Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра вычислительных систем

ОТЧЕТ
по практической работе 5

по дисциплине «**Программирование**»

Выполнил:
студент гр. ИС-142
«26» марта 2022 г. _____ /Наумов А.А/

Проверил:
д.т.н., ассистент Кафедры ВС
«26»марта 2022 г. _____ /Курзин А.С./

Оценка «_____»

Новосибирск 2022

ОГЛАВЛЕНИЕ

ЗАДАНИЕ

Общая информация

Номер варианта выбирается в соответствии с номером студента в журнале по формуле $(i-1)\%n+1$, где i – номер студента по журналу, n – количество заданий.

При разработке подпрограмм в соответствии с заданием следует учитывать, что:

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.

Для заданного имени текущего пользователя обновить список входных файлов, преобразовав пути, заданные относительно его домашнего каталога (вида `~/somedir`), к абсолютным путям. Имя каталога (`dir`), в котором находятся домашние каталоги пользователей, вводится с клавиатуры.

Вход:

```
delim: +

user name: jack

dir: /home/stud

paths: ~/games/packman.cpp+~alex/docs+~/study/Prog/lab4.c+/usr/bin/gcc
```

Выход:

```
new paths: /home/stud/games/packman.cpp+~alex/docs+/home/stud/study/Prog/lab4.c +/usr/bin/gcc
```

ВЫПОЛНЕНИЕ РАБОТЫ

Программа разделена на несколько главных функций: **input** (считывающая вводные данные), **check** (проверяющая введённые данные на корректность), **process** (обрабатывающая весь ввод, получающая нужный результат в виде новых путей), **output** (выводящая результат) и **printError** (для вывода сообщений об ошибках).

В функции **input** происходит считывание ввода с помощью функции **scanf** с заданным форматом: **“ %[^\n]”**, который позволяет считать строку до нажатия **Enter** (**‘\n’**) (постановки символа новой строки) и сразу же очистить буфер **stdin** с помощью пробела перед **%**.

Функция check проверяет случаи некорректного ввода директорий, имени пользователя и путей, возвращая код ошибки, который считывается в **printError** для вывода сообщения.

Ввод **username = “jack”**, **dir = “/home/stud”**. Далее идет функция process: программа выделяет память функцией **malloc** для нового (результирующего) пути. По умолчанию мы делаем его такого же размера, как изначальный путь, если он будет идти, например, от директории **“/”**. Дополнительно выделяем память под строку **user**, в которой будем хранить шаблон именного каталога пользователя (вида **“~username”**). С помощью функции **stok** делим строку (входной путь) на токены, каждый из которых (через цикл **while**) проверяем, начинается ли он с **“~/”** или “**~username**”. Если путь начинается с **“~/”**, перевыделяем память функцией **realloc** с увеличением размера строки на **userdir**, которая имеет вид **/home/stud/jack**. Записываем в результат **userdir** с помощью функции **scat** и меняем вид изначального пути с **~/somepath**… на **/somepath**…, чтобы в дальнейшем добавить этот путь в результат.

В случае, если путь начинается с **~username** (**~jack** в нашем случае), перевыделяем память под результат, добавляя **slen(dir)** ячеек, где **dir – “/home/stud”**, т.к. под имя пользователя у нас уже есть выделенная память. Также добавляем в результат **dir**, меняем путь с **~jack/…** на **/jack/…** .

После всех операций заносим оставшийся (возможно модифицированный) путь в результат, возвращаем его через **return**. В конце выводим получившийся результат функцией **output**, где происходит вывод посредством функции **printf**.

