Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра вычислительных систем

**ОТЧЕТ**

по практической работе 4

по дисциплине «**Программирование**»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. ИВ-221  «16» апреля 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Слащинин М.К./ |
|  |  |  |
| Проверил:  Старший Преподаватель Кафедры ВС  «17» апреля 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Фульман В.О./ |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[**ЗАДАНИЕ** 3](#_Toc35593781)

[**ВЫПОЛНЕНИЕ РАБОТЫ** 4](#_Toc35593782)

[**ПРИЛОЖЕНИЕ**](#_Toc35593783) 13

# **ЗАДАНИЕ**

***Вариант №6***

*Преобразовать все Windows-пути формата Cygwin к оригинальным Windows-путям.*

**ВЫПОЛНЕНИЕ РАБОТЫ**

В данной работе мне необходимо было разработать прогрмму, которая принимает на вход Windows-пути формата Cygwin и преобразовывает их в обычные Windows-пути. При этом происходият проверки на корректность пути, допустимости символов, а также на запрещенные символы. Основным требованием было разбить программу на 4 функции: **input()** - функция ввода данных; **check()** - функция проверки корректности данных; **process()** - функция обработки строки; **output()** - функция вывода данных; Также необходимо было реализовать функции обработки строк из стандартной библиотеки string.h.

**функция вычисления длины строки:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **size\_t** **slen**(**char** \*mass) {  **size\_t** count = **0**;  **while** (\*mass != '\0') {  count++;  mass++;  }  **return** count;  } |

**функция разбиения строки на элементы-токены:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **int** **stok**(**char** \*src, **char** symb, **char** \*\*strs) {  **int** i = **0**;  **int** numstrs = **1**;  strs[i] = src;  **while** (src[i] != '\0') {  **if** (src[i] == symb) {  src[i] = '\0';  strs[numstrs] = &(src[i]) + **1**;  numstrs++;  }  i++;  }  **return** numstrs;  } |

**функция проверки символа на принадлежность заданному множеству символов:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **char** \***sspn**(**char** \*mass, **char** element) {  **for** (; \*mass != '\0'; mass++) {  **if** (\*mass == element) {  **return** mass;  }  }  **return** NULL;  } |

**функция сравнения строк:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **int** **scmp**(**char** \*mass, **char** \*mass2) {  **int** len = slen(mass);  **for** (**int** i = **0**; i < len; i++) {  **if** (mass[i] != mass2[i]) {  **return** mass[i] - mass2[i];  }  }  **return** **0**;  } |

**функция копирования строк:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **int** **scmp**(**char** \*mass, **char** \*mass2) {  **void** scpy(**char** \*mass, **char** \*mass2) {  **while** (\*mass != '\0') {  \*mass2++ = \*mass++;  }  \*mass = '\0';  } |

**функция восстановления строки после разбиения на элементы-токены:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | **void** **unstok**(**char** \*str, **char** delim, **char** \*\*ptr, **int** cnt) {  **for** (**int** i = **1**; i < cnt; i++) {  \*(ptr[i] - **1**) = delim;  }  } |

**функция преобразования буквы из нижнего решистра в верхний:**

|  |  |
| --- | --- |
| 1 | **char** **min\_to\_max**(**char** element) { **return** ((**int**)element - **32**); } |

*Помимо вышеперечисленных функций мне понадобились собственные функции и структуры:*

Структура path\_to — хранит в себе 2 поля. 1 поле — массив типа char размером MAX\_PATH = 260, 2 поле — переменная типа size\_t, которая хранит длину строки. Эта структура создана для хранения пути. Для работы с этой структурой создал несколько функций:

**функция инициализации структуры:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | path\_to \***path\_to\_init**() {  path\_to \*new = (path\_to \*)malloc(**sizeof**(path\_to));  **if** (new == NULL) {  **return** NULL;  }  new->len = **0**;  **return** new;  } |

**функция добавления элемента в строку:**

|  |  |
| --- | --- |
| 1  2  3  4 | **void** **path\_to\_add**(path\_to \*path, **char** value) {  path->string[path->len] = value;  path->len++;  } |

**функция освобождения памяти структуры:**

|  |  |
| --- | --- |
| 1  2  3  4 | **void** **path\_to\_free**(path\_to \*path) {  free(path);  path = NULL;  } |

Также мне пригодилась еще одна структура: errors\_of\_path; Структура имеет 2 поля: указатель на char, переменную типа enum;

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **typedef** **enum** PathError {  ErrorTypeNon,  ErrorTypeAintPath,  ErrorTypeTooLongLenght,  ErrorTypeBannedSymbol,  ErrorTypeVagueSymbol,  ErrorTypeIncorrectPath  } PathError; |

**функция инициализации структуры:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | errors\_of\_path \***errors\_init**() {  errors\_of\_path \*new = (errors\_of\_path \*)malloc(**sizeof**(errors\_of\_path));  **if** (new == NULL) {  **return** NULL;  }  **return** new;  } |

**функция освобождения памяти под структуру**:

|  |  |
| --- | --- |
| 1  2  3  4 | **void** **errors\_free**(errors\_of\_path \*error) {  free(error);  error = NULL;  } |

**Описание основных функций и запуск тестов:**

функция **input()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **void** **input**(path\_to \*path) {  printf("delim: +**\n**");  printf("path: ");  **char** b;  **while** ((b = getchar()) != '\n') {  path\_to\_add(path, b);  }  } |

На вход функции поступает указатель на структуру path\_to, а дальше циклом while через getchar() происходит добавление элементов в строку.

