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. 9384 _____ Анкудинов К.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Анкудинов Константин

Группа 9384

Тема работы: Обработка текста

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:

“Содержание”, “Введение”, “Задание работы”, “Ход выполнения работы”, “Заключение”, “Список использованных источников”.

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 15.10.2019

Дата сдачи реферата: 21.12.2019

Дата защиты реферата: 21.12.2019

Студент гр. 9384 _____ Анкудинов К.В.

Преподаватель _____ Жангиров Т.Р.

АННОТАЦИЯ

Необходимо ввести текст, предложения в которых разделены точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских букв и цифр. Длина текста и каждого предложения заранее не известна. Сам текст сохраняется в динамический массив строк. Задачи, поставленные для написания программы, представлены в виде четырех функций, которые форматируют текст. В программе реализован элементарный интерфейс общения с пользователем и выполнение запрашиваемых им действий.

Пример работы программы приведен в приложении А.

SUMMARY

You must enter text in which sentences are separated by a period. Sentences - a set of words separated by a space or comma, words - a set of Latin letters and numbers. The length of the text and each sentence is not known in advance. The text itself is saved in a dynamic array of strings. The tasks set for writing the program are presented in the form of four functions that format the text. The program implements an elementary interface for communicating with the user and performing the actions requested by him.

An example of the program is given in Appendix A.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

ЦЕЛЬ

Создать стабильную программу, производящую выбранную пользователем обработку данных. Реализация программы должна содержать работу с динамически выделенной памятью и использование функций стандартных библиотек языка Си. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается. Получить практические навыки работы с символьными строками. Научиться выполнять основные функции с ними.

Разработка велась на базе операционной системы Linux и Mak OS с помощью командной строки.

В результате была создана программа, считывающая вводимый пользователем с консоли или файла текст, выводящая меню со списком доступных функций пользователю на экран и выполняющая выбранную из них. По окончании работы программа выводит обработанный массив предложений – текст и меню для завершения программы.

1. ЗАДАНИЕ РАБОТЫ

Вариант 23

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):

Удалить все четные по счету предложения в которых четное количество слов.

Отсортировать все слова в предложениях по возрастанию количества букв в верхнем регистре в слове.

Заменить все слова в тексте длина которых не более 3 символов на подстроку “Less Then 3”.

Найти в каждом предложении строку максимальной длины, которая начинается и заканчивается цифрой. Вывести найденные подстроки по убыванию длины подстроки.

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.

2.Выполнение работы

2.1 Подключение библиотек

Подключаем стандартную библиотеку <stdio.h>, для работы с такими функциями, как: printf(), scanf(). Подключаем библиотеку <stdlib.h>, для работы с динамической памятью, а также чтобы воспользоваться функцией qsort(). Далее подключаем библиотеку <string.h>, для работы со строками и использования функций: strstr(), strcasecmp(), strlen(), strchr(). также библиотеку <ctype.h> для использования функций isspace() и isdigit(). также библиотеку <stdbool.h> для работы с переменными типа bool. <errno.h> для написания ошибок.

2.2. Объявление всех необходимых функций для решения данного задания

Объявляем следующие функции, чтобы в дальнейшем их определить и использовать в главной функции программы main():

void Usage(); - Отобразить текст с описанием использования программы

void ShowMenu(); - Отобразить меню

void ToLowerString(char *String, size_t Size); //Привести строку к нижнему регистру

size_t WordCount(const char *String, size_t Size); //Получить количество слов в строке

void RemoveBeginSpace(char *String); //Удалить пробелы в начале строки

void RemoveCharFromString(char *String, char Char); //Удалить символ в строке

bool IsLetter(char Char); //Является ли символ буквой

char** CreateArraySentence(size_t *SentenceCount); //Создать массив предложений

char* ReplaceWord(const char *s, const char *oldW, const char *newW); //Замена строки

int GetSubString(char *source, char *target, int from, int to); //Получить подстроку

int PStrCmp(const void* A, const void* B);

void ReadFile(const char *FilePath); //Чтение файла

void SelectFunction(); //Выбор функции

void Function1(); - 1 Операция выбранная пользователем

void Function2(); - 1 Операция выбранная пользователем

void Function3(); - 1 Операция выбранная пользователем

void Function4(); -1 Операция выбранная пользователем
void Function5(); - Повторный вызов меню
void Function6(); - Выход из программы

2.3. Описание главной функции main

Функция принимает на вход два аргумента. Если первый элемент отсутствует, выводится ошибка и программа завершается с вызовом справки. Если и второй аргумент присутствует, то происходит проверка введенного содержимого. Если подан текст – он записывается как текст, если подана ссылка на файл – вызывается ReadFile. Иначе выводим ошибки. Далее, ели ни одно условие не выполнено, выводится справка, программа заканчивает работу. Если ошибок нет, то запускается меню.

