МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)
Кафедра САПР

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине «Программирование»

Тема: Массивы указателей. Динамическое управление памятью

Студенты гр. 9892 _____ Лескин К.А.

_____ Миллер В.В.

Преподаватель _____ Кузьмин С.А.

Санкт-Петербург
2020

Цель работы

Получение практических навыков разработки программы, обрабатывающей данные, представленные массивом указателей.

Формулировка задания

Разработать программу, обрабатывающую текстовую информацию, представленную массивом указателей на строки (массивы символов). Программа должна считать данные (текстовую информацию) из входного файла, представленного набором строк (каждая строка представляет собой последовательность символов, среди которых могут быть буквы, пробелы, знаки препинания и т.п.).

Далее над текстом выполняется операция, определённая в вариантах задания. После чего результаты выполнения операции должны быть отражены на экране и записаны в новый файл.

Доступ к каждой строке массива, а также к отдельному элементу (символу) строки должно осуществляться через указатели.

Индивидуальный вариант: Удалить все слова, содержащие заданный символ, в каждой строке текста.

Форматы входных и выходных файлов

Входной файл представляет собой обычный текстовый файл, в котором содержится набор исходных строк текста.

Выходной файл запысывается следующем формате:

- вначале должны быть записаны исходные строки;
- потом должно быть записано название совершённой операции;
- потом должны быть записаны введённые данные и параметры;
- после чего уже должны быть записаны полученные строки.

Описание структур данных

Для хранения строк, считанных из файла, используется массив указателей на строки (каждая строка имеет тип данных **char***). Доступ к массиву осуществляется через указатель на него. Память под строки и массив указателей на них выделяется динамически. Структура хранения данных указана на рис. 1.

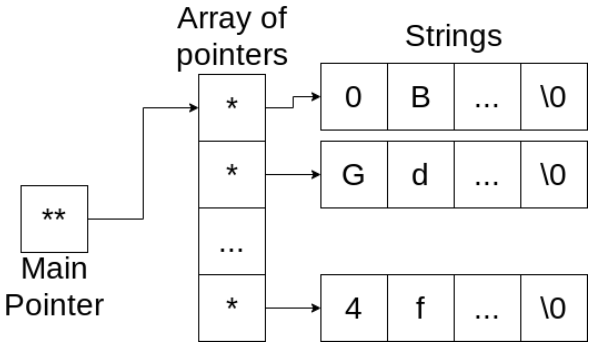

Рис. 1 – Структура хранения данных, считанных из файла

Описание пользовательских функций

DeleteWordsWithChar

void DeleteWordsWithChar(char *strings[], int lines_number, char bad_char);

Функция по удалению слов, содержащих заданный символ, из массива строк. Принимает указатель на массив строк, количество строк и символ, который должен содержаться в словах для удаления.

Алгоритм работы функции:

Цикл по line от 0 до lines_number

- Записываем указатель на текущую строку в string
- Выделяем память под новую строку и записываем указатель в new_string
- Выделяем память под слово и записываем указатель в word
- Если IfStringContainsChar для текущей строки возвращает **false**, сбрасываем текущую итерацию цикла
- Цикл по character_pos от 0 до длины string
  - Записываем текущий символ в character

- Если character == ' ' или character == '\n' или character == EOF или character == '\0'
  - Если IfStringContainsChar возвращает **true** для word
    strcat(new_string, word)
    strcat(new_string, character)
  - Освободить указатель word
  - Выделить память под слово и записать указатель в word
  - Сбросить текущую итерацию цикла
- Иначе
  strcat(word, character);

- Освобождаем указатель на проверенную строку
- Если послений символ строки не \n, добавляем \n в конец
- Присваиваем указателю адрес ячейки памяти, на которую указывает new_string

ReadFromFile

int ReadFromFile(const std::string& file_name, char **&strings);
Функция для чтения строк из файла.
Принимает имя файла и ссылку на указатель на указатель на char (необходимо, чтобы модифицировать оригинальный указатель).
Возвращает количество прочитанных строк.
Алгоритм работы функции:

- Пытемся открыть файл. В случаем провала возвращаем -1
- Задаём константы string_chunk_len = 10, lines_chunk_len = 3
- Выделяем память под массив строк длинной lines_chunk_len
- Выделяем память под нулевую строку длинной string_chunk_len
- Создаём место под считываемый символ
- Пока считанный из файла символ в current_char не равен EOF
  - ++line_size
  - Если длина строки line_size кратна string_chunk_len, расширяем текущую строку на string_chunk_len
  - Записываем считанный символ в текущую строку.

