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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Обработка текста

Студент гр. 1304 Арчибасов Е.О.

Преподаватель Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Студент:Арчибасов Е.О.

Группа: 1304

Тема работы: Обработка текста

Исходные данные:

Текст, состоящий из предложений, разделенных точкой. Предложение – набор слов, разделенных пробелом или запятой. Слово – набор латинских букв и цифр. Длина текста и каждого предложения заранее неизвестна.

Содержание пояснительной записки:

1. А
2. Введение
3. Работа программы
4. Сборка программы
5. Тестирование
6. Инструкция по использованию
7. Заключение
8. Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 15.10.2021
Дата сдачи реферата: 20.12.2021
Дата защиты реферата: 22.12.2021

Студент Арчибасов Е.О.

Преподаватель Чайка К.В.

АННОТАЦИЯ

Разработана программа, в самом начале работы которой пользователю предлагается ввести текст, состоящий их цифр и латинских букв. После этого пользователь может выбрать одно из 5 доступных действий:

1. Преобразовать предложения так, чтобы каждое первое слово в нем начиналось с заглавной буквы, а остальные символы были прописными
2. Удалить все предложения, состоящие из четного количества слов.
3. Отсортировать предложения по сумме количеств гласных букв в каждом втором слове.
4. Вывести на экран все предложения, в которых в середине предложения встречаются слова, состоящие из прописных букв. Данные будут выделены синим цветом.
5. Выйти из программы.

.

СОДЕРЖАНИЕ

Оглавление

ВВЕДЕНИЕ

Целью данной работы является изучение основных конструкций языка Си, основных функций стандартных библиотек. А также изучение структур. Во время выполнения данной задачи мы сможем оценить свои навыки в программировании. Для решения задач нам потребуется:

1. Изучить основные теоритические положения языка Си, его функции и различные конструкции.
2. Разработать и создать пользовательский интерфейс для более удобного использования программы.
3. Разобраться с тем, как можно более рационально использовать память.
4. Учесть всевозможные ошибки, при использовании пользователем данной программы

ОПИСАНИЕ СТРУКТУР

Word

Структура Word содержит в себе поля обозначающие следующее:

1. Char *symbols – массив символов, составляющих слово
2. Char end – символ который следует после этого слова
3. Int size – размер слова
4. Int memorySize – количество символов выделенное в памяти для этого слова

Инициализация Word происходит в функции initW(Word *word). На вход данная функция принимает указатель на структуру и устанавливает ей значения по умолчанию. Так же массив chars инициализируется на memorySize символов с помощью malloc. Если инициализировать массив не удалось, когда malloc вернула NULL, функция оповещает об этом пользователя, упоминая адрес структуры и желаемый размер, после чего завершает работу программы с кодом 0.

Добавить символ в структуру Word можно с помощью функции appChar(Word *word, char c). На вход она принимает указатель на структуру и символ, который нужно туда добавить. Прежде чем добавить символ в массив символов, в функции проверяется, не произойдет ли выход за границы массива при выполнении этого действия. В случае выхода под массив запрашивается на defaultN символов больше, чем было. При этом, если память не была выделена, то есть функция realloc вернула NULL, функция оповещает об этом пользователя, упоминая адрес структуры и желаемый размер, после чего завершает работу программы с кодом 0.

Для освобождения памяти, занимаемой структурой Word, была создана функция freeWord(Word *word), которая принимает слово, которое нужно удалить. Удаление слова – это освобождение памяти, занимаемой массивом символов этого слова, а также обнуления текущего и максимального размера слова.

Sentence

Структура Sentence содержит в себе следующие поля:

1. Char *words – массив структур Word, представляющий это предложение, который создается и изменяется динамически
2. Int size – размер предложения
3. Int memorySize – количество слов выделенное в памяти для этого предложения

Инициализация Sentence происходит в функции initS(Sentence *sentence), которая принимает на вход указатель на структуру и устанавливает ей значения по умолчанию. Массив words здесь же инициализируется на memorySize слов с помощью malloc . Если инициализировать массив не удалось, когда malloc вернула NULL, функция оповещает об этом пользователя, упоминая адрес структуры и желаемый размер, после чего завершает работу программы с кодом 0.

Для создания нового слова в предложении создана функция newW(Sentence *sentence). Функция принимает на вход указатель на предложение, в котором нужно создать слово, и возвращает указатель на инициализированное слово. Если words заполнен, происходит увеличение размера на default (если realloc вернул NULL, функция уведомляет об этом пользователя, передавая адрес структуры и необходимый размер, и завершает работу программы), после чего инициализируется следующая структура и возвращается ее адрес.