2.4. Описание функции Usage

Функция выводит текст справки на Английском языке, в случае если не правильно передан текст на вход программе. Example: Application.exe -f C:\\file_with_text.txt.

Example: Application.exe -m \"Hello, world! I am a console program written in the C programming language.

Example: Application.exe -h Show this help text.

2.5. Описание функции ShowMenu

Функция выводит текст меню на Английском языке, в случае если текст введен правильно и не произведен выход из программы. Menu:

1 - Delete all sentences that are even in the account, in which an even number of words.

2 - Sort all words in sentences in ascending order of upper case letters in a word.

3 - Replace all words in the text with a length of no more than 3 characters by the substring \"Less Then 3\".

4 - Find in each sentence a string of maximum length that begins and ends with a digit. Display the found substrings in descending order of the length of the substring.

5 - Show text. personal initiative :)

6 - Close application.

Select menu item (1, 2, 3, 4, 5 or 6)

Item:

2.6. Описание функции ToLowerString

Функция переводит символ в нижний регистр. Принимая на вход строку и её длину .

В случае если находит символ в верхнем регистре с помощью tolower меняет регистр.

2.7. Описание функции RemoveBeginSpace

Функция удаляет пробелы в начале предложения принимая на вход строку.

2.8. Описание функции RemoveCharFromString

Функция удаляет преданный символ в переданной строке.

Происходит проход по строке до \0 и удаление встреченного не желательного символа

2.9. Описание функции CreateArraySentence

Функция создает массив предложений

На вход принимаем длину и текст. С помощью использования буфера считаем количество предложений с точкой. Далее выделяет память для каждого предложения и запускает функцию RemoveBeginSpace. В случае, если произошла ошибка, выводится текст ошибки. В конце очищаем буфер и возвращаем массив предложений.

2.10. Описание функции ReplaceWord

Функция, которая принимает на вход строку ее конец и начало. Проходит строку до конца, при этом производит подсчет длины предложений. Создает результирующую строку, которая ведет на вывод.

2.11. Описание функции GetSubString

Функция занимается нахождением строки в строке, принимая на вход предложение и требуемые данные для поиска. Возвращает результирующую строку, которая была создана динамически.

2.12. Описание функции ReadFile

Функция принимает на вход указатель на файл. В случае, если файл открыт успешно переходит в конец файла и считает его размер. Затем выделяет необходимую память. И проверяет на валидность файла и ссылки.

2.13. Описание функции SelectFunction

Обработка меню. Позволяет пользователю ввести определенную операцию. Далее с помощью switch запускает одну из функций (см функцию ShowMenu()). В случае, если пользователь ввел текст выводится ошибка.

2.14. Описание функции Function1()

Функция удаляет все четные по счету предложения в которых четное количество слов с помощью функций CreateArraySentence обходя массив и удаляя каждое второе предложение. В конце очищает память.

2.15. Описание функции Function2()

Функция сортирует все слова в предложениях по возрастанию количества букв в верхнем регистре в слове. Сортируя с помощью qsort и переводит символы в верхний регист с помощью соотвествующей функции. В конце очищает память.

2.16. Описание функции Function3()

Функция заменяет все слова в тексте длина которых не более 3 символов на подстроку “Less Then 3”. Для этого создает буфер и считает количество символов. Затем очищает выделенную память и выводит.

2.17. Описание функции Function4()

Функция находит в каждом предложении строку максимальной длины, которая начинается и заканчивается цифрой. Вывести найденные подстроки по

убыванию длины подстроки. Для этого обходит все строки и вызывает функцию проверки подстроки далее сортирует их и выводит. В конце очищает память.

2.18. Описание функции `Function5()`

Выводит текст и перезапускает меню.