- Если считанный символ == \n
  - Записываем символ конца строки в текущую строку
  - line_size = 0
  - ++line
  - Если размер массива line кратен lines_chunk_len, расширяем массив строк на lines_chunk_len
  - Выделяем память под нулевую строку длинной string_chunk_len

- Закрываем файл
- Возвращаем количество считанных строк

WriteInFile

void WriteInFile(const std::string&, char **, char **, char, int);
Функция для записи массива строк в файл.

Принимает указатель на массив оригинальных строк, указатель на массив изменённых строк, символ, который содержался в удалённых словах и количество строк.

Алгоритм работы функции:

- Пытемся открыть файл. В случаем провала возвращаемся
- Записываем в файл оригинальные строки
- Записываем в файл совершённую операцию
- Записываем в файл параметры операции
- Записываем в файл модифицированные строки
- Закрываем файл

IfStringContainsChar

bool IfStringContainsChar(char* string, char bad_char);
Функция для проверки содержания в строке заданного символа.
Принимает указатель на строку и проверяемый символ.
Возвращает true, если символ найден, false в противном случае.
Алгоритм работы функции:

- Записываем указатель на строку в word_ptr
- Пока значение word_ptr не 0

  - Если strchr(&bad_char, *word_ptr) Возвращаем **true**
  - ++word_ptr
- Возвращаем **false**

PrintArrayOfStrings

void PrintArrayOfStrings(char **strings, int lines_number);
Функция для печати массива строк.
Принимает указатель на массив строк и количество строк.
Алгоритм работы функции:

- Циклом по line от 0 до lines_number печатаем строку

strcat

char *strcat (char *dest, char character);
Перегрузка функции strcat из strings.h.
Принимает указатель на исходную строку и символ для добавления в конец.
Возвращает указатель на новую строку с добавленным символом.
Алгоритм работы функции:

- Выделяем память под 2 символа (char_ptr)
- В первую ячейку записываем character
- Во вторую ячейку записываем \0
- strcat(dest, char_ptr)
- Освобождаем память char_ptr
- Возвращаем указатель на dest

Алгоритм программы

- Вызываем ReadFromFile
- Проверяем, что файл считан успешно
- Переписываем считанные данные в original_strings
- Считываем с клавиатуры символ
- Вызываем DeleteWordsWithChar
- Печатаем результат с помощью PrintArrayOfStrings
- Записываем Изменения в файл с помощью WriteInFile

Тестирование программы

Тесты с описанием действий. Обязательно картинки с результатами

Выводы

В результате выполнения данной лабораторной работы мы закрепили и применили на практике знания, полученные в ходе изучения темы "Массивы указателей. Динамическое управление памятью". Реализованная программа соответствует поставленным задачам и безошибочно выполняет свою работу. Функционал программы позволяет не только выполнять вычисления, но и реализует полноценное взаимодействие с пользователем, корректно обрабатывать его запросы и выдавать ему ожидаемый результат. Выполнение данной лабораторной работы позволило углубить наши знания в технологии программирования типовых задач обработки массивов указателей и динамического управления памятью.