Для корректного освобождения памяти, занимаемой структурой Sentence, была создана функция freeSent(Sentence *sentence). Функция принимает на вход предложение, которое нужно “удалить”. Удаление предложения – это освобождение памяти, занимаемой каждым словом этого предложения, массивом слов этого предложения в целом, а также обнуления текущего и максимального размера предложения.

Text

Структура Text содержит в себе следующие поля:

1. Char *sents – массив структур Sent.
2. Int size – размер предложения
3. Int memorySize – количество слов предложений в памяти для этого текста

Инициализация Text происходит в функции initText(Text *txt). Функция принимает на вход указатель на структуру и устанавливает ей значения по умолчанию. Массив sentences здесь же инициализируется на memorySize предложений с помощью malloc. Если не удалосьпроинициализировать массив, так как malloc вернул NULL, функция оповещает об этом пользователя как и ранее.

Для создания нового предложения в тексты создана функция newSent(Text *txt). Функция принимет указатель на текст, в котором нужно создать предложение. Возвращает указатель на предложение. Если sent заполнен, происходит увеличение размера на default (если realloc вернул NULL, функция уведомляет об этом пользователя, передавая адрес структуры и необходимый размер, и завершает работу программы), после чего инициализируется следующая структура и возвращается ее адрес.

Для удаления предложения из текста создана функция rmSent(Text *txt, int index). Функция принимает на вход указатель на текст и индекс удаляемого

предложения в этом тексте. Сначала в функции происходит проверка на нахождения индекса в границах массива предложений текста. После этого освобождается память, занимаемая удаляемым предложение, а предложения, стоящие за ним, сдвигаются на одну позицию влево с помощью memmove. С помощтю realloc освобождаем последний, уже не нужный, элемент.

Для корректного освобождения памяти, занимаемой структурой Text, была создана функция freeText(Text *txt). Удаление текста – это освобождение памяти, которую занимают все предложения этого текста.

ВВОД

Для сохранения входных данных в структуру Text в файле objects/text.h объявлена, а в objects/text.c реализована функция readText(Text*), которая принимает указатель на инициализированную структуру Text, в которую будут сохраняться данные.

В функции readText(Text *txt) поддерживается считанный символ , указатель на текущее предложение currSent. В цикле читается новый символ. Если этот символ равен пробелу, и текущее слово включает в себя символы, это значит, что нужно создать новое слово, предварительно указав, что текущее не имеет окончания, кроме символа пробела. Если он равен запятой, тогда нужно создать новое слово, предварительно указав, что текущее оканчивается на запятую. Если он равен точке, тогда создаем новое предложение, а в нем – новое слово, предварительно пометив, что текущее слово оканчивается на точку. Если же этот символ не равен ни пробелу, ни запятой, ни точке, и при этом не равен переносу строки и не пустой, этот символ добавляется в текущее слово. Это выполняется, пока считанный символ не окажется пустым или равным переносу строки. Если в результате чтения цикла было создано новое пустое предложение, то его нужно просто удалить

УДАЛЕНИЕ ПОВТОРНО ВСТРЕЧАЮЩИХСЯ ПРЕДЛОЖЕНИЙ

Функции removeDoubleSent(Text *txt) на вход поступает указатель на текст, из которого нужно удалить эти предложения. В цикле она перебирает первое предложение, слева направо. Во вложенном цикле перебирается другое предложение, которое нужно проверить, справа налево до первого, не включая его. Функция sentEql(Sentence *first, Sentence *scnd) проверяет, являются ли эти предложения одинаковыми. Если да, то удаляется второе предложение.

Выполнение задач

По условию, функция должна преобразовать предложение так, чтобы каждое первое слово в нем начиналось с заглавной буквы. Для реализации данной задачи была составлена функция *editSent(Text* txt)*. В данной функции мы пробегаемся по всем предложениям, внутри предложений пробегаемся по

словам и с помощью функции *toupper* возводим первый символ каждого слова в верхний регистр.

По условию, функция должна отсортировать предложения по сумме количеств гласных букв в каждом втором слове. Для реализации данной задачи была составлена функция *sortSent(Text* txt)*. В этой функции мы вызываем встроенную быструю сортировку предложений, передав в качестве функции-компаратора функцию *compFunc(const void* va, const void* vb)*. Для получения количества гласных в каждом втором слове предложения используется функция *countVowels(Sent* sent)*. В этой функции мы пробегаемся по словам, начиная со второго и пропуская по одному слову, в каждом слове проходимся по всем его символам и сравниваем каждый символ с каждой гласной буквой.