2.19. Описание функции `Function6()`

Завершает работу программы

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы были применены на практике приемы работы с динамической памятью, стандартными библиотеками языка Си. Была создана стабильная программа, выполняющая функции, указанные в задании и выбранные пользователем. Программа осуществляет взаимодействие с пользователем через примитивный консольный интерфейс и корректно обрабатывает различные виды корректных входных данных (в соответствии с условием)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. и Ритчи Д. Язык программирования Си. М.: Вильямс, 1978 288 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

ФАИЛ MAIN.C:

```
//---------------------------------------------------------------------------
#pragma warning (disable: 4127) //для "while(true)"
#pragma warning (disable: 4717) //для функции "SelectFunction"
//---------------------------------------------------------------------------
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <errno.h>
#include <stdbool.h>
//---------------------------------------------------------------------------
char *Text = NULL; //Входной текст
size_t TextSize = 0; //Размер входного текста
//---------------------------------------------------------------------------
void Usage(); //Отобразить текст с описанием использования программы
void ShowMenu(); //Отобразить меню
void ToLowerString(char *String, size_t Size); //Привести строку к нижнему регистру
size_t WordCount(const char *String, size_t Size); //Получить количество слов в строке
void RemoveBeginSpace(char *String); //Удалить пробелы в начале строки
void RemoveCharFromString(char *String, char Char); //Удалить символ в строке
bool IsLetter(char Char); //Является ли символ буквой
char** CreateArraySentence(size_t *SentenceCount); //Создать массив предложений
char* ReplaceWord(const char *s, const char *oldW, const char *newW); //Замена строки
int GetSubString(char *source, char *target, int from, int to); //Получить подстроку
int PStrCmp(const void* A, const void* B);
void ReadFile(const char *FilePath); //Чтение файла
void SelectFunction(); //Выбор функции
//---------------------------------------------------------------------------
void Function1();
void Function2();
```

```
void Function3();
void Function4();
void Function5();
void Function6();
//---------------------------------------------------------------------------
int main(int argc, char **argv)
{
   if (argc == 1) //Если агументы не указанны
   {
      printf("Error: not specified argument.\n");
      Usage();
      return EXIT_FAILURE;
   }
   char *Argument = argv[1];
   ToLowerString(Argument, strlen(Argument));
   if (strcmp(Argument, "-m") == 0) //Если текст передается в виде аргумента
   {
      Text = argc == 3 ? argv[2] : NULL;
      if (Text) //Если текст не был передан в качестве значения аргумента
      {
         TextSize = strlen(Text);
      }
      else
      {
         printf("Error: not specified argument value (text).");
      }
   }
   else if (strcmp(Argument, "-f") == 0) //Если в виде аргумента передается путь к файлу - читаем текст из файла и продолжаем
   {
      char *FilePath = argc == 3 ? argv[2] : NULL;
      if (FilePath) //Если путь к файлу был передан в качестве значения аргумент - читаем файл
      {
         ReadFile(FilePath);
```

```
        }
        else //Путь к файлу не был передан в качестве значения аргумента
        {
            printf("Error: not specified argument value (file path).");
        }
    }
    else if (strcmp(Argument, "-h") == 0) //Запрос на отображения инструкции по использованию программы
    {
        Usage();
        return EXIT_SUCCESS;
    }
    else
    {
        printf("Error: invalid argument '%s'\n", Argument);
        return EXIT_FAILURE;
    }
    if (!Text)
    {
        return EXIT_FAILURE;
    }
    SelectFunction();
    return EXIT_SUCCESS;
}
//---------------------------------------------------------------------------
void Usage()
{
    printf("Example: Application.exe -f C:\\file_with_text.txt\n");
    printf("Example: Application.exe -m \"Hello, world! I am a console program written in the C programming language.\"\n");
    printf("Example: Application.exe -h Show this help text.\n");
}
//---------------------------------------------------------------------------
void ShowMenu()
{
```

```
    printf("\nMenu:\n");
    printf("1 - Delete all sentences that are even in the account, in which an even number of words.\n");
    printf("2 - Sort all words in sentences in ascending order of upper case letters in a word.\n");
