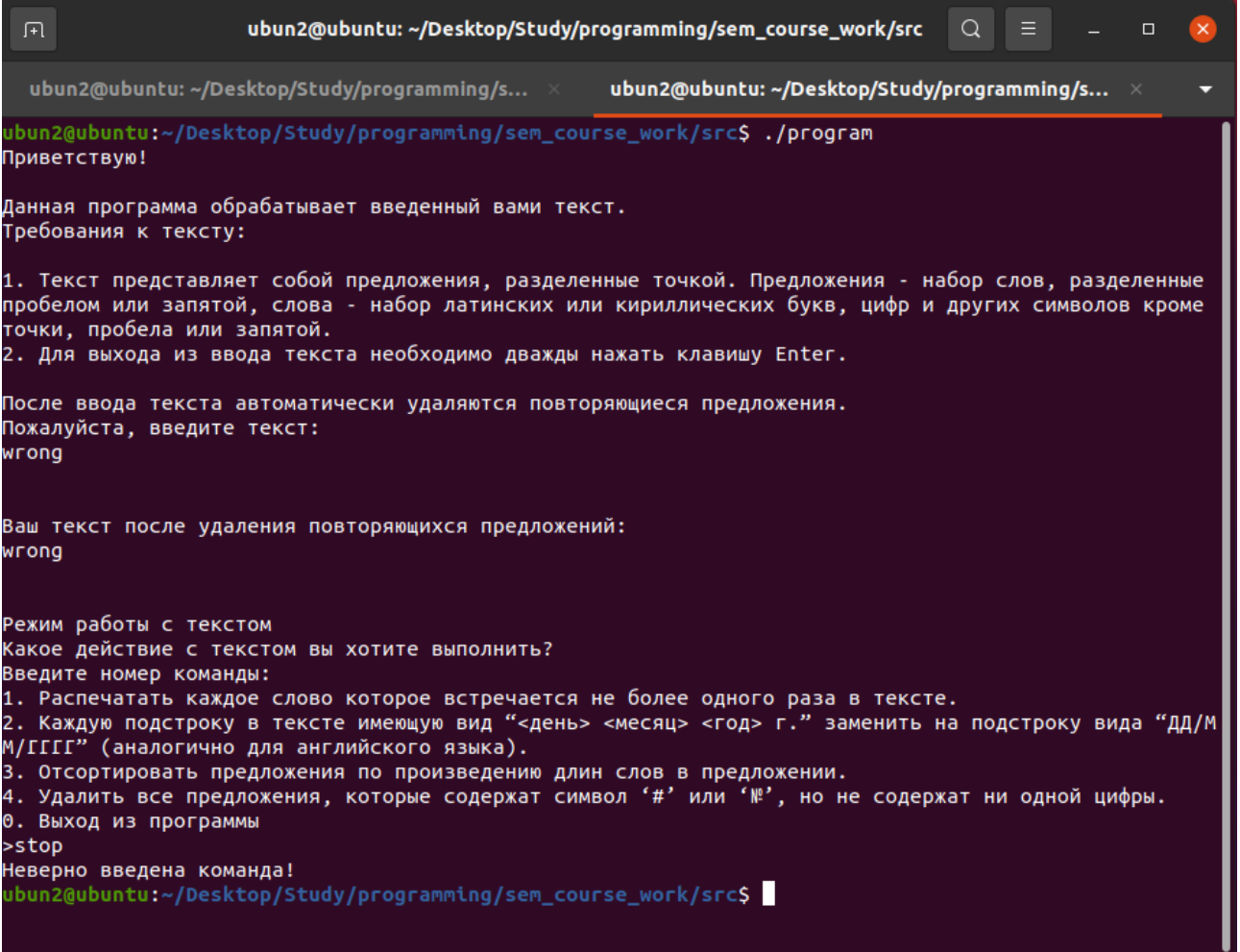
1. Текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой.
2. Для выхода из ввода текста необходимо дважды нажать клавишу Enter.

После ввода текста автоматически удаляются повторяющиеся предложения.
Пожалуйста, введите текст:
exit

Ваш текст после удаления повторяющихся предложений:
exit

Режим работы с текстом
Какое действие с текстом вы хотите выполнить?
Введите номер команды:
1. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид "<день> <месяц> <год> г." заменить на подстроку вида "ДД/М
М/ГГГГ" (аналогично для английского языка).
3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.
0. Выход из программы
>0
ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$
```


Обработка неправильного ввода команды:



```
ubun2@ubuntu: ~/Desktop/Study/programming/sem_course_work/src
ubun2@ubuntu: ~/Desktop/Study/programming/s... x ubun2@ubuntu: ~/Desktop/Study/programming/s... x
ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$ ./program
Приветствую!

Данная программа обрабатывает введенный вами текст.
Требования к тексту:

1. Текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских или кириллических букв, цифр и других символов кроме точки, пробела или запятой.
2. Для выхода из ввода текста необходимо дважды нажать клавишу Enter.

После ввода текста автоматически удаляются повторяющиеся предложения.
Пожалуйста, введите текст:
wrgong

Ваш текст после удаления повторяющихся предложений:
wrgong

Режим работы с текстом
Какое действие с текстом вы хотите выполнить?
Введите номер команды:
1. Распечатать каждое слово которое встречается не более одного раза в тексте.
2. Каждую подстроку в тексте имеющую вид "<день> <месяц> <год> г." заменить на подстроку вида "ДД/ММ/ГГГГ" (аналогично для английского языка).
3. Отсортировать предложения по произведению длин слов в предложении.
4. Удалить все предложения, которые содержат символ '#' или '№', но не содержат ни одной цифры.
0. Выход из программы
>stop
Неверно введена команда!
ubun2@ubuntu:~/Desktop/Study/programming/sem_course_work/src$
```