По условию, по этому запросу программа должна удалить из текста все предложения, в которых четное количество слов. Функция *rmEventSent(Text *txt)*, которая принимает указатель на текст. В этой функции перебирается каждое предложение в тексте с конца, и если количество слов четное, тогда предложение удаляется из текста с помощью функции

По условию, функция должна вывести на экран все предложения, в середине которых встречаются слова, состоящие только из прописных букв. Для реализации данной задачи была составлена функция *printSentCenter(Text* txt)*. В этой функции мы проходимся по предложениям в тексте, берем у каждого предложения его слово посередине. Если в предложении четное количество слов, проверяется два слова. В процессе проверки мы проходимся по всем символам данного слова и считаем количество заглавных букв в нем. Если это количество равно длине слова, то его выводим это слово синим, а все остальные слова в предложении – белым. Если в предложении четное количество слов, то такая же проверка выполняется и со вторым словом посередине, и если ни одно из слов не нужно выводить синим – вывод предложения пропускается.

Главная функция

В главной функции инициализируется и считывается текст. Далее из него удаляются повторяющиеся предложения. Для преобразования номеров команд в текст был объявлен массив строк *funcN[5]*. Далее мы считываем символ, чтобы понять какую команду из funcN нам выполнять Если была получена команда под номером 5, осуществляется выход из программы. В конце вызывается функция *freeText(Text *txt)*, чтобы убедиться, что вся динамически выделенная память очистится на любых системах.

ТЕСТИРОВАНИЕ

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define defaultN 5
#define DefaultColor "\033[0m"
#define Blue "\033[46;31m"

// Структуры

typedef struct {
    char *symbols;
    char end;
    int size, memorySize;
} Word;

typedef struct {
    Word *words;
    int size, memorySize;
} Sent;

typedef struct {
    Sent *sents;
    int size, memorySize;
} Text;


void freeW(Word *word) {
    free(word->symbols);
    word->size = 0, word->memorySize = 0;
}

void freeSent(Sent *sentence) {
    for (int i = 0; i < sentence->size; ++i)
        freeW(&sentence->words[i]);
    free(sentence->words);
    sentence->size = 0, sentence->memorySize = 0;
}

void rmSent(Text *txt, int index) {
    if (index < 0 || index >= txt->size)
        return;
    freeSent(&txt->sents[index]);
    memmove(&txt->sents[index], &txt->sents[index + 1], (txt->size - index - 1) * sizeof(Sent));
    txt->sents = realloc(txt->sents, sizeof(Sent) * (--txt->size));
    if (txt->sents == NULL) {
        printf("\033[0;31mCouldn't reallocate memory for sents array in text at %p to size %d\n",
txt, txt->size);
        exit(0);
    }
    txt->memorySize = txt->size;
}

void freeText(Text *txt);

void printWord(Word* word) {
    for(int i = 0; i < word->size; i++)
        printf("%c", word->symbols[i]);
    if (word->end)
        printf("%c", word->end);
```

```
}

void printText(Text *txt){
    for (int i = 0; i < txt->size; i++)
        for (int j = 0; j < txt->sents[i].size; j++){
            printWord(&txt->sents[i].words[j]);
            printf(" ");
        }
    printf("\n");
}

void initW(Word *word) {
    word->size = 0;
    word->end = 0;
    word->memorySize = defaultN;
    word->symbols = malloc(word->memorySize * sizeof(char));
    if (word->symbols == NULL) {
        printf("\033[0;31mCouldn't allocate memory for symbols array in word at %p to size %d\n",
word, word->memorySize);
        exit(0);
    }
}

void appChar(Word *word, char c) {
    if (word->size >= word->memorySize) {
        word->symbols = realloc(word->symbols, sizeof(char) * (word->memorySize += defaultN));
        if (word->symbols == NULL) {
            printf("\033[0;31mCouldn't reallocate memory for symbols array in word at %p to size
%d\n",
                   word, word->memorySize);
            exit(0);
        }
    }
    word->symbols[word->size++] = c;
}

