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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема «Обработка текстовых данных»

Студентка гр. 1304 _____ Нго Тхи Йен

Преподаватель _____ Чайка К. В

Санкт-Петербург
2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка: Нго Тхи Йен

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 букв и цифр. Длина текста и каждого предложения заранее не известна.

Содержание пояснительной записки:

Предполагаемый объем пояснительной записки:
Не менее 25 страниц.

Дата выдачи задания: 15.10.2021

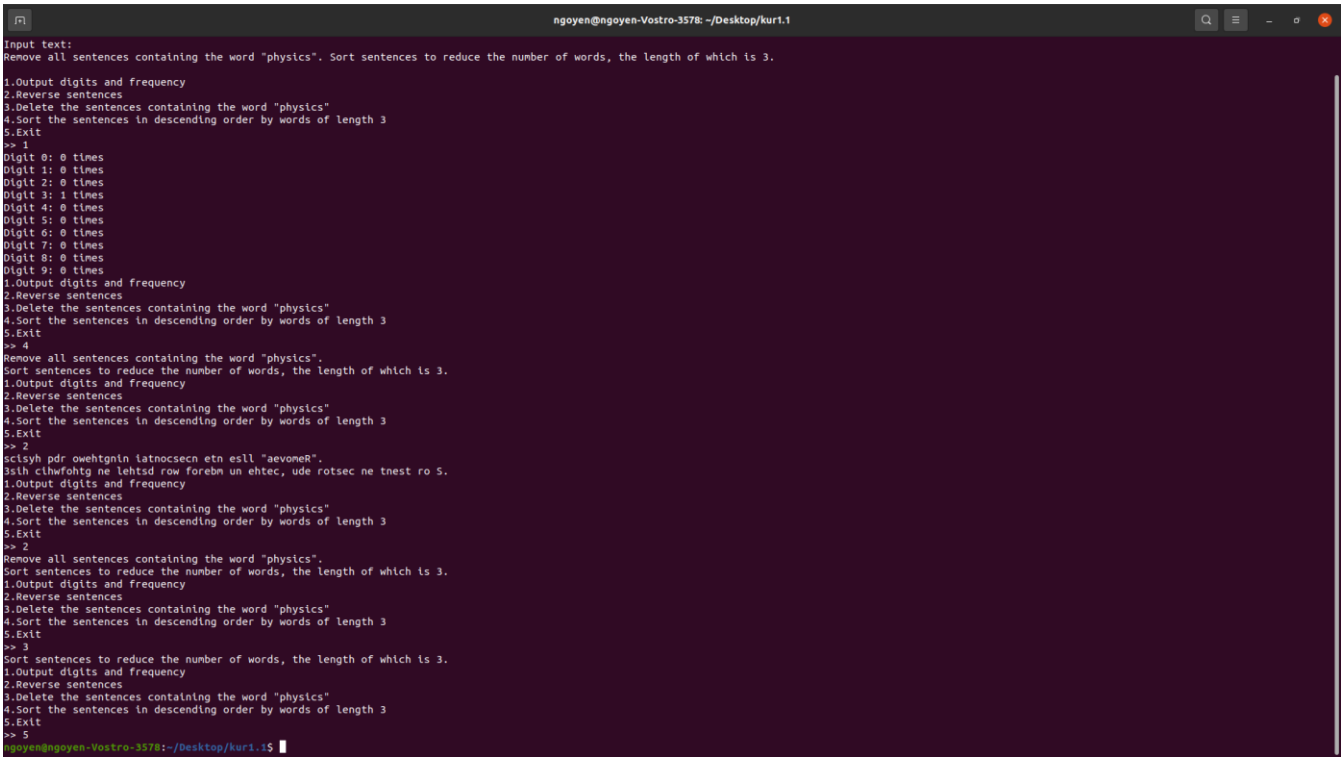
Дата сдачи реферата:

Дата защиты реферата: 14.12.2021

Аннотация

В данной курсовой 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.

