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. ИВ-221
«15» мая 2023 г. _____ Беляев.Е.И.

Проверил:
Старший преподаватель Кафедры ВС
«15» мая 2023 г. _____ Фульман В.О.

Оценка «_____»

Новосибирск 2023

ОГЛАВЛЕНИЕ

ЗАДАНИЕ.....3
ВЫПОЛНЕНИЕ РАБОТЫ.....4
ПРИЛОЖЕНИЕ.....10

ЗАДАНИЕ

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

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.

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.

ВЫПОЛНЕНИЕ РАБОТЫ

Разработка функций обработки строк на основе стандартных функций. Strings.c

Is_letter()

Определяет в соответствии с таблицей ASCII является ли символ латинской буквой. Возвращает 0, если ответ утвердительный, -1 — ответ отрицательный.

```
1  int is_letter(char c)
2  {
3      if((c > 172 || c < 141) && (c < 101 || c > 132))
4          return -1;
5      return 0;
6  }
```

Slen()

Функция вычисления длины строки.

```
1  int slen(char *str)
2  {
3      int i = 0;
4      while(str[i] != '\0')
5          i++;
6      return i;
7  }
```

Sspn()

Функция выполняет поиск символов из строки **str2** в строке **str1**. Возвращает индекс первого найденного символа в **str1**, либо размер **str1** если символы не были найдены.

```
 1  int sspn(char *str1, char *str2)
 2  {
 3      if(str1 == NULL || str2 == NULL)
 4          return -1;
 5      int size1 = slen(str1), size2 = slen(str2);
 6      for(int i = 0; i < size1; i++)
 7      {
 8          for(int j = 0; j < size2; j++)
 9          {
10              if(str1[i] == str2[j])
11                  return i;
12          }
13      }
14      return size1;
15  }
```

Scpy()

Функция копирования строк. Переписывает в строку **dest** данные из строки **src**, пока не встретит символ конца строки. Возвращает указатель на копию **src**.

```
char *scpy(char *dest, char *src)
{
    if(src == NULL || dest == NULL)
        return NULL;

    int size = slen(src);
    for(int i = 0; i < size; i++)
        dest[i] = src[i];
    dest[size] = '\0';
    return dest;
}
```

Stok()

Функция разбиения строки на элементы-токены, разделенные заданным символом **delim**. Возвращает указатель на первый символ выделенной части строки, либо **NULL**, если строку **str** невозможно разделить на части.

```
char *stok(char *str, char delim)
{
    if(str == NULL)
        return NULL;

    int size = slen(str);
    for(int i = 0; i < size + 1; i++)
    {
        if(str[i] == delim)
        {
            str[i] = '\0';
            return str;
        }
        else if(str[i] == '\0')
            return str;
    }

    return NULL;
}
```

Concat()

Функция конкатенации строк. Переписывает в строку **dest** данные из **append** до символа конца строки. При этом символ \0 строки **dest** затерается первым

символом строки **append**. Возвращает указатель на результат объединения двух строк.

```
char *concat (char *dest, char *append)
{
    if(dest == NULL || append == NULL)
        return NULL;

    int size_dest = slen(dest);
    int size_append = slen(append);
    int j = 0;
    for(int i = size_dest; i < size_dest + size_append; i++)
    {
        dest[i] = append[j];
        j++;
    }
    return dest;
}
```

Lab5.c

Input()

Функция выделяет место по указателю str и с помощью getchar() получает из потока stdin символы для записи в str, пока не встретится \n символ новой строки. Используется realloc(), так как изначально не ясно, сколько символов нужно будет сохранить в строке str. В случае ошибки при выделении памяти возвращается NULL.

```
char *input()
{
    char *str, c;
    int i = 0, k = 2, size = 64;
    str = calloc(sizeof(char), size);
    if(str == NULL)
        return NULL;
    while((c = getchar()) != '\n')
    {
        if(i == size)
        {
            str = realloc(str, sizeof(char) * size * k);
            size *= 2;
            if(str == NULL)
                return NULL;
        }
        str[i] = c;
        i++;
    }
    str[i] = '\0';
    return str;
}
```

Check()