Как должна работать программа:

```
alexey@DESKTOP-1NTB03J:~/lab5$ ./main
delim: +
username: jack
dir: /home/stud
old paths: ~/games/packman.cpp+~alex/docs+~/study/Prog/lab4.c+/usr/bin/gcc
new paths: /home/stud/jack/games/packman.cpp+~alex/docs+/home/stud/jack/study/Prog/lab4.c+/usr/bin/gcc
```

```
alexey@DESKTOP-1NTB03J:~/lab5$ ./main
delim: +
username: kac
dir: d
old paths: /sler
ERROR: bad dir character
```

```
alexey@DESKTOP-1NTB03J:~/lab5$ ./main
delim: +
username: jack
dir: /dir
old paths: df
ERROR: bad path beginning
```

```
alexey@DESKTOP-1NTB03J:~/lab5$ ./main
delim: +
username: jack
dir: /ri
old paths: /sdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdg
sdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksd
gsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoks
dgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphoksdgsdfophkfgphok
ERROR: too long argument
```

ПРИЛОЖЕНИЕ;

Makefile

```
CC+FLAGS = gcc -Wall -MMD

all: main
-include *.d

main: main.o strings.o program.o
	$(CC+FLAGS) -o $@ $^

strings.o: strings.c
	$(CC+FLAGS) -c -o $@ $<

program.o: program.c
	$(CC+FLAGS) -c -o $@ $<

clean:
	rm -rf *.o *.d main
```

main.c

```
1  #include <ctype.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include "strings.h"
 5 #include "program.h"
 6
 7 int main()  {
 8     printf("delim: +\n");
 9     char *paths = malloc(sizeof(char) * 1024);
10     char *dir = malloc(sizeof(char) * 256);
11     char *username = malloc(sizeof(char) * 32);
12     input(paths, dir, username);
13     int inputErrors = check(dir, paths, username);
14     if(inputErrors != 0) {
15         printError(inputErrors);
16         return 0;
17     }
18     char *result = process(paths, dir, username);
19     output(result);
20     return 0;
21 }
```

program.c

```
1   #include <ctype.h>
  2 #include <stdio.h>
  3 #include <stdlib.h>
  4 #include "strings.h"
  5 #include "program.h"
  6
  7 // int checkPaths(char *paths) {
  8 //     char *path = stok(paths, "+");
```

```
 9 //     static int code = 0;
10 //     while(path != NULL) {
11 //         if(code != 0) {
12 //             continue;
13 //         }
14 //         if(path[0] != '/' && path[0] != '~') {
15 //             code = -5;
16 //         } else if(path[slen(path)-1] == '/') {
17 //             code = -6;
18 //         } else if(sstr(path, "\\") != NULL) {
19 //             code = -7;
20 //         } else if(slen(path) > 255) {
21 //             code = -8;
22 //         }
23 //         path = stok(NULL, "+");
24 //     }
25 //     return code;
26 // }
27
28
29 void input(char *paths, char *dir, char *username)
30 {
31     printf("username: ");
32         scanf(" %[^\n]", username);
33     printf("dir: ");
34         scanf(" %[^\n]", dir);
35         printf("old paths: ");
36         scanf(" %[^\n]", paths);
37 }
38
39 int check(char *dir, char *paths, char *username) {
40     if(dir == NULL || paths == NULL || username == NULL) {
41         return -1;
42     } else if(dir[0] != '/' || !isalpha(username[0])) {
43         return -2;
44     } else if(dir[slen(dir)-1] == '/' || username[slen(username)-1] ==
45 '/') {
46         return -3;
47     } else if(slen(dir) < 2 || slen(username) < 4 || slen(paths) < 2) {
48         return -4;
49     } else if(slen(dir) > 255 || slen(username) > 31) {
50         return -8;
51     } else {
52         return 0;
53     }
54 }
55
56 void printError(int inputErrors) {
57     printf("ERROR: ");
58     switch(inputErrors) {
59         case -1:
60             printf("some of arguments doesn't exist\n");
61             break;
62         case -2:
63             printf("bad dir character\n");
64             break;
65         case -3:
66             printf("bad last character\n");
```

```
 67             break;
 68         case -4:
 69             printf("too short argument\n");
 70             break;
 71         case -5:
 72             printf("bad path beginning\n");
 73             break;
 74         case -6:
 75             printf("path ending error\n");
 76             break;
 77         case -7:
 78             printf("bad character in path\n");
 79             break;
 80         case -8:
 81             printf("too long argument\n");
 82             break;
 83         }
 84 }
 85
 86 char* process(char* paths, char* dir, char* username) {
 87     int size = slen(paths) + 1;
 88     char *result = malloc(size * sizeof(char));
 89
 90     char *user = malloc(sizeof(char) * (slen(username) + 1));
 91     scat(user, "~");
 92     scat(user, username);
 93
 94     char *delim = "+";
 95     char *path = stok(paths, delim);
 96     while(path != NULL) {
 97         if(path[0] == '~' && path[1] == '/') {
 98             char *userdir = malloc(sizeof(char) * (slen(dir) +
 99 slen(username) + 1));
100             scat(userdir, dir);
101             scat(userdir, "/");
102             scat(userdir, username);
103             size += slen(userdir);
104             result = realloc(result, size * sizeof(char));
105             scat(result, userdir);
106             scpy(&path[1], path);
107             free(userdir);
108         } else if(sstr(path, user) != NULL) {
109             size += slen(dir);
110             result = realloc(result, size * sizeof(char));
111             scat(result, dir);
112             scat(result, "/");
113             scpy(&path[1], path);
114         }
115         scat(result, path);
116         path = stok(NULL, delim);
117         if(path != NULL) {
118             scat(result, delim);
119         }
120     }
121     return result;
122 }
123
124 void output(char *newPaths) {
```

```
        printf("new paths: %s\n", newPaths);
    }
```