void initS(Sent *sentence) {
    sentence->size = 0;
    sentence->memorySize = defaultN;
    sentence->words = malloc(sentence->memorySize * sizeof(Word));
    if (sentence->words == NULL) {
        printf("\033[0;31mCouldn't allocate memory for words array in sentence at %p to size
%d\n", sentence,
               sentence->memorySize);
        exit(0);
    }
}

int sentEql(Sent *first, Sent *scnd) {
    if (first->size != scnd->size)
        return 0;
    for (int i = 0; i < first->size; ++i) {
        if (first->words[i].size != scnd->words[i].size || first->words[i].end != scnd-
>words[i].end)
            return 0;
        for (int j = 0; j < first->words[i].size; ++j)
            if (tolower(first->words[i].symbols[j]) != tolower(scnd->words[i].symbols[j]))
                return 0;
    }
    return 1;
}

Word *newW(Sent *sentence) {
    if (sentence->size >= sentence->memorySize) {
```

```
        sentence->words = realloc(sentence->words, sizeof(Word) * (sentence->memorySize *= 2));
        if (sentence->words == NULL) {
            printf("\033[0;31mCouldn't reallocate memory for words array in sentence at %p to
size %d\n", sentence,
                   sentence->memorySize);
            exit(0);
        }
    }
    initW(&sentence->words[sentence->size]);
    return &sentence->words[sentence->size++];
}


void removeDoubleSent(Text *txt) {
    for (int i = 0; i < txt->size;i++)
        for (int j = txt->size-1; j > i; --j)
            if (sentEql(&txt->sents[i], &txt->sents[j]))
                rmSent(txt, j);
}



const char *funcN[5] = {
        "Преобразовать предложения так, чтобы каждое первое слово в нем начиналось с заглавной
буквы, а остальные символы были прописными",
        "Отсортировать предложения по сумме количеств гласных букв в каждом втором слове",
        "Удалить все предложения, состоящие из четного количества слов",
        "Вывести на экран все предложения, в которых в середине предложения встречаются слова,
состоящие из прописных букв",
        "Выйти",
};

void editSent(Text *txt){
    for( int i = 0; i < txt->size;i++){
        for( int j = 0; j < txt->sents[i].size ;j++){
            txt->sents[i].words[j].symbols[0] = toupper(txt->sents[i].words[j].symbols[0]);
        }
    }
    printText(txt);
}

char vowels[] = {'a', 'e', 'u', 'i', 'o', 'y'};

int countVowels(Sent* sent) {
    int result = 0;
    for (int i = 1; i < sent->size; i += 2)
        for (int j = 0; j < sent->words[i].size; j++)
            for (int k = 0; k < 6; k++)
                if (sent->words[i].symbols[j] == vowels[k]) {
                    result++;
                    break;
                }
    return result;
}

int compFunc(const void* va, const void* vb) {
    Sent *a = (Sent*) va;
    Sent *b = (Sent*) vb;
    return countVowels(a) - countVowels(b);
}

void sortSent(Text *txt){
    qsort(txt->sents, txt->size, sizeof(Sent), compFunc);
    printText(txt);
```

```
}

void rmEventSent(Text *txt) {
    for (int i = txt->size - 1; i > -1; --i)
        if (txt->sents[i].size % 2 == 0)
            rmSent(txt, i);
    printText(txt);
}


void printSentCenter(Text *txt){
    for(int i = 0; i < txt->size; i++){
        int numbUpper = 0; // количество символов в верхнем рег.
        int k = 0; // количество символов в общем
        int lenString = txt->sents[i].size;
        int indexW;
        if (lenString%2 != 0){
            indexW = lenString/2;
            while(k < txt->sents[i].words[indexW].size){
                if(isupper(txt->sents[i].words[indexW].symbols[k])){
                    numbUpper++;
                }
                k++;
            }
            if(numbUpper == k){
                for(int j = 0; j < indexW; j++){
                    printWord(&txt->sents[i].words[j]);
                    printf(" ");
                }
                printf("%s", Blue);
                printWord(&txt->sents[i].words[indexW]);
                printf("%s", DefaultColor);
                printf(" ");
                for (int j = indexW + 1; j < txt->sents[i].size; j++){
                    printWord(&txt->sents[i].words[j]);
                    printf(" ");
                }
            }
        } else {
            indexW = lenString/2;
            while(k < txt->sents[i].words[indexW-1].size){
                if(isupper(txt->sents[i].words[indexW-1].symbols[k])){
                    numbUpper++;
                }
                k++;
            }
            int blue1 = (numbUpper == k);
            numbUpper = 0, k = 0;
            while(k < txt->sents[i].words[indexW].size){
                if(isupper(txt->sents[i].words[indexW].symbols[k])){
                    numbUpper++;
                }
                k++;
            }
            int blue2 = (numbUpper == k);
            if (blue1 || blue2) {
                for(int j = 0; j < indexW - 1; j++){
                    printWord(&txt->sents[i].words[j]);
                    printf(" ");
                }
                if (blue1)
                    printf("%s", Blue);
                printWord(&txt->sents[i].words[indexW-1]);
                printf("%s", DefaultColor);
```

