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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Программирование»
Тема: Использование указателей

Студентка гр. 0382 _____ Ситченко К.С.

Преподаватель _____ Жангиров Т.Р

Санкт-Петербург
2020

Цель работы.

Изучить работу массивов и указателей в языке Си.

Задание.

Вариант 2.

Напишите 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.

Указатель – это переменная, содержащая адрес другой переменной. Синтаксис объявления указателя: <тип_переменной_на_которую_ссылается_указатель>* <название переменной>. Каждая переменная имеет своё место в оперативной памяти, т.е. адрес, по которому к ней обращается программа и может обращаться программист. Унарная операция & даёт адрес объекта. Она применима только к переменным и элементам массива, конструкции вида &(x-1) и &3 являются незаконными. В то же время, у указателя есть унарная операция разыменования *, которая позволяет получить значение ячейки, на которую ссылается указатель.

Если указатели p и q указывают на элементы одного массива, то к ним можно применять операторы отношения ==, !=, <, >= и т. д. Например, отношение вида p < q истинно, если p указывает на более ранний элемент массива, чем q. Любой указатель всегда можно сравнить на равенство и неравенство с нулем. А вот для указателей, не указывающих на элементы одного массива, результат арифметических операций или сравнений не определен. Интересно, что запись вида a[i] эквивалентна записи *(a+i), а поскольку от перестановки слагаемых сумма не меняется, запись вида i[a] имеет право на существование.

Для работы с динамической памятью используются следующие функции:

- malloc (void* malloc (size_t size)) - выделяет блок из size байт и возвращает указатель на начало этого блока
- calloc (void* calloc (size_t num, size_t size)) - выделяет блок для num элементов, каждый из которых занимает size байт и инициализирует все биты выделеного блока нулями
- realloc (void* realloc (void* ptr, size_t size)) - изменяет размер ранее выделенной области памяти на которую ссылается указатель ptr. Возвращает указатель на область памяти, измененного размера.

- free (void free (void* ptr)) - высвобождает выделенную ранее память.

Формально, в языке Си нет специального типа данных для строк, но представление их довольно естественно - строки в языке Си — это массивы символов, завершающиеся нулевым символом ('\0'). Это порождает следующие особенности, которые следует помнить:

- Нулевой символ является обязательным.
- Символы, расположенные в массиве после первого нулевого символа, никак не интерпретируются и считаются мусором.
- Отсутствие нулевого символа может привести к выходу за границу массива.
- Фактический размер массива должен быть на единицу больше количества символов в строке (для хранения нулевого символа)
- Выполняя операции над строками, нужно учитывать размер массива, выделенный под хранение строки.

Строки могут быть инициализированы при объявлении.

Считывание строки можно произвести с помощью:

1. char* fgets(char *str, int num, FILE *stream)

- Безопасный способ (явно указывается размер буфера)
- Считывает до символа переноса строки
- Помещает символ переноса строки в строку-буфер (!)

2. int scanf(const char * format, arg1, arg2, ...argN);

- %s в форматной строке для ввода строки
- Считывает символы до первого символа табуляции (не помещая его в строку)
- Не контролирует размер буфера
- Потенциально опасна

3. char* gets(char* str);

- Не контролирует размер буфера
- Потенциально опасна

Выполнение работы.

Для решения задачи было создано несколько функций.

1. char* get_line() – функция, которая посимвольно считывает каждое предложение текста, удаляет пробелы и табуляцию в начале и записывает в конец символ конца строки. Возвращает исправленное предложение.

2. int has_qm (char* line) – функция, которая проверяет, заканчивается ли предложение на вопросительный знак. Если да, то возвращает 1, иначе – 0.

3. int main() – основная функция, в начале которой объявляется терминальное предложение, выделяется динамическая память под двумерный массив для хранения текста. Далее в цикле *while* выполняются первые две функции, формирование текста, подсчет предложений, оканчивающихся на вопросительный знак проверка памяти, и если ее не хватает, то с помощью *realloc* выделяется новый объем памяти, и сравнение каждого предложения с терминальным, когда оно встречается, цикл завершается. Затем в цикле происходит печать динамического массива, а после – освобождение памяти, которая была под него выделена. В конце печатается предложение, содержащее информацию об изменении текста.

Разработанный программный код см. в приложении А.

Тестирование.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	Integer iaculis a te4llus? Suspendisse quis mi neque7. Sed finibus magna tempus? Class aptenT per cOnubia nostra! Nam 7elementum congue; Donec accumsan vitae lacinia. Dragon flew away!	Suspendisse quis mi neque7. Class aptenT per cOnubia nostra! Nam 7elementum congue; Donec accumsan vitae lacinia. Dragon flew away! Количество предложений до 6 и количество предложений после 4	Программа сработала верно

Выводы.

Была изучена работа массивов и указателей в языке Си.

Разработана программа, которая считывает введенный с клавиатуры текст и записывает его в динамический массив. Также в функциях производится обработка каждого предложения текста, для этого используются указатели и функции для работы с динамической памятью.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

1. Название файла: main.c

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

char* get_line(){
  int memory = 100;
  int line_length = 0;
  int sym, fl = 1;
  char *line = malloc(memory *sizeof(char));
  while (1) {
    sym = getchar();
    if (fl == 1 && (sym == ' '|| sym == '\t' || sym == '\n'))
      continue;
    else
      fl = 0;
    line[line_length++] = sym;
    if (sym == '.' || sym == ';' || sym == '?' || sym == '!')
      break;
    if (line_length == memory) {
      memory += 100;
      line = realloc(line, memory);
    }
  }
  line[line_length] = '\0';
  return line;
}

int has_qm (char* line){
  int qm = 0;
  if (line[strlen(line)-1] == '?')
    qm = 1;
  return qm;
}

int main() {
  int text_lenght = 0;
  int qm_amount = 0;
  int memory = 100;
  char* stop_line = "Dragon flew away!";
  char** text = malloc(memory*sizeof(char*));
  char* line;
  while (1) {
    line = get_line();
    if (!has_qm(line))
      text[text_lenght++] = line;
    else qm_amount += 1;
    if (text_lenght == memory) {
      memory += 100;
      text = realloc(text, memory*sizeof(char*));
    }
    if (!strcmp(line, stop_line)) break;
  }
```

```
  for (int i = 0; i < text_lenght; i++){
      puts(text[i]);
      free(text[i]);
    }
  free(text);
  printf("Количество предложений до %d и количество предложений
после %d\n", text_lenght+qm_amount-1, text_lenght-1);
  return 0;
}
```