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-222
«24» мая 2023 г. _____ Очнев А.Д.

Проверил:
Старший преподаватель кафедры
вычислительных систем
«24» мая 2023 г. _____ Фульман В.О.

Оценка «_____»

Новосибирск 2023

ОГЛАВЛЕНИЕ

ЗАДАНИЕ

Необходимо реализовать обработку строковой информации в соответствии с заданием. В качестве обрабатываемых данных рассматриваются пути к файлам в операционных системах Windows и GNU/Linux.

1. Исходная задача должна быть разбита на четыре основные подзадачи:
    1. ввод данных – функция `input()` – предусматривает взаимодействие с пользователем и возвращает строку, содержащую входные данные (путь);
    2. проверка корректности данных – `check()` – которая обеспечивает проверку допустимости длины входной строки и используемых в ней символов. Значения, возвращаемые функцией `check()`, должны позволять определить тип ошибки и (если возможно) номер символа, в которой она обнаружена;
    3. обработка – `process()` – входных данных согласно заданию;
    4. вывод данных – `output()` – на экран, обеспечивает отображение полученных результатов или сообщений об ошибке.

    Взаимодействие между указанными функциями осуществляется через данные. Каждая из них, при необходимости, также разбивается на подзадачи. Разбиение производится до тех пор, пока подзадачи не становятся тривиальными, например, вычисление длины строки.
2. Каждая подзадача оформляется в виде функции.
3. Не допускается использования стандартных функций обработки строк. Все операции над строками должны быть реализованы самостоятельно в виде отдельных подпрограмм. Эти подпрограммы необходимо разместить в отдельном файле `strings.c`, а прототипы функций – в файле `strings.h`.

Примерами типичных функций, которые должны присутствовать в каждой программе являются:

1. функция вычисления длины строки (`slen`);
2. функция разбиения строки на элементы-токены (`stok`), разделенные заданным символом (например, символ “/” при анализе пути или символ “.”, при разборе IP-адресов или доменных имен);
3. функция проверки символа на принадлежность заданному множеству символов (`sspn`), необходимая для проверки допустимости используемых символов.
4. функция сравнения строк (`scmp`);
5. функция копирования строк (`scpy`).

Набор тестов должен обеспечивать проверку поведения программы для всех классов входных данных:

1. некорректный файловый путь;
2. превышение допустимой длины пути;
3. допустимый путь, который не удовлетворяет указанным в задании условиям;
4. допустимый путь, удовлетворяющий условиям.

Вариант 4:

Преобразовать все пути, содержащие служебные каталоги: текущий каталог («.») и родительский каталог («..»); к минимально возможным путям.

ВЫПОЛНЕНИЕ РАБОТЫ

В файле strings.c расположены реализованные функции обработки строк.

```
size_t slen(char *str) {
  size_t len = 0;
  for (int i = 0; str[i] != '\0'; i++)
    len++;
  return len;
}
```

Функция slen считает длину строки. Принимает на вход указатель на строку, возвращает длину строки типа size_t.

```
char *stok(char *str, char *delim) {
  static char *p;
  if (!str) {
    str = p;
  }
  if (!str) {
    return NULL;
  }
  while (1) {
    if (is_del(*str, delim)) {
      str++;
      continue;
    }
    if (*str == '\0') {
      return NULL;
    }
    break;
  }
  char *tmp = str;
  while (1) {
    if (*str == '\0') {
      p = str;
      return tmp;
    }
    if (is_del(*str, delim)) {
      *str = '\0';
      p = str + 1;
      return tmp;
    }
    str++;
  }
}
```

Функция stok принимает на вход указатель на строку и указатель на символ. Возвращает подстроку в строке, разделённую символом. Возвращает NULL если конец файла и подстрок больше нет.

```c
char *scpy(char *str1, char *str2) {
  if (str1 == NULL) {
    return NULL;
  }
  int len = slen(str1);
  for (int i = 0; i < len; i++) {
    str2[i] = str1[i];
  }
  str2[len] = '\0';
  return str2;
}
```

Функция scpy принимает на вход 2 указателя на строки, копирует содержание второй строки в первую, возвращает строку, в которую были скопированы символы.

```c
int scmp(char *str1, char *str2) {
  while (*str1) {
    if (*str1 != *str2)
      break;
    str1++;
    str2++;
  }
  if ((*str1 - *str2) > 0)
    return 1;
  else if ((*str1 - *str2) < 0)
    return -1;
  return 0;
}
```