Приложение А. Листинг программного кода

main.cpp

```
1  //
2  // Created by K on 15.09.2020.
3  //
4
5  #include "files_lib.h"
6  #include "delete_words_with_char.h"
7  #include "print_array_of_strings.h"
8
9  int main()
10 {
11     char *input_file_name = (char*)"../input.txt";
12     char **strings{};
13     int lines_number = ReadFromFile(input_file_name,
           strings);
14     if(lines_number == -1 || strings == nullptr)
15     {
16         std::cout << "An Error occurred while reading file\
               n";
17         return 1;
18     }
19
20
21     char **original_strings;
22     original_strings = (char**)malloc(sizeof(char*)*
           lines_number);
23     for (int line = 0; line < lines_number; ++line)
24     {
25         original_strings[line] = (char*)malloc(sizeof(char)
               *strlen(strings[line]));
26         strcpy(original_strings[line], strings[line]);
27     }
28
29     char bad_char;
30     std::cout << "\nInput char:";
31     std::cin >> bad_char;
32     std::cout << "Deleting words with char: " << bad_char
           << "\n";
33     DeleteWordsWithChar(strings, lines_number, bad_char);
```

```
34
35      std::cout << "RESULT:\n
            ==========================================================\
            n";
36      PrintArrayOfStrings(strings, lines_number);
37      std::cout << "\n
            ==========================================================\
            n";
38
39      std::cout << "Writing changes in file...\n";
40      char *output_file_name = (char*)"../output.txt";
41      WriteInFile(output_file_name, original_strings, strings
            , bad_char, lines_number);
42      std::cout << "Done\n";
43      return 0;
44  }
```

delete_words_with_char.cpp

```
1  //
2  // Created by kirill on 19.09.2020.
3  //
4
5  #include "delete_words_with_char.h"
6
7  void DeleteWordsWithChar(char **strings, int lines_number,
       char bad_char)
8  {
9      for(int line = 0; line < lines_number; ++line)
10     {
11         char *string = strings[line];
12         char *new_string = (char*)calloc(sizeof(char),
               strlen(string));
13         char *word = (char*)calloc(sizeof(char), strlen(
               string));
14
15         if(!IfStringContainsChar(string, bad_char))
               continue;
16
17         for(unsigned long character_pos = 0; character_pos
               < strlen(string); ++character_pos)
18         {
19             char character = string[character_pos];
20             if (character == ' ' || character == '\n' ||
                   character == EOF || character == '\0')
21             {
22                 if(!IfStringContainsChar(word, bad_char))
23                 {
24                     strcat(new_string, word);
25                     strcat(new_string, character);
26                 }
27                 free(word);
28                 word = (char*)calloc(sizeof(char), strlen(
                       string));
29                 continue;
30             }
31             else
32             {
33                 strcat(word, character);
34             }
35         }
```

```
36
37              free(strings[line]);
38              if(new_string[strlen(new_string) - 1] != '\n')
39              {
40                  strcat(new_string, '\n');
41              }
42              strings[line] = new_string;
43          }
44      }
```

delete_words_with_char.h

```
//
// Created by kirill on 19.09.2020.
//

#ifndef LAB1_DELETE_WORDS_WITH_CHAR_H
#define LAB1_DELETE_WORDS_WITH_CHAR_H

#include <cstring>
#include <malloc.h>

#include "if_string_contains_char.h"
#include "strcat_impl.h"

void DeleteWordsWithChar(char *strings[], int lines_number,
    char bad_char);

#endif //LAB1_DELETE_WORDS_WITH_CHAR_H
```

files_lib.cpp

```
1  //
2  // Created by kirill on 19.09.2020.
3  //
4
5
6  #include "files_lib.h"
7
8  int ReadFromFile(const std::string& file_name, char **&
       strings)
9  {
10     // open file
11     FILE *input_file;
12     if((input_file = fopen(file_name.c_str(), "r")) ==
           nullptr)
13     {
14         std::cout << "Could not open \"" << file_name << "
               \"\n";
15         return -1;
16     }
17
18     const int string_chunk_len = 10, lines_chunk_len = 3;
19     int line = 0;
20     int line_size = 0;
21     strings = (char**)malloc(sizeof(char*) *
           lines_chunk_len);
22     strings[0] = (char*)malloc(sizeof(char) *
           string_chunk_len);
23     char current_char;
24     while((current_char = static_cast<char>(fgetc(
           input_file))) != EOF)
25     {
26         ++line_size;
27
28         // extend memory for string if needed
29         if(line_size % string_chunk_len == 0)
30         {
31             strings[line] = (char*)realloc(strings[line],
                   sizeof(char) * (line_size + string_chunk_len
                   ));
32         }
33
34         // add current_char in string
```

```
35          strcat(strings[line], current_char);
36
37          // move to new line if needed
38          if (current_char == '\n')
39          {
40              strcat(strings[line], '\0');
41              line_size -= line_size;
42              ++line;
43              if(line % lines_chunk_len == 0)
44              {
45                  strings = (char**)realloc(strings, sizeof(
                        char*) * (line + lines_chunk_len));
46              }
47              strings[line] = (char*)calloc(string_chunk_len,
                    sizeof(char));
48          }
49      }
50      fclose(input_file);
51      return line_size ? line+1 : line;
52  }
53
54  void WriteInFile(const std::string& file_name, char **
        original_strings, char **strings, char bad_char, int
        lines_number)
55  {
56      FILE* output_file = nullptr;
57      // write in file
58      if((output_file = fopen(file_name.c_str(), "w")) ==
            nullptr)
59      {
60          std::cout << "Could not open \"" << file_name << "
                \"\n";
61          return;
62      }
63
64      for(int line = 0; line < lines_number; ++line)
65      {
66          fputs(original_strings[line], output_file);
67      }
68
69      fputs("\n\nOperation: Delete words with a specific
            character\n", output_file);
70      fputs("Parameters: ", output_file);
```

```
71      fputc(bad_char, output_file);
72      fputs("\n\n", output_file);
73
74      for(int line = 0; line < lines_number; ++line)
75      {
76          fputs(strings[line], output_file);
77      }
78
79      fclose(output_file);
80  }
```