Результаты, полученные после прохождения программы:

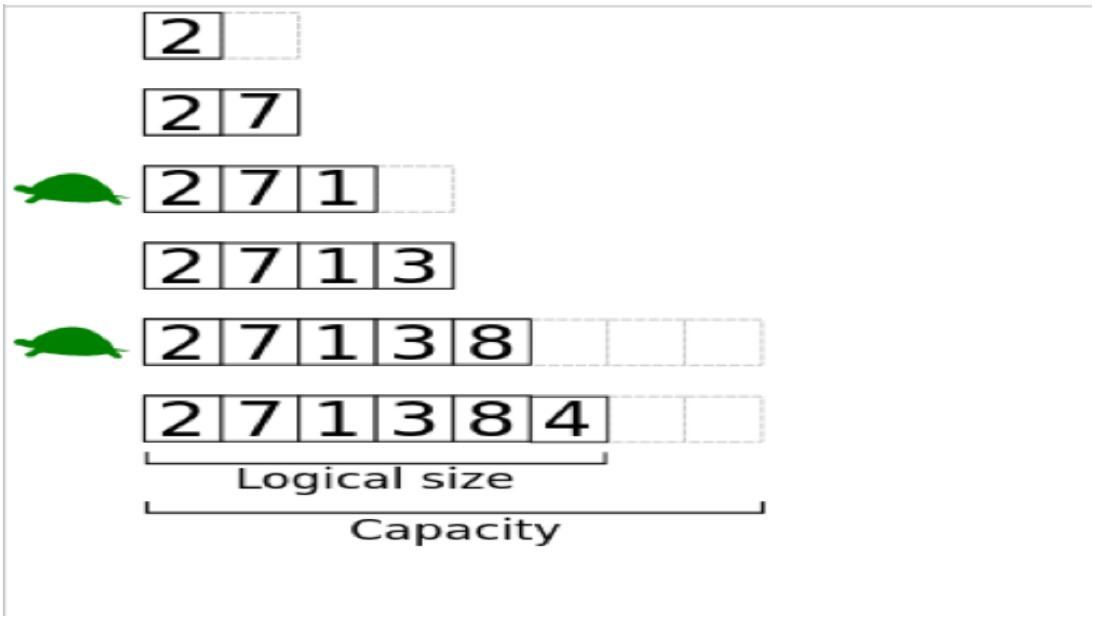


Содержание

Введение

Динамическим называется массив, размер которого может изменяться во время исполнения программы. Возможность изменения размера отличает динамический массив от статического, размер которого задаётся на момент компиляции программы. Для изменения размера динамического массива язык программирования, поддерживающий такие массивы, должен предоставлять встроенную функцию или оператор. Динамические массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать хранимые объёмы данных, а регулировать размер массива в соответствии с реально необходимыми объёмами.

Также иногда к динамическим относят массивы переменной длины, размер которых не фиксируется при компиляции, а задаётся при создании или инициализации массива во время исполнения программы. От «настоящих» динамических массивов они отличаются тем, что для них не предоставляются средства автоматического изменения размера с сохранением содержимого, так что при необходимости программист должен реализовать такие средства самостоятельно.



Цель работы

Целью данной работы- использовать динамического массива строк для обработка текстовых данных.

Задачи данной работы:

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):

1. Вывести список всех цифр встречаемых во всем тексте и их частоту.

2. Преобразовать предложение так, чтобы символы кроме разделительных шли в обратном порядке. Например, для строки “abc defg.” результатом будет “gfe dcba.”.

3. Удалить все предложения в которых встречается слово “physics”.

4. Отсортировать предложения по уменьшению количества слов длина которых равняется 3.

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.

Структура элемента списка (тип - Sentence)

- str - строка неизвестной длины , предложение.
- size - целое число, длина строки.

```
struct Sentence{
    char *str;
    int size;
    int n_word;
```

```
};
```

Структура элемента списка (тип - Text)

- sent - строка неизвестной длины, текст.
- n - целое число, длина текст.