    printf("3 - Replace all words in the text with a length of no more than 3 characters by the substring \"Less Then 3\".\n");
    printf("4 - Find in each sentence a string of maximum length that begins and ends with a digit. Display the found substrings in descending order of the length of the substring.\n");
    printf("5 - Show text. personal initiative :)\n");
    printf("6 - Close application.\n");
    printf("Select menu item (1, 2, 3, 4, 5 or 6)\n");
    printf("Item: ");



}
//---------------------------------------------------------------------------
void ToLowerString(char *String, size_t Size)
{
    for (size_t i = 0; i < Size; ++i) //Обходим строку
    {
        if (isalpha(String[i]) != 0) //Если текущий символ является буквой - переводим её в нижний регистр
        {
            String[i] = (char)tolower((int)String[i]);
        }
    }
}
//---------------------------------------------------------------------------
size_t WordCount(const char *String, size_t Size)
{
    size_t Result = 0; //Количество найденных слов
    bool PreviousCharLetter = false; //Флаг предыдущего символа (буква или нет)
    for (size_t i = 0; i < Size; ++i) //Обходим строку
    {
        if (IsLetter(String[i])) //Если текущий символ является буквой
```

```
        {
            PreviousCharLetter = true;
        }
        else //Текущий символ не является буквой
        {
            if (PreviousCharLetter) //Но если предыдущий символ был буквой - то считаем что нашли слово и сбрасываем флаг предыдущего символа
            {
                ++Result;
                PreviousCharLetter = false;
            }
        }
        if (i == Size - 1 && PreviousCharLetter) //Последняя итерация и предыдущий символ является буквой (ситуация для последних последних слов в предложении)
        {
            ++Result;
        }
    }
    return Result;
}
//---------------------------------------------------------------------------
void RemoveBeginSpace(char *String)
{
    while (String[0] == ' ')
    {
        memmove(String, String + 1, strlen(String));
    }
}
//---------------------------------------------------------------------------
void RemoveCharFromString(char *String, char Char)
{
    char *Source, *Dest;
    for (Source = Dest = String; *Source != '\0'; Source++)
    {
        *Dest = *Source;
```

```
        if (*Dest != Char)
        {
            Dest++;
        }
    }
    *Dest = '\0';
}
//---------------------------------------------------------------------------
bool IsLetter(char Char)
{
    return (Char >= 65 && Char <= 90) || (Char >= 97 && Char <= 122) ? true : false;
}
//---------------------------------------------------------------------------
char** CreateArraySentence(size_t *SentenceCount)
{
    char **Array = NULL;
    char *Buffer = (char *)malloc(TextSize + 1);
    if (Buffer)
    {
        strcpy(Buffer, Text);
        char *Char = strchr(Buffer, '.');
        while (Char) //Считаем количество предложений
        {
            (*SentenceCount)++;
            Char = strchr(Char + 1, '.');
        }
        size_t Index = 0; //Индекс текущего обрабатываемого предложения
        Array = (char**)malloc(*SentenceCount * sizeof(char*)); //Выделяем память под массив с предложениями
        if (Array) //Выделение памяти прошло успешно
        {
            Char = strtok(Buffer, ".");
            while (Char)
            {
                RemoveBeginSpace(Char); //Удаляем проблемы в начале строки
```

```
            size_t SentenceSize = strlen(Char);
            Array[Index] = (char*)malloc(SentenceSize * sizeof(char) + 1); //Выделяем память под
предложение
            if (Array[Index])
            {
               memmove(Array[Index], Char, SentenceSize);
               Array[Index][SentenceSize] = '\0';
               ++Index;
               Char = strtok(NULL, ".");
            }
            else //Выделение памяти прошло с ошибкой
            {
               printf("Error: memory allocation.\n");
               return NULL;
            }
         }
      }
      else
      {
         printf("Error: memory allocation.\n");
      }
      free(Buffer);
   }
   return Array;
}
//---------------------------------------------------------------------------
char* ReplaceWord(const char *String, const char *OldString, const char *NewString)
{
   int Iterator, Count = 0;
   int NewStringSize = strlen(NewString);
   int OldStringSize = strlen(OldString);
   for (Iterator = 0; String[Iterator] != '\0'; Iterator++)
   {