files_lib.h

```
1  //
2  // Created by kirill on 19.09.2020.
3  //
4
5  #ifndef LAB1_FILES_LIB_H
6  #define LAB1_FILES_LIB_H
7
8  #include <iostream>
9  #include <cstring>
10 #include <string>
11
12 #include "strcat_impl.h"
13
14 int ReadFromFile(const std::string& file_name, char **&
       strings);
15 void WriteInFile(const std::string&, char **, char **, char
       , int);
16
17 #endif //LAB1_FILES_LIB_H
```

if_string_contains_char.cpp

```cpp
//
// Created by kirill on 19.09.2020.
//

#include "if_string_contains_char.h"

bool IfStringContainsChar(char* string, char bad_char)
{
    char *word_ptr = string;
    while(*word_ptr) // != nullptr
    {
        if(strchr(&bad_char, *word_ptr))
        {
            return true;
        }
        ++word_ptr;
    }
    return false;
}
```

if_string_contains_char.h

```
//
// Created by kirill on 19.09.2020.
//

#ifndef LAB1_IF_STRING_CONTAINS_CHAR_H
#define LAB1_IF_STRING_CONTAINS_CHAR_H

#include <cstring>

bool IfStringContainsChar(char* string, char bad_char);

#endif //LAB1_IF_STRING_CONTAINS_CHAR_H
```

print_array_of_strings.cpp

```
1  //
2  // Created by kirill on 19.09.2020.
3  //
4
5  #include "print_array_of_strings.h"
6
7  void PrintArrayOfStrings(char **strings, int lines_number)
8  {
9      for(int line = 0; line < lines_number; ++line)
10     {
11         printf("%s", strings[line]);
12     }
13 }
```

print_array_of_strings.h

```
1  //
2  // Created by kirill on 19.09.2020.
3  //
4
5  #ifndef LAB1_PRINT_ARRAY_OF_STRINGS_H
6  #define LAB1_PRINT_ARRAY_OF_STRINGS_H
7
8  #include <cstdio>
9
10 void PrintArrayOfStrings(char **strings, int lines_number);
11
12 #endif //LAB1_PRINT_ARRAY_OF_STRINGS_H
```

strcat_impl.cpp

```
1  //
2  // Created by kirill on 19.09.2020.
3  //
4
5  #include "strcat_impl.h"
6
7  char *strcat (char *dest, char character)
8  {
9      char* char_ptr = (char*)malloc(sizeof(char)*2);
10     *char_ptr = character;
11     *(char_ptr + 1) = '\0';
12     strcat(dest, char_ptr);
13     free(char_ptr);
14     return dest;
15 }
```

strcat_impl.h

```
1  //
2  // Created by kirill on 19.09.2020.
3  //
4
5  #ifndef LAB1_STRCAT_IMPL_H
6  #define LAB1_STRCAT_IMPL_H
7
8  #include <cstring>
9  #include <malloc.h>
10
11 char *strcat (char *dest, char character);
12
13 #endif //LAB1_STRCAT_IMPL_H
```