strings.c

```
#include "strings.h"
#include <stdio.h>
#include <stdlib.h>

char *scpy(const char *src, char *des) //копирует строку
{
        int i;
        for (i = 0; src[i] != 0; i++) {
                des[i] = src[i];
        }
        des[i] = 0;
        return des;
}

int slen(const char *str) //длину возвращает строки
{
        int count = 0;;
        for (int i = 0; str[i] != '\0'; i++) {
                count++;
        }
        return ++count;
}

int scmp(const char *s1, const char *s2) //сравнивает строки
{
        char c1, c2;
        while(1) {
                c1 = *s1++;
                c2 = *s2++;
                if(c1 > c2) {
                        return 1;
                } else if(!c1) {
                        break;
                } else {
                        return -1;
                }
        }
        return 0;
}

char *scat(char *dest, const char *src) //вставляют одну строку в конец
другой
{
        char *end = dest + slen(dest) - 1;
        while(*src != '\0') {
                *end++ = *src++;
        }
        *end = '\0';
        return dest;
}
```

```
char *schr(const char *s, const char c) //ищет первое вхождение символа в
строку
{
        while(*s && *s != c) {
                ++s;
        }
        return (*s) ? (char*)s : NULL;
}

char *stok(char *str, const char *delim) // разделение на токены
{
        static char *next;
        if(str) {
                next = str;
                while(*next && schr(delim, *next)) {
                        *next++ = '\0';
                }
        }
        if(!*next) {
                return NULL;
        }
        str = next;
        while(*next && !schr(delim, *next)) {
                ++next;
        }
        while(*next && schr(delim, *next)) {
                *next++ = '\0';
        }
        return str;
}

const char* sstr(const char* s1, const char* s2) // находит первое вхождение
одной строки в другую
{
        if(*s2 == '\0') {
                return s1;
        }
        char *ptr;
        for(int i = 0; i < slen(s1); i++) {
                if(*(s1+i) == *s2) {
                        ptr = sstr(s1+i+1, s2+1);
                        return (ptr) ? ptr-1 : NULL;
                }
        }
        return NULL;
}
```

program.c

```
#pragma once

void input(char *paths, char *dir, char *username);
int check(char *dir, char *paths, char *username);
void printError(int inputErrors);
char* process(char* paths, char* dir, char* username);
void output(char *newPaths);
int checkPaths(char *paths);
```

strings.h

```
#pragma once

char *scpy(const char *src, char *des);
int slen(const char *str);
char *stok(char *str, const char *delim);
int scmp(const char *s1, const char *s2);
char *scat(char *dest, const char *src);
char *schr(const char *s, const char c);
const char* sstr(const char* s1, const char* s2);
```