      if (strstr(&String[Iterator], OldString) == &String[Iterator])
      {
```

```
            Count++;
            Iterator += OldStringSize - 1;
        }
    }
    char *Result = (char *)malloc(Iterator + Count * (NewStringSize - OldStringSize) + 1);
    if (Result)
    {
        Iterator = 0;
        while (*String)
        {
            if (strstr(String, OldString) == String)
            {
                strcpy(&Result[Iterator], NewString);
                Iterator += NewStringSize;
                String += OldStringSize;
            }
            else
            {
                Result[Iterator++] = *String++;
            }
        }
        Result[Iterator] = '\0';
    }
    else //Ошибка выделения памяти
    {
        printf("Error: memory allocation.\n");
    }
    return Result;
}
//---------------------------------------------------------------------------
int GetSubString(char *Source, char *Target, int From, int To)
{
    int Length = 0;
    int Iterator = 0, Jterator = 0;
    while (Source[Iterator++] != '\0')
```

```
    {
        Length++;
    }
    if (From < 0 || From > Length)
    {
        printf("Invalid \'from\' index\n");
        return 1;
    }

    if (To > Length)
    {
        printf("Invalid \'to\' index\n");
        return 1;
    }
    for (Iterator = From, Jterator = 0; Iterator <= To; Iterator++, Jterator++)
    {
        Target[Jterator] = Source[Iterator];
    }
    Target[Jterator] = '\0';
    return 0;
}
//---------------------------------------------------------------------------
int PStrCmp(const void* A, const void* B)
{
    return strcmp(*(const char**)A, *(const char**)B);
}
//---------------------------------------------------------------------------
void ReadFile(const char *FilePath)
{
    FILE *File = fopen(FilePath, "r");
    if (File) //Если файл открыт успешно
    {
        fseek(File, 0, SEEK_END); //Переводим курсор в конец файла
        long FileSize = ftell(File); //Получаем размер файла
        rewind(File); //Возвращаем курсор в исходное положение
```

```
        Text = (char *)malloc(FileSize * sizeof(char)); //Выделяем память равную размеру файла
        if (Text) //Память выделена успешно
        {
            TextSize = fread(Text, sizeof(char), FileSize, File); //Читаем содержимое файла и
помещаем его в память
            Text[TextSize] = '\0';
        }
        else //Ошибка выделения памяти
        {
            printf("Error: memory allocation.\n");
        }
        fclose(File);
    }
    else
    {
        printf("Error open file '%s': %s\n", FilePath, strerror(errno));
    }
}
//---------------------------------------------------------------------------
void SelectFunction()
{
    ShowMenu();
    int SelectedItem = 0;
    scanf("%d", &SelectedItem);
    if (SelectedItem) //Если был введен индекс пункта меню - проверяем его
    {
        switch (SelectedItem)
        {
        case 1: Function1(); break;
        case 2: Function2(); break;
        case 3: Function3(); break;
        case 4: Function4(); break;
        case 5: Function5(); break;
        case 6: Function6(); break;
        default: printf("Error: invalid selected item '%d'\n\n", SelectedItem); break;
```

```
        }
    }
    else //Введенное значение невалидное (скорее всего ввели текст)
    {
        printf("Error: invalid entered value.\n\n");
    }
    SelectFunction();
}
//-------------------------------------------------------------------------
void Function1()
{
    size_t SentenceCount = 0; //Количество предложений
    char **Array = CreateArraySentence(&SentenceCount);
    for (size_t i = 0; i < SentenceCount; ++i) //Обходим все предложения
    {
        bool ShowSentence = i % 2 != 0;
        if (!ShowSentence) //Если текущее предложение чётное - анализируем его
        {
            ShowSentence = WordCount(Array[i], strlen(Array[i])) % 2 != 0; //Если количество слов в предложении чётное - удаляем его. А точнее, просто не выводим в консоль.
        }
        if (ShowSentence)
        {
            printf("%s. ", Array[i]);
        }
        if (i == SentenceCount - 1)
        {
            printf("\n");
        }
    }
    for (size_t i = 0; i < SentenceCount; ++i) //Очищаем память каждого предложения
    {
        free(Array[i]);
    }
    free(Array); //Очищаем память массива
```

```
}
//---------------------------------------------------------------------------
void Function2()
{
   size_t SentenceCount = 0; //Количество предложений
   char **Array = CreateArraySentence(&SentenceCount);