```
                printf(" ");
                if (blue2)
                    printf("%s", Blue);
                printWord(&txt->sents[i].words[indexW]);
                printf("%s", DefaultColor);
                printf(" ");
                for(int j = indexW + 1; j < txt->sents[i].size; j++){
                    printWord(&txt->sents[i].words[j]);
                    printf(" ");
                }
            }
        }
    }
}

void initText(Text *txt) {
    txt->size = 0;
    txt->memorySize = defaultN;
    txt->sents = malloc(txt->memorySize * sizeof(Sent));
}

Sent *newSent(Text *txt) {
    if (txt->size >= txt->memorySize) {
        txt->sents = realloc(txt->sents, sizeof(Sent) * (txt->memorySize *= 2));
        if (txt->sents == NULL) {
            printf("\033[0;31mCouldn't reallocate memory for sents array in text at %p to size
%d\n", txt, txt->memorySize);
            exit(0);
        }
    }
    initS(&txt->sents[txt->size]);
    return &txt->sents[txt->size++];
}

void readText(Text *result) {
    char c;
    Sent *currSent = newSent(result);
    Word *currentWord = newW(currSent);
    do {
        c = getchar();
        if (c == ' ') {
            if (currentWord->size) {
                currentWord->end = 0;
                currentWord = newW(currSent);
            }
        } else if (c == ',') {
            currentWord->end = ',';
            currentWord = newW(currSent);
        } else if (c == '.') {
            currentWord->end = '.';
            currSent = newSent(result);
            currentWord = newW(currSent);
        } else if (c && c != '\n')
            appChar(currentWord, c);
    } while (c && c != '\n');
    if (c == '\n' && !currentWord->size)
        freeSent(&result->sents[--result->size]);
}

void freeText(Text *txt) {
    for (int i = 0; i < txt->size; ++i)
        freeSent(&txt->sents[i]);
    free(txt->sents);
    txt->size = 0, txt->memorySize = 0;
```

```
}

int main() {
    Text txt;
    initText(&txt);
    printf("Введите текст, состоящий из цифр и латинских букв\n\n");
    readText(&txt);
    printf("\n");
    printf("\nВведенный текст прочитан\n");
    printf("\n");
    removeDoubleSent(&txt);
    printf("\n");
    printf("\nДублирующиеся предложения удалены\n");
    printText(&txt);
    int numb;
    do{
        numb = 0;
        printf("\n");
        for (int i = 0; i < 5; ++i) {
            printf("%d: %s\n", i + 1, funcN[i]);
        }
        while (numb < 1 || numb > 5) {
            printf("Введите номер команды (1-5):\n");
            scanf("%d", &numb);
        }
        numb--;
        switch (numb){
            case 0:
                editSent(&txt);
                printf("Выполнено: %s\n", funcN[numb]);
                break;
            case 1:
                sortSent(&txt);
                printf("Выполнено: %s\n", funcN[numb]);
                break;
            case 2:
                rmEventSent(&txt);
                printf("Выполнено: %s\n", funcN[numb]);
                break;
            case 3:
                printSentCenter(&txt);
                printf("Выполнено: %s\n", funcN[numb]);
                break;
            case 4:
                break;
            default:
                break;
        }
    }while (numb != 4);
    freeText(&txt);
    printf("Работа программы законченафнаф\n");
    return 0;
}
```

ЗАКЛЮЧЕНИЕ

Вывод:

После выполнения программы и проверки всех задач на

работоспособность можно сказать следующее: в ходе выполнения данной работы были изучены основные конструкции, функции стандартных библиотек языка Си, а также структуры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Правила оформления пояснительной записки// se.moevm.http://se.moevm.info/doku.php/courses:programming:report.(дата обращения 18.12.2021)

2. Функции стандартных библиотек Си.//Wikipedia. https://ru.wikipedia.org/wiki/(дата обращения 19.12.2021