Функция scmp принимает на вход 2 строки, возвращает 0 если сравниваемые строки идентичны, положительное число если строки отличаются и ASCII код первого отличающегося символа в первой строке больше кода символа на той же позиции во второй строке, отрицательное число если строки отличаются и ASCII код первого отличающегося символа в первой строке меньше кода символа на той же позиции во второй строке.

```c
char *schr(char *str, char c) {
  while (*str != c && *str != '\0')
    str++;
  if (*str == c)
    return str;
  else
    return NULL;
}
```

Функция schr принимает на вход указатель на строку и символ, ищет первое вхождение этого символа в строку. Возвращает указатель на символ в строке, если его нет, возвращает NULL.

```
int is_del(char c, char *delim) {
  while (*delim != '\0') {
    if (c == *delim)
      return 1;
    delim++;
  }
  return 0;
}
```

Функция is_del принимает на вход символ и указатель на строку, возвращает 1, если символ входит в строку. Если символ не входит в строку, возвращает 0.

```
void delchar(char *str) {
  for (int i = 0; i < slen(str); ++i)
    str[i] = str[i + 1];
}
```

Функция delchar принимает на вход указатель на строку и удаляет из строки первый элемент.

```
char *scat(char *str1, char *str2) {
  char *ptr = str1 + slen(str1);
  while (*str2 != '\0') {
    *ptr++ = *str2++;
  }
  *ptr = '\0';
  return str1;
}
```

Функция scat принимает на вход 2 указателя на строки, дописывает в конец первой строки вторую строку и возвращает указатель на получившуюся строку.

```
char *sstr(char *str, char *ptr) {
  while (*str != '\0') {
    if ((*str == *ptr) && compare(str, ptr)) {
      return str;
    }
    str++;
  }

  return "0";
}
```

Функция sstr принимает на вход 2 указателя на строки и возвращает указатель на первое вхождение строки ptr в строку str. Если вхождение не обнаружено, то возвращает 0.

```c
int compare(char *str, char *ptr) {
  while (*str != '\0' && *ptr != '\0') {
    if (*str != *ptr) {
      return 0;
    }
    str++;
    ptr++;
  }
  if (*ptr == '\0') {
    return 1;
  }
  return 0;
}
```

Функция compare принимает на вход 2 указателя на строки и возвращает 1, если строка ptr равна первым n символам строки str. Иначе возвращает 0.

Функции, обеспечивающие работу приложения, хранятся в файле parser.c.

```c
char *input(char *str) {
  scanf("%s", str);
  return str;
}
```

Функция input принимает на вход указатель на пустую строку и возвращает указатель на строку, заполненную из консоли.

```c
int output(char *str) {
  if (printf("%s\n", str) != 0)
    return 0;
  else
    return -1;
}
```

Функция output принимает на вход указатель на строку и возвращает 0, если получилось вывести строку и -1, если произошла ошибка.

```c
int is_it_linux(char *str) {
  if (schr(str, '\\') != NULL || schr(str, ':') != NULL)
    return -1;
  return 0;
}
```

Функция is_it_linux принимает на вход указатель на строку и возвращает 0, если в строке нет символов ‘\’ и ‘:’, иначе возвращает -1.

```c
int is_it_windows(char *str) {
  if (schr(str, '/') != NULL)
    return -1;
  if ((((str[0] >= 65 && str[0] <= 90) || (str[0] >= 97 && str[0] <= 122)) &&
       ((str[1] >= 65 && str[1] <= 90) || (str[1] >= 97 && str[1] <= 122)) &&
       str[2] == ':' && str[3] == '\\') ||
      (((str[0] >= 65 && str[0] <= 90) || (str[0] >= 97 && str[0] <= 122)) &&
       str[1] == ':' && str[2] == '\\')) {
    return 0;
  }
  return -1;
}
```

Функция is_it_windows принимает на вход указатель на строку, и возвращает 0, если в пути есть имя раздела и в пути нет символа ‘/’. Иначе возвращает -1.

```c
int symbols_check(char *str) {
  if (schr(str, '*') != NULL || schr(str, '?') != NULL ||
      schr(str, '<') != NULL || schr(str, '>') != NULL ||
      schr(str, '|') != NULL)
    return -1;
  return 0;
}
```

Функция symbols_check принимает на вход указатель на строку, и возвращает 0, если в строке нет запрещённых для пути символов. Иначе возвращает -1.

```c
int error_check(char *str) {
  if (symbols_check(str) != 0) {
    printf("Некорректные символы в пути %s\n", str);
    return -1;
  } else if (slen(str) > MAX_PATH) {
    printf("Превышена максимальная длинна пути %s\n", str);
    return -1;
  } else if (is_it_linux(str) == 0) {
    return 0;
  } else if (is_it_windows(str) != 0) {
    printf("Некорректные символы в Windows пути %s\n", str);
  }
  return 0;
}
```