   for (size_t i = 0; i < SentenceCount; ++i) //Обходим все предложения
   {
      char *Buffer = (char *)malloc(strlen(Array[i]) + 1);
      if (Buffer)
      {
         strcpy(Buffer, Array[i]);
         RemoveCharFromString(Buffer, ',');
         size_t Index = 0;
         size_t Count = WordCount(Buffer, strlen(Buffer));
         char **ArrayWord = (char**)malloc(Count * sizeof(char*));
         char *Word = strtok(Buffer, " ");
         while (Word)
         {
            ArrayWord[Index] = (char *)malloc(strlen(Word) + 1);
            strcpy(ArrayWord[Index], Word);
            ++Index;
            Word = strtok(NULL, " ");
         }
         qsort(ArrayWord, Count, sizeof(ArrayWord[0]), PStrCmp); //Сортируем массив слов
         for (size_t j = 0; j < Count; ++j)
         {
            printf("%s\n", ArrayWord[j]);
         }
         for (size_t j = 0; j < Count; ++j)
         {
            free(ArrayWord[j]);
         }
         free(Buffer);
      }
```

```
        else //Ошибка выделения памяти
        {
            printf("Error: memory allocation.\n");
        }
    }
    for (size_t i = 0; i < SentenceCount; ++i) //Очищаем память каждого предложения
    {
        free(Array[i]);
    }
    free(Array); //Очищаем память массива
}
//---------------------------------------------------------------------------
void Function3()
{
    char *Buffer = (char *)malloc(TextSize + 1);
    if (Buffer)
    {
        strcpy(Buffer, Text);
        while (true)
        {
            size_t CurrentWordSize = 0;
            bool PreviousCharLetter = false;
            for (size_t i = 0, c = strlen(Buffer); i < c; ++i)
            {
                if (IsLetter(Buffer[i])) //Если текущий символ буква - сохраняем его
                {
                    PreviousCharLetter = true;
                    ++CurrentWordSize;
                }
                else //Текущий символ не буква - считаем что слово нашли
                {
                    if (PreviousCharLetter && CurrentWordSize && CurrentWordSize <= 3) //Если
предыдущий символ являлся буквой и текущее слово не более 3-х символов
                    {
                        PreviousCharLetter = false;
```

```
                    char Char[128];
                    if (GetSubString(Buffer, Char, i - CurrentWordSize, i - 1) == 0)
                    {
                        char *Temp = ReplaceWord(Buffer, Char, "Less Then 3");
                        free(Buffer);
                        Buffer = (char *)malloc(strlen(Temp) + 1);
                        strcpy(Buffer, Temp);
                        free(Temp);
                        break;
                    }
                }
                CurrentWordSize = 0;
            }
        }
        if (PreviousCharLetter)
        {
            printf("%s\n", Buffer);
            break;
        }
    }
    free(Buffer);
  }
  else //Ошибка выделения памяти
  {
    printf("Error: memory allocation.\n");
  }
}
//---------------------------------------------------------------------------
void Function4()
{
  size_t SentenceCount = 0; //Количество предложений
  char **Array = CreateArraySentence(&SentenceCount);
  for (size_t i = 0; i < SentenceCount; ++i) //Обходим все предложения
  {
    char *Buffer = (char *)malloc(strlen(Array[i]) + 1);
```

```
if (Buffer)
{
   strcpy(Buffer, Array[i]);
   RemoveCharFromString(Buffer, ',');

   size_t Index = 0;
   size_t Count = WordCount(Buffer, strlen(Buffer));
   char **ArrayWord = (char**)malloc(Count * sizeof(char*));
   char *Word = strtok(Buffer, " ");
   while (Word)
   {
      ArrayWord[Index] = (char *)malloc(strlen(Word) + 1);
      strcpy(ArrayWord[Index], Word);
      ++Index;
      Word = strtok(NULL, " ");
   }
   qsort(ArrayWord, Count, sizeof(ArrayWord[0]), PStrCmp); //Сортируем массив слов
   for (size_t j = 0; j < Count; ++j)
   {
      char *w = ArrayWord[j];
      if (isdigit(w[0]) && isdigit(w[strlen(w) - 1]))
      {
         printf("%s\n", w);
      }
   }
   for (size_t j = 0; j < Count; ++j)
   {
      free(ArrayWord[j]);
   }
   free(Buffer);
}
else //Ошибка выделения памяти
{
   printf("Error: memory allocation.\n");
}
```

```
    }
    for (size_t i = 0; i < SentenceCount; ++i) //Очищаем память каждого предложения
    {
        free(Array[i]);
    }
    free(Array); //Очищаем память массива
}
//---------------------------------------------------------------------------
void Function5()
{
    printf("%s\n", Text);
}
//---------------------------------------------------------------------------
void Function6()
{
    exit(EXIT_SUCCESS);
}
```