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

Функция readText() для читать текст.

```
struct Text readText(){ // (!) возвращаем не указатель (!)
        int size = MEM_STEP;
        struct Sentence** text= malloc(size*sizeof(struct Sentence*));
        int n=0;
        struct Sentence* temp;
        int nlcount = 0;
        do{
                temp = readSentence();
                if(temp->str[0] == '\n')
                        nlcount++;
                        // memory leack
                else{
                        nlcount = 0;
                        text[n] = temp;
                        n++;
                //      puts(temp->str);
                }
                if(n >= size)
                {
```

```
                size += MEM_STEP;
                text = realloc(text,size*sizeof(struct Sentence*));
            }

        }while(nlcount<2);
        struct Text txt;
        txt.size = size;
        txt.sents = text;
        txt.n = n;
        return txt;
}
```

Функция readSentence() для читать предложение.

```
struct Sentence* readSentence(){
        int size = MEM_STEP;
        char *buf = malloc(size*sizeof(char));
        int n=0;
        char temp = ' ';
        do{
            if(n >= size-2){
                char *t = realloc(buf, size+MEM_STEP);
                if(!t){
                    printf("Error:");
                    return NULL;/* ERROR */}
                size+=MEM_STEP;
                buf = t;
            }
            do
                temp = getchar();
            while(temp == ' ' && n == 0);
            buf[n]= temp;
            n++;
        }while(temp!='\n' && temp!='.' && temp!='!');
        buf[n]= '\0';
        struct Sentence *sentence = malloc(sizeof(struct Sentence));
```

```
        sentence->str = buf;   //(*sentence).str = buf;
        sentence->size = size;
        return sentence;
}
```

Функция deleteDuplicateSen() для удалить все повторно встречающиеся предложения.

```
void deleteDuplicateSen(struct Sentence**sent,int n) // de...(text.sent,text.n)
{
        for(int i = 0; i < n-1; i++){
                for(int j = i + 1; j < n; j++){
                        if(sent[i]->str != NULL && sent[j]->str != NULL){
                                if (isDuplicate(sent[i]->str, sent[j]->str))
                                {
                                        //delete sent
                                        char * p = sent[j]->str;
                                        sent[j]->str = NULL;
                                        free(p);
                                }
                        }
                }
        }
}
```

Функции isDuplicate() для найти все повторно встречающиеся предложения.

```
int isDuplicate(char * str1,char* str2)
{
        int leng1 = strlen(str1);
        int leng2 = strlen(str2);
        char *copystr1 = (char*)malloc(leng1*sizeof(char));
        char *copystr2 = (char*)malloc(leng2*sizeof(char));
        if(!copystr1)
                return 0;
        if(!copystr2)
                return 0;
```

```
        strcpy(copystr1,str1);
        strcpy(copystr2,str2);
        tolowerStr(copystr1);
        tolowerStr(copystr2);
        if(strcmp(copystr1,copystr2) == 0)
        {
                free(copystr1);
                free(copystr2);
                return 1;
        }
        free(copystr1);
        free(copystr2);
        return 0;
}
```

Функции count_digits_sentence() для подсчет всех цифр встречаемых во предложение и их частоту.

```
void count_digits_sentence(char* frequency,char* str)
{
        int k;
        if(str == NULL)
                return;
   int n = strlen(str);
   for(int i = 0;i < n;i++)
                if(isdigit(str[i]) != 0)
                {
                        k = str[i] - '0';
                        frequency[k]+=1;
                }
}
```

Функции count_digits_text() для вывести список всех цифр встречаемых во всем тексте и их частоту.

```
void count_digits_text(struct Sentence**sent,int n)
{
   char* frequency = (char*)malloc(BASE*sizeof(char));
   // input to array frequency
        if(!frequency)
                return;
   for(int i = 0; i < BASE; i++)
      frequency[i] = 0;
   for(int i = 0; i < n; i++)
      count_digits_sentence(frequency,sent[i]->str);
   // output
   for (int i = 0; i < BASE; i++)
      printf("Digit %d: %d times\n",i,frequency[i]);
        free(frequency);
        return;
}
```