Функция error_check принимает на вход указатель на строку. Вызываются другие функции, которые проверяют наличие ошибок в строке, и в случае ошибки выводят её причину, после чего функция возвращает -1. Если ошибок в строке не было, возвращается 0.

```
char *delete_parental(char *str) {
  delchar(str);
  delchar(str);
  delchar(str);
  delchar(str);
  str--;
  while (*str != '\\' && *str != '/') {
    delchar(str);
    str--;
  }
  return str;
}
```

Функция delete_parental принимает на вход указатель на строку и возвращает указатель на строку с удалённым родительским каталогом из пути.

```
char *delete_current(char *str) {
  delchar(str);
  delchar(str);
  return str;
}
```

Функция delete_parental принимает на вход указатель на строку и возвращает указатель на строку с удалённым текущем каталогом из пути.

```
char *process(char *str) {
  char *c = sstr(str, "/../");
  if (*c == '0') {
    c = sstr(str, "\\..\\");
  }
  if (*c != '0') {
    delete_parental(c);
  }

  c = sstr(str, "/./");
  if (*c == '0') {
    c = sstr(str, "\\.\\");
  }
  if (*c != '0') {
    delete_current(c);
  }
  return str;
}
```

Функция delete_parental принимает на вход указатель на строку и возвращает указатель на преобразованную строку, с удалёнными родительским и текущем путём.

В файле main.c используем вышеописанные функции для реализации программы.

```
int main() {
  char *delim = malloc(1);
  char *in = malloc(MAX_PATH * 12);
  char *out = malloc(MAX_PATH * 12);
  char *tmp = malloc(MAX_PATH);
  printf("delim: ");
  input(delim);
  printf("paths: ");
  input(in);
  tmp = stok(in, delim);
  while (tmp != NULL){
    if (error_check(tmp) == 0){
      tmp = process(tmp);
      scat(out, "+");
      scat(out, tmp);
    } else {
      return -1;
    }
    tmp = stok(NULL, delim);
  }
  delchar(out);
  output(out);
  free(tmp);
  free(out);
  free(in);
  free(delim);
  return 0;
}
```

Выделяем память под необходимые строки, считываем разделяющий символ и путь. Отделяем первый путь. В цикле, пока существуют пути, проверяем путь на корректность. Если путь корректен, то вызываем функцию для его обработки. Присоединяем обработанные пути в одну строку, выводим её, после чего очищаем память.

ПРИЛОЖЕНИЕ

strings.h

```
1   #pragma once
2
3   #include <stdio.h>
4
5
6   #define MAX_PATH 260
7
8   size_t slen(char *str);
9   int scmp(char *str1, char *str2);
10  char *scpy(char *str1, char *str2);
11  char* stok(char *str, char *delim);
12  char *schr(char *str, char c);
13  int is_del(char c, char *delim);
14  void delchar(char *str);
15  char *scat(char *str1, char *str2);
16  char *sstr(char *str, char *ptr);
17  int compare(char *str, char *ptr);
```

strings.c

```c
#include <stdio.h>

#include "strings.h"

size_t slen(char *str) {
  size_t len = 0;
  for (int i = 0; str[i] != '\0'; i++)
    len++;
  return len;
}

char *stok(char *str, char *delim) {
  static char *p;
  if (!str) {
    str = p;
  }
  if (!str) {
    return NULL;
  }
  while (1) {
    if (is_del(*str, delim)) {
      str++;
      continue;
    }
    if (*str == '\0') {
      return NULL;
    }
    break;
  }
  char *tmp = str;
  while (1) {
    if (*str == '\0') {
      p = str;
      return tmp;
    }
    if (is_del(*str, delim)) {
      *str = '\0';
      p = str + 1;
      return tmp;
    }
    str++;
  }
}

char *scpy(char *str1, char *str2) {
  if (str1 == NULL) {
    return NULL;
  }
  int len = slen(str1);
  for (int i = 0; i < len; i++) {
    str2[i] = str1[i];
  }
  str2[len] = '\0';
```

```
56      return str2;
57    }
58
59    int scmp(char *str1, char *str2) {
60      while (*str1) {
61        if (*str1 != *str2)
62          break;
63        str1++;
64        str2++;
65      }
66      if ((*str1 - *str2) > 0)
67        return 1;
68      else if ((*str1 - *str2) < 0)
69        return -1;
70      return 0;
71    }
72
73
74    char *schr(char *str, char c) {
75      while (*str != c && *str != '\0')
76        str++;
77      if (*str == c)
78        return str;
79      else
80        return NULL;
81    }
82
83    int is_del(char c, char *delim) {
84      while (*delim != '\0') {
85        if (c == *delim)
86          return 1;
87        delim++;
88      }
89      return 0;
90    }
91
92    void delchar(char *str) {
93      for (int i = 0; i < slen(str); ++i)
94        str[i] = str[i + 1];
95    }
96
97
98    char *scat(char *str1, char *str2) {
99      char *ptr = str1 + slen(str1);
100     while (*str2 != '\0') {
101       *ptr++ = *str2++;
102     }
103     *ptr = '\0';
104     return str1;
105   }
106
107   char *sstr(char *str, char *ptr) {
108     while (*str != '\0') {
109       if ((*str == *ptr) && compare(str, ptr)) {
110         return str;
111       }
112       str++;
```

```
113       }
114
115       return "0";
116     }
117
118
119     int compare(char *str, char *ptr) {
120       while (*str != '\0' && *ptr != '\0') {
121         if (*str != *ptr) {
122           return 0;
123         }
124         str++;
125         ptr++;
126       }
127       if (*ptr == '\0') {
128         return 1;
129       }
130       return 0;
      }
```

