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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Программирование»

Тема: «Использование указателей»

Студентка гр. 7381 _____ Кревчик А.Б.

Преподаватель _____ Берленко Т. А.

Санкт-Петербург

2017

Цель работы

Познакомиться с указателями, строками, динамической памятью, а также с функциями для работы с ними.

Задание

Написать программу, которая форматирует некоторый текст и выводит результат на консоль.

На вход программе подается текст который заканчивается предложением "Dragon flew away!".

Предложение (кроме последнего) может заканчиваться на:

- . (точка)
- ; (точка с запятой)
- ? (вопросительный знак)

Программа должна изменить и вывести текст следующим образом:

- Каждое предложение должно начинаться с новой строки.
- Табуляция в начале предложения должна быть удалена.
- Все предложения, которые заканчиваются на '?' должны быть удалены.
- Текст должен заканчиваться фразой "Количество предложений до n и количество предложений после m", где n - количество предложений в изначальном тексте (**без учета** терминального предложения "Dragon flew away!") и m - количество предложений в отформатированном тексте (без учета предложения про количество из данного пункта).

* Порядок предложений не должен меняться

* Статически выделять память под текст нельзя

* Пробел между предложениями является разделителем, а не частью какого-то предложения

Основные теоретические положения

Заголовочные файлы, необходимые для создания проекта:

1. **<stdio.h>** - содержит прототип функции «int printf(const char* format [, argument]...);», которая используется для вывода в поток вывода.

Синтаксис:
#include < stdio.h >
int printf (const char *format, ...);

Аргументы:
format – указатель на строку с описанием формата.

Возвращаемое значение:
При успешном завершении вывода возвращается количество выведенных символов. При ошибке возвращается отрицательное число.

2. **<string.h>** - содержит прототип функции «size_t strlen(const char * string);», которая определяет длину строки.

Синтаксис:
#include < string.h >
size_t *strlen (const char *str);

Аргументы:
str – указатель на строку.

Возвращаемое значение:
Количество символов в строке до первого вхождения символа конца строки.

Описание:
Функция strlen вычисляет количество символов в строке до первого вхождения символа конца строки. При этом символ конца строки не входит в подсчитанное количество символов.

2. **<stdlib.h>** - содержит прототипы функций «void* calloc (size_t num, size_t size);» и «void free (void* ptr);», которые динамически выделяют память под массив данных, предварительно инициализируя её нулями и высвобождают динамически выделенную ранее память.

Синтаксис:
#include < stdlib.h >
void * calloc(size_t number, size_t size);

Аргументы:
number -оличество элементов массива, под который выделяется память. size - размер одного элемента в байтах.

Возвращаемое значение:
Указатель на выделенный блок памяти. Тип данных на который ссылается указатель всегда void*, поэтому это тип данных может быть приведен к желаемому типом данных. Если функции не удалось выделить требуемый блок памяти, возвращается нулевой указатель.

Описание:
Функция calloc выделяет блок памяти для массива размером — num элементов, каждый из которых занимает size байт, и инициализирует все свои биты в нулями. В результате выделяется блок памяти размером number * size байт, причём весь блок заполнен нулями.

Синтаксис:
#include < stdlib.h >
void free(void * ptrmem);

Аргументы:
ptrmem – указатель на блок памяти, ранее выделенный функциями malloc, calloc или realloc, которую необходимо высвободить. Если в качестве аргумента передается нулевой указатель, никаких действий не происходит.

Возвращаемое значение: Нет.

Описание:
Функция free освобождает место в памяти. Блок памяти, ранее выделенный с помощью вызова malloc, calloc или realloc освобождается. То есть освобожденная память может дальше использоваться программами или ОС.

3. **<ctype.h>** - содержит прототипы функций «int isdigit(int character);» и «int isspace(int character);», которые возвращают истинное значение true, если аргумент - десятичная цифра, и false(ложь) в других случаях; и возвращают истинное значение true, если аргумент - любой знак пробела, и false(ложь) в других случаях.

Синтаксис:
#include < ctype.h >
int isdigit(int character);

Аргументы:
character -символ для проверки, передается в функцию как значение типа int, или EOF.

Возвращаемое значение:
Значение, отличное от нуля (т.е. истинно), если аргумент функции — это десятичная цифра . Ноль (т.е. ложь), в противном случае.

Описание:
Функция isdigit проверяет аргумент, передаваемый через параметр character, является ли он десятичной цифрой.

Синтаксис:
#include < ctype.h >
int isspace(int character);

Аргументы:
character - символ для проверки, передаётся в функцию как значение типа int, или EOF.

Возвращаемое значение:
Значение, отличное от нуля (т.е. истинно), если аргумент функции — это символ пробела. Ноль (т.е. ложь), в противном случае.

Описание:
Функция isspace проверяет параметр character, является ли он символом пробела.

Вывод

В ходе данной работы были изучены указатели, функции для работы с динамической памятью (malloc, calloc, realloc и free), а также методы работы со строками в C.

Исходный код проекта:

Main.c

```
#include <stdio.h>

#include <stdlib.h>

  int main() {

  int i=-1,m=0,n=0,p,index,l;

  char* s=NULL;

  do {

            i++;

            s=(char* )realloc(s,i+1);

            s[i]=getchar();

            if (s[i]=='.' || s[i]==';' || s[i]=='?')

            n++;

  }

  while(s[i]!='!');

  for(p=0, i=0;p<=n;p++){

            for(index=i;s[i]!='.' && s[i]!=';' && s[i]!='?' && s[i]!='!';i++);

             if(s[i]!='?'){

            l=i;

            for(i=index;s[i]==' ' || s[i]=='\t';i++);

 for(;i<=l;i++)

            putchar(s[i]);

            putchar('\n');

            m++;}

i++;
```

```
}

printf("Количество предложений до %d и количество предложений после
%d\n", n, m-1);

free (s);

 return 0;

}
```

Makefile

```
  all: main.o

        gcc main.o

  main.o: main.c

        gcc –c main.c
```