Функции swap_sen() для преобразовать предложение так, чтобы символы кроме разделительных шли в обратном порядке.

```
char* swap_sen(char* str,int n)
{
        if(str == NULL)
                return NULL;
        char *newstr = (char*)malloc(n*sizeof(char));
        int i = strlen(str) - 1;
        int q = 1;
        int j = 0;
        for(int k = 0; k < n ; k++)
        {
                if(isalnum(str[k]))
                        newstr[k] = 'a';
                else
                        newstr[k] = str[k];
        }
        while(j<n) // turing
        {
                switch(q)
                {
                        case 1:
                        {
                                if(isalnum(str[i]))
                                        q = 2;
```

```
                        i--;
                        break;
                }
                case 2:
                {
                        if(isalnum(newstr[j]))
                        {
                                newstr[j] = str[i+1];
                                j++;
                                q = 1;
                        }
                        else
                        {
                                j++;
                        }
                        break;
                }
        }
    }
    free(str);
    return newstr;
}
```

Функции swap_text() для преобразовать предложения так, чтобы символы кроме разделительных шли в обратном порядке.

```
void swap_text(struct Sentence**sent,int n)
{
    for(int i = 0;i < n;i++)
        sent[i]->str = swap_sen(sent[i]->str,sent[i]->size);
    return;
}
```

Функции delete_sentences() для удалить все предложения в которых встречается слово “physics”.

```
void delete_sentences(struct Sentence**sent,int n)
{
  for(int i = 0; i < n;i++)
  if(is_substring_sentence(sent[i]->str,"physics"))
  {   //delete
     char * p = sent[i]->str;
        sent[i]->str = NULL;
        free(p);
  }
  return;
}
```

Функции count_NumWord_turing() для подсчет слов длина которых равняется 3 в

предложение.

```
int count_NumWord_turing(char* str) // length  = 3
{
	int pre_q = 0;
	int q = 0;
	int i = 0;
	int start_word = 0;
	int end_word;
	int index_word = -1;
	int count = 0;
	int n  = strlen(str);
	while(i < n) // turing
	{
		switch(q)
		{
			case 0:
			{
				if(isalnum(str[i]))
					q = 1;
				else
					i++;
				pre_q = 0;
				break;
			}
			case 1:
			{
				if(isalnum(str[i]))
					i++;
				else
				{
					q = 2;
				}
				pre_q = 1;
				break;
			}
			case 2:
			{
				if(isalnum(str[i]))
					q = 1;
				else
					i++;
				pre_q = 2;
				break;
			}
		}
		if(pre_q != 1 && q == 1)
		{
			start_word = i;

		}
```

```
                if(pre_q == 1 && q == 2)
                {
                        end_word = i-1;
                        if(end_word - start_word == 2)
                        {
                                count ++;
                        }
                }
        }
        return count;
}
```

Функции bubbleSort() для отсортировать предложения по уменьшению количества.

```
void bubbleSort(struct Sentence**sent,int n)
{
        int i, j;
        int haveSwap = 0;
        char* temp;
        int newN;
        for (i = 0; i < n-1; i++){
      haveSwap = 0;
      for (j = 0; j < n-i-1; j++){
        if (sent[j]->n_word < sent[j+1]->n_word){
                                // change number of vowel;
                                newN = sent[j]->n_word;
                                sent[j]->n_word = sent[j+1]->n_word;
                                sent[j+1]->n_word = newN;
                                // change index sentences;
                                temp = sent[j]->str;
                                sent[j]->str = sent[j+1]->str;
                                sent[j+1]->str = temp;
           haveSwap = 1;
        }
      }
      if(haveSwap == 0){
```

```
            break;
        }
    }
            return;
}
```