функция **check()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | **void** **check**(path\_to \*path, errors\_of\_path \*error) {  **char** \*ck = sspn(path->string, '/');  **if** (ck == NULL || (slen(path->string) < **3**)) {  error->type\_of\_error = ErrorTypeAintPath;  **return**;  }  **if** (path->len > MAX\_PATH) {  error->type\_of\_error = ErrorTypeTooLongLenght;  **return**;  }  **char** \*check = black\_symb(path);  **if** (check != NULL) {  error->type\_of\_error = ErrorTypeBannedSymbol;  error->column = check;  **return**;  }  **for** (**int** i = **0**; i < path->len; i++) {  **if** (!(path->string[i] >= **32** && path->string[i] <= **126**)) {  error->type\_of\_error = ErrorTypeVagueSymbol;  error->column = &path->string[i];  **return**;  }  }  **char** \*pointers\_of\_path[**10**];  **int** count\_of\_path = stok(path->string, '+', pointers\_of\_path);  **for** (**int** i = **0**; i < count\_of\_path; i++) {  **char** \*pointers\_of\_dir[**10**];  **int** count\_of\_dir = stok(pointers\_of\_path[i], '/', pointers\_of\_dir);  **int** t = **0**;  **if** (slen(pointers\_of\_dir[t]) == **0**) {  t++;  }  **if** (scmp(pointers\_of\_dir[t], "cygdrive") == **0**) {  **if** (!((slen(pointers\_of\_dir[t + **1**]) == **1**) &&  (\*pointers\_of\_dir[t + **1**] >= **97** &&  \*pointers\_of\_dir[t + **1**] <= **122**))) {  error->column = pointers\_of\_dir[t + **1**];  error->type\_of\_error = ErrorTypeIncorrectPath;  **return**;  }  }  unstok(pointers\_of\_path[i], '/', pointers\_of\_dir, count\_of\_dir);  }  unstok(path->string, '+', pointers\_of\_path, count\_of\_path);  error->type\_of\_error = ErrorTypeNon;  **return**; |

Функция принимает на вход указатель на структуру path\_to и указатель на структуру errors\_of\_path. В ней происходит проверка на ряд ошибок, и если какая-то ошибка все-таки выявилась, она передается структуре errors, если же ошибок не нашлось, в errors передается информация об их отсутствии.

**функция process()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | **void** **process**(path\_to \*path, errors\_of\_path \*error, path\_to \*new) {  **if** (error->type\_of\_error != **0**) {  **return**;  }  **char** \*pointers\_of\_path[**10**];  **int** count\_of\_path = stok(path->string, '+', pointers\_of\_path);  **for** (**int** i = **0**; i < count\_of\_path; i++) {  **int** t = **0**;  **char** \*pointers\_of\_dir[**10**];  **int** count\_of\_dir = stok(pointers\_of\_path[i], '/', pointers\_of\_dir);  **if** (slen(pointers\_of\_dir[**0**]) == **0**) {  t++;  }  **if** (scmp(pointers\_of\_dir[t], "cygdrive") == **0**) {  \*pointers\_of\_dir[t + **1**] = min\_to\_max(\*pointers\_of\_dir[t + **1**]);  unstok(pointers\_of\_path[i], **92**, pointers\_of\_dir, count\_of\_dir);  path\_to\_add(new, \*pointers\_of\_dir[t + **1**]);  path\_to\_add(new, ':');  **for** (**int** k = **10** + t; k < slen(pointers\_of\_path[i]); k++) {  path\_to\_add(new, pointers\_of\_path[i][k]);  }  **if** ((i + **1**) < count\_of\_path) {  path\_to\_add(new, '+');  }  } **else** {  unstok(pointers\_of\_path[i], '/', pointers\_of\_dir, count\_of\_dir);  **for** (**int** k = **0**; k < slen(pointers\_of\_path[i]); k++) {  path\_to\_add(new, pointers\_of\_path[i][k]);  }  **if** ((i + **1**) < count\_of\_path) {  path\_to\_add(new, '+');  }  }  }  } |

В функции происходит самая главная часть программы — преобразование пути. Функция принимает на вход 2 указателя на структуру path\_to(1 указатель — наш входной путь, второй указатель — итоговый путь), указатель на структуру errors\_of\_path.