parser.h

```
1   #pragma once
2
3   #define MAX_PATH 260
4
5   char *input(char *str);
6   int output(char *str);
7   int is_it_linux(char *str);
8   int is_it_windows(char *str);
9   int error_check(char *str);
10  int symbols_check(char *str);
11  char *process(char *str);
12  char *delete_parental(char *str);
13  char *delete_current(char *str);
```

parser.c

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

#include "parser.h"
#include "strings.h"

char *input(char *str) {
  scanf("%s", str);
  return str;
}

int output(char *str) {
  if (printf("%s\n", str) != 0)
    return 0;
  else
    return -1;
}

int is_it_linux(char *str) {
  if (schr(str, '\\') != NULL || schr(str, ':') != NULL)
    return -1;
  return 0;
}

int is_it_windows(char *str) {
  if (schr(str, '/') != NULL)
    return -1;
  if ((((str[0] >= 65 && str[0] <= 90) || (str[0] >= 97 && str[0] <=
122)) &&
       ((str[1] >= 65 && str[1] <= 90) || (str[1] >= 97 && str[1] <=
122)) &&
       str[2] == ':' && str[3] == '\\') ||
      (((str[0] >= 65 && str[0] <= 90) || (str[0] >= 97 && str[0] <=
122)) &&
       str[1] == ':' && str[2] == '\\')) {
    return 0;
  }
  return -1;
}

int symbols_check(char *str) {
  if (schr(str, '*') != NULL || schr(str, '?') != NULL ||
      schr(str, '<') != NULL || schr(str, '>') != NULL ||
      schr(str, '|') != NULL)
    return -1;
  return 0;
}

int error_check(char *str) {
  if (symbols_check(str) != 0) {
    printf("Некорректные символы в пути %s\n", str);
    return -1;
```

```
56      } else if (slen(str) > MAX_PATH) {
57        printf("Превышена максимальная длинна пути %s\n", str);
58        return -1;
59      } else if (is_it_linux(str) == 0) {
60        return 0;
61      } else if (is_it_windows(str) != 0) {
62        printf("Некорректные символы в Windows пути %s\n", str);
63      }
64      return 0;
65    }
66
67    char *process(char *str) {
68      char *c = sstr(str, "/../");
69      if (*c == '0') {
70        c = sstr(str, "\\..\\");
71      }
72      if (*c != '0') {
73        delete_parental(c);
74      }
75
76
77      c = sstr(str, "/./");
78      if (*c == '0') {
79        c = sstr(str, "\\.\\");
80      }
81      if (*c != '0') {
82        delete_current(c);
83      }
84      return str;
85    }
86
87    char *delete_parental(char *str) {
88      delchar(str);
89      delchar(str);
90      delchar(str);
91      delchar(str);
92      str--;
93      while (*str != '\\' && *str != '/') {
94        delchar(str);
95        str--;
96      }
97      return str;
98    }
99
100   char *delete_current(char *str) {
        delchar(str);
        delchar(str);
        return str;
      }
```

main.c

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

#include "parser.h"
#include "strings.h"

int main() {
  char *delim = malloc(1);
  char *in = malloc(MAX_PATH * 12);
  char *out = malloc(MAX_PATH * 12);
  char *tmp = malloc(MAX_PATH);
  printf("delim: ");
  input(delim);
  printf("paths: ");
  input(in);
  tmp = stok(in, delim);
  while (tmp != NULL){
    if (error_check(tmp) == 0){
      tmp = process(tmp);
      scat(out, "+");
      scat(out, tmp);
    } else {
      return -1;
    }
    tmp = stok(NULL, delim);
  }
  delchar(out);
  output(out);
  free(tmp);
  free(out);
  free(in);
  free(delim);
  return 0;
}
```