Функции count_Num_sen() для подсчет слов длина которых равняется 3 в предложения.

```
void count_Num_sen(struct Sentence**sent,int n)
 {
        for(int i = 0;i < n;i++)
        {
                if(sent[i]->str)
                        sent[i]->n_word = count_NumWord_turing(sent[i]->str);
                else
                        sent[i]->n_word = 0;
        }
        return;
 }
```

Функции sortSents() для отсортировать предложения по уменьшению количества слов длина которых равняется 3.

```
void sortSents(struct Sentence**sent,int n)
{
        count_Num_sen(sent,n);
        bubbleSort(sent,n);
        return;
}
```

Главная функция

```
int main(){
        int choose = 0;
        printf("Input text:\n");
        struct Text text = readText();
        deleteDuplicateSen(text.sents,text.n);
```

```
while(1)
{
	printf("1.Output digits and frequency\n");
	printf("2.Reverse sentences\n");
	printf("3.Delete the sentences containing the word \"physics\" \n");
	printf("4.Sort the sentences in descending order by words of length 3\n");
	printf("5.Exit\n");
	printf(">> ");
	scanf("%d",&choose);
	switch(choose){
		case 1 :
		{
	count_digits_text(text.sents,text.n);
			break;
		}
		case 2:
		{
	swap_text(text.sents,text.n);
			printfText(text.sents,text.n);
			break;
		}
		case 3:
		{
	Delete_sentences(text.sents,text.n);
	printfText(text.sents,text.n);
			break;
		}
		case 4:
		{
			sortSents(text.sents,text.n);
			printfText(text.sents,text.n);
			break;
		}
		case 5:
		{
```

```
                    return 0;
                }
        default:
        return 0;
            }
        }
        return 0;
}
```

Исходный код программы

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MEM_STEP 5
#define BASE 10

struct Sentence{
	char *str;
	int size;
	int n_word;
};
struct Text{
	struct Sentence **sents;
	int n;
	int size;
};
// read text
struct Text readText();
struct Sentence* readSentence();
// delete duplicate sentence (sen)
void deleteDuplicateSen(struct Sentence**sent,int n);
int isDuplicate(char * str1,char* str2);
void tolowerStr(char *str);
// count digits
void count_digits_sentence(char* frequency,char* str);
void count_digits_text(struct Sentence**sent,int n);
//

// delete sentences
int is_substring_sentence(char *str, char* substr);
void delete_sentences(struct Sentence**sent,int n);

//4
void count_Num_sen(struct Sentence**sent,int n);
//0
struct Text readText(){ // (!) возвращаем не указатель (!)
	int size = MEM_STEP;
	struct Sentence** text= malloc(size*sizeof(struct Sentence*));
	int n=0;
	struct Sentence* temp;
	int nlcount = 0;
	do{
		temp = readSentence();
		if(temp->str[0] == '\n')
			nlcount++;
			// memory leack
		else{
```

```
                nlcount = 0;
                text[n] = temp;
                n++;
        //      puts(temp->str);
        }
        if(n >= size)
        {
                size += MEM_STEP;
                text = realloc(text,size*sizeof(struct Sentence*));
        }

    }while(nlcount<2);
    struct Text txt;
    txt.size = size;
    txt.sents = text;
    txt.n = n;
    return txt;
}

struct Sentence* readSentence(){
    int size = MEM_STEP;
    char *buf = malloc(size*sizeof(char));
    int n=0;
    char temp = ' ';
    do{
        if(n >= size-2){
            char *t = realloc(buf, size+MEM_STEP);
            if(!t){
                printf("Error:");
                return NULL;/* ERROR */}
            size+=MEM_STEP;
            buf = t;
        }
        do
            temp = getchar();
        while(temp == ' ' && n == 0);
        buf[n]= temp;
        n++;
    }while(temp!='\n' && temp!='.' && temp!='!');
    buf[n]= '\0';
    struct Sentence *sentence = malloc(sizeof(struct Sentence));
    sentence->str = buf;   //(*sentence).str = buf;
    sentence->size = size;
    return sentence;
}