**функция output()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | **void** **output**(path\_to \*path, errors\_of\_path \*error, path\_to \*new) {  **switch** (error->type\_of\_error) {  **case** ErrorTypeNon:  **if** (scmp(path->string, new->string) == **0**) {  printf("**\n**");  printf("Nothing to change**\n**");  }**else** {  printf("**\n**");  printf("new path: %s**\n**", new->string);  }  **break**;  **case** ErrorTypeAintPath:  printf("**\n**");  printf("ERROR**\n**");  printf("You didn't wrote a PATH!!!**\n**");  **break**;  **case** ErrorTypeTooLongLenght:  printf("**\n**");  printf("ERROR**\n**");  printf("Your PATH is too long!!!**\n**");  printf("Your size = %lu. Max size = 260 symbols**\n**", path->len);  **break**;  **case** ErrorTypeBannedSymbol:  printf("**\n**");  printf("ERROR**\n**");  printf("Your PATH is contain symbol: '%c' from black-list!!!**\n**",  \*error->column);  **break**;  **case** ErrorTypeVagueSymbol:  error->column[**2**] = '\0';  printf("**\n**");  printf("ERROR**\n**");  printf("Your PATH is contain NOT available symbol: '%s'**\n**", error->column);  **break**;  **case** ErrorTypeIncorrectPath:  printf("**\n**");  printf("ERROR**\n**");  printf("Incorrect PATH!!!**\n**");  printf("Inaccuracy in the element :'%s'**\n**", error->column);  **break**;  }  } |

Функция предназначена для обработки ошибок и, если их нет, вывода нового пути на экран. На вход функции поступает 2 указателя на старый и новый пути, а также указатель на структуру errors.

В конце необходимо прокрыть программу тестами.

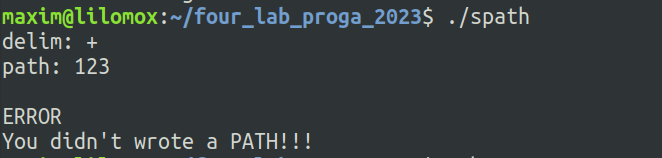
1. некорректный файловый путь.

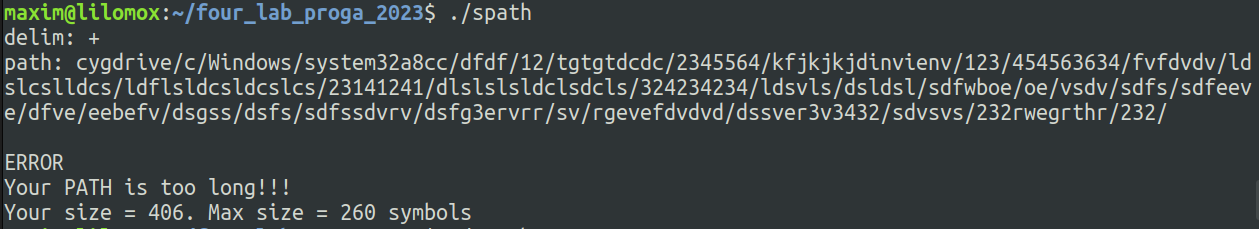
2. превышение допустимой длины пути.

3. допустимый путь, который не удовлетворяет указанным в задании условиям.

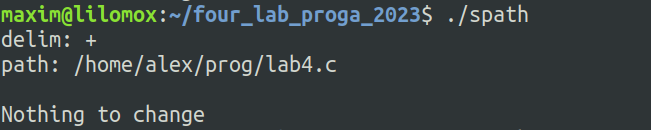
4. допустимый путь, удовлетворяющий условиям.

1.

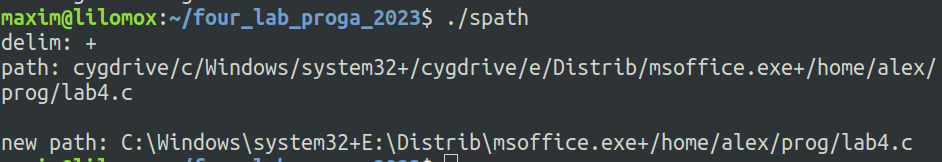


2.

3.



4.



# **П****РИЛОЖЕНИЕ**

Функция **main()** в файле **main.c**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | **int** **main**() {  path\_to \*path;  path\_to \*new;  errors\_of\_path \*error;  path = path\_to\_init();  new = path\_to\_init();  error = errors\_init();  **if** (path == NULL) {  printf("Can't allocate memory**\n**");  path\_to\_free(new);  errors\_free(error);  **return** -**1**;  }  **if** (new == NULL) {  printf("Can't allocate memory**\n**");  path\_to\_free(path);  errors\_free(error);  **return** -**1**;  }  **if** (error == NULL) {  printf("Can't allocate memory**\n**");  path\_to\_free(path);  path\_to\_free(new);  **return** -**1**;  }  input(path);  check(path, error);  process(path, error, new);  output(path, error, new);  path\_to\_free(path);  path\_to\_free(new);  errors\_free(error);  } |