// 0 Найти и удалить все повторно встречающиеся предложения

void deleteDuplicateSen(struct Sentence**sent,int n) // de...(text.sent,text.n)
{
    for(int i = 0; i < n-1; i++){
        for(int j = i + 1; j < n; j++){
```

```
                        if(sent[i]->str != NULL && sent[j]->str != NULL){
                                if (isDuplicate(sent[i]->str, sent[j]->str))
                                {
                                        //delete sent
                                        char * p = sent[j]->str;
                                        sent[j]->str = NULL;
                                        free(p);
                                }
                        }
                }
        }
}

int isDuplicate(char * str1,char* str2)
{
        int leng1 = strlen(str1);
        int leng2 = strlen(str2);
        char *copystr1 = (char*)malloc(leng1*sizeof(char));
        char *copystr2 = (char*)malloc(leng2*sizeof(char));
        if(!copystr1)
                return 0;
        if(!copystr2)
                return 0;
        strcpy(copystr1,str1);
        strcpy(copystr2,str2);
        tolowerStr(copystr1);
        tolowerStr(copystr2);
        if(strcmp(copystr1,copystr2) == 0)
        {
                free(copystr1);
                free(copystr2);
                return 1;
        }
        free(copystr1);
        free(copystr2);
        return 0;
}

void tolowerStr(char *str)
{
        int n = strlen(str);
        for(int i = 0; i < n; i++)
                str[i] = tolower(str[i]);
        return;
}

// 1 Вывести список всех цифр встречаемых во всем тексте и их частоту.

void count_digits_sentence(char* frequency,char* str)
{
   int k;
        if(str == NULL)
```

```
                return;
    int n = strlen(str);
        //printf("%d\n",n);
    for(int i = 0;i < n;i++)
                if(isdigit(str[i]) != 0)
                {
                        k = str[i] - '0';
                        frequency[k]+=1;
                }

}

void count_digits_text(struct Sentence**sent,int n)
{
    char* frequency = (char*)malloc(BASE*sizeof(char));
    // input to array frequency
        if(!frequency)
                return;
    for(int i = 0; i < BASE; i++)
       frequency[i] = 0;
    for(int i = 0; i < n; i++)
       count_digits_sentence(frequency,sent[i]->str);
    // output
    for (int i = 0; i < BASE; i++)
       printf("Digit %d: %d times\n",i,frequency[i]);
        free(frequency);
        return;
}

// 2 Преобразовать предложение так, чтобы символы кроме разделительных шли в обратном порядке

char* swap_sen(char* str,int n)
{
        if(str == NULL)
                return NULL;
        char *newstr = (char*)malloc(n*sizeof(char));
        int i = strlen(str) - 1;
        int q = 1;
        int j = 0;
        for(int k = 0; k < n ; k++)
        {
                if(isalnum(str[k]))
                        newstr[k] = 'a';
                else
                        newstr[k] = str[k];
        }
        while(j<n) // turing
        {
                switch(q)
                {
                        case 1:
                        {
```

```
                        if(isalnum(str[i]))
                                q = 2;
                        i--;
                        break;
                }
                case 2:
                {
                        if(isalnum(newstr[j]))
                        {
                                newstr[j] = str[i+1];
                                j++;
                                q = 1;
                        }
                        else
                        {
                                j++;
                        }
                        break;
                }
        }
    }
    free(str);
    return newstr;
}

void swap_text(struct Sentence**sent,int n)
{
    for(int i = 0;i < n;i++)
        sent[i]->str = swap_sen(sent[i]->str,sent[i]->size);
    return;
}

//3 Удалить все предложения в которых встречается слово “physics”

int is_substring_sentence(char *str, char* substr)
{
    if(str == NULL)
        return 0;
  char* p = strstr(str,substr);
  return p != NULL ? 1 : 0;
}

void delete_sentences(struct Sentence**sent,int n)
{
  for(int i = 0; i < n;i++)
  if(is_substring_sentence(sent[i]->str,"physics"))
  {   //delete
     char * p = sent[i]->str;
        sent[i]->str = NULL;
        free(p);
  }
  return;
```

```
}
//4 Отсортировать предложения по уменьшению количества слов длина которых равняется 3

int count_NumWord_turing(char* str) // length  = 3
{
	int pre_q = 0;
	int q = 0;
	int i = 0;
	int start_word = 0;
	int end_word;
	int index_word = -1;
	int count = 0;
	int n  = strlen(str);
	while(i < n) // turing
	{
		switch(q)
		{
			case 0:
			{
				if(isalnum(str[i]))
					q = 1;
				else
					i++;
				pre_q = 0;
				break;
			}
			case 1:
			{
				if(isalnum(str[i]))
					i++;
				else
				{
					q = 2;
				}
				pre_q = 1;
				break;
			}
			case 2:
			{
				if(isalnum(str[i]))
					q = 1;
				else
					i++;
				pre_q = 2;
				break;
			}
		}
		if(pre_q != 1 && q == 1)
		{
			start_word = i;

		}
```

```
                if(pre_q == 1 && q == 2)
                {
                        end_word = i-1;
                        if(end_word - start_word == 2)
                        {
                                count ++;
                        }
                }
        }
        return count;
}

void bubbleSort(struct Sentence**sent,int n)
{
        int i, j;
        int haveSwap = 0;
        char* temp;
        int newN;
        for (i = 0; i < n-1; i++){
      haveSwap = 0;
      for (j = 0; j < n-i-1; j++){
        if (sent[j]->n_word < sent[j+1]->n_word){
                                        // change number of vowel;
                                        newN = sent[j]->n_word;
                                        sent[j]->n_word = sent[j+1]->n_word;
                                        sent[j+1]->n_word = newN;
                                        // change index sentences;
                                        temp = sent[j]->str;
                                        sent[j]->str = sent[j+1]->str;
                                        sent[j+1]->str = temp;
           haveSwap = 1;
        }
      }
      if(haveSwap == 0){
        break;
      }
   }
        return;
}

void count_Num_sen(struct Sentence**sent,int n)
{
        for(int i = 0;i < n;i++)
        {
                if(sent[i]->str)
                        sent[i]->n_word = count_NumWord_turing(sent[i]->str);
                else
                        sent[i]->n_word = 0;
        }
        return;
}
```

```
void sortSents(struct Sentence**sent,int n)
{
	count_Num_sen(sent,n);
	bubbleSort(sent,n);
	return;
}

void printfText(struct Sentence**sent,int n)
{
	for(int i = 0; i < n; i++)
	if(sent[i]->str != NULL)
		puts(sent[i]->str);
}

int main(){
	int choose = 0;
	printf("Input text:\n");
	struct Text text = readText();
	deleteDuplicateSen(text.sents,text.n);
	while(1)
	{
		printf("1.Output digits and frequency\n");
		printf("2.Reverse sentences\n");
		printf("3.Delete the sentences containing the word \"physics\" \n");
		printf("4.Sort the sentences in descending order by words of length 3\n");
		printf("5.Exit\n");
		printf(">> ");
		scanf("%d",&choose);
		switch(choose){
			case 1 :
			{
		count_digits_text(text.sents,text.n);
				break;
			}
			case 2:
			{
		swap_text(text.sents,text.n);
				printfText(text.sents,text.n);
				break;
			}
			case 3:
			{
		delete_sentences(text.sents,text.n);
		printfText(text.sents,text.n);
				break;
			}
			case 4:
			{
				sortSents(text.sents,text.n);
				printfText(text.sents,text.n);
				break;
			}
```

```
                        case 5:
                        {
                                return 0;
                        }
                default:
            return 0;
                }
        }
    return 0;
}
```

Заключение

В ходе данной курсовой работы я узнала, как работать со динамическим массивдля решения цели задачи и получала результаты.