Проверяет каждый символ из str с помощью sspn(), пока не встретит символ конца строки или не превысит лимит длины пути MAX_PATH (260 символов). Используется в main() после input() для проверки введенных данных и передачи данных в output_error() в случае ошибки.

```
1  int check(char *str, int *num, char delim)
2  {
3      int i, path_i = 0;
4      char *sim = "\"*:<>?\\|";
5      if(str == NULL)
6          return -4;
7      if((i = sspn(str, sim)) != slen(str))
8      {
9          *num = i;
10         return -3;
11     }
12     i = 0;
13     while(str[i] != '\0')
14     {
15         if(str[i] == delim)
16             path_i = 0;
17         if(path_i == MAX_PATH)
18             return -1;
           if(str[i] == ':' && (is_letter(str[i - 
19 1]) == -1 || str[i + 1] != '/'))
20             return -2;
21         path_i++;
22         *num = ++i;
23     }
24     return 0;
25 }
```

Procces()

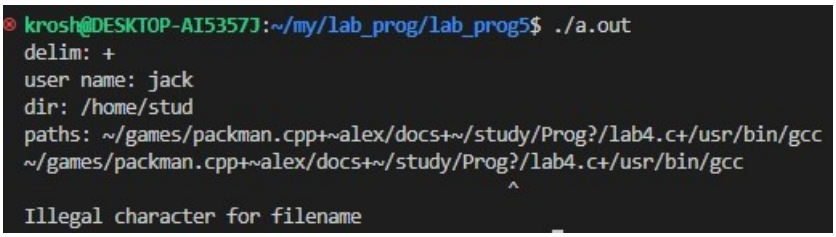
Выполняет действия со строками в соответствии с заданием, используя для этого функции из strings.c. Возвращает char **new_paths, в котором лежат новые обработанные пути.

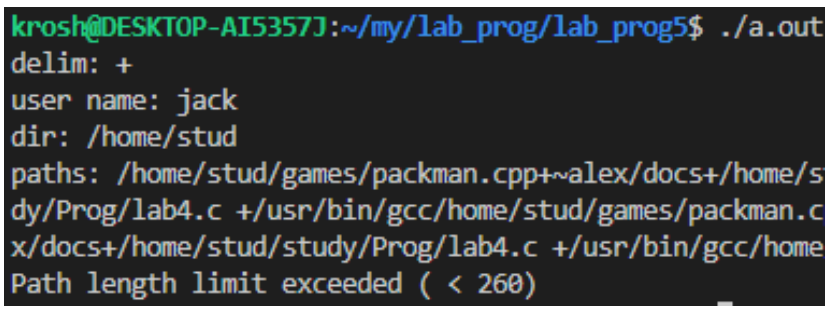
```
1  char **process(char *dir, char *str, char delim, int *i)
2  {
3      char **new_paths, *copy_dir = NULL, *tok = NULL;
4      int size_str, change, index = 0;
5
6      change = scout(str, delim) + 1;
7
8      new_paths = malloc(sizeof(char *) * change);
9
10     for(int i = 0; i < change; i++)
11     {
12         tok = stok(&str[index], delim);
13         size_str = slen(tok);
14         new_paths[i] = malloc(sizeof(char) * (size_str + 1));
```

```
15          new_paths[i] = scpy(new_paths[i], tok);
16          index += size_str + 1;
17      }
18
19      for(int i = 0; i < change; i++)
20      {
21          for(int j = 0; new_paths[i][j] != '\0'; j++)
22          {
23              if(new_paths[i][j] == '~' && new_paths[i][j + 1] == '/')
24              {
25                  copy_dir = malloc(sizeof(char) * (slen(dir) +
-   slen(&new_paths[i][j + 1])));
26                  copy_dir = scpy(copy_dir, dir);
27                  copy_dir = concat(copy_dir, &new_paths[i][j + 1]);
28                  free(new_paths[i]);
29                  new_paths[i] = copy_dir;
30                  copy_dir = NULL;
31              }
32          }
33      }
34      *i = change - 1;
35      return new_paths;
36  }
```

Output_error()

Выводит на экран сообщение об ошибке в соответствии с mode.






```
1   void output_error(char *str, int mode, int column)
2   {
3       if(mode == -1)
4           printf("Path length limit exceeded ( < 260)\n");
5       if(mode == -2)
6       {
7           show_error(str, column);
8           printf("Use rule violated for ':' in %d column\n", column);
9       }
```

```
    if(mode == -3)
    {
        show_error(str, column);
        printf("Illegal character for filename\n");
    }
    if(mode == -4)
        printf("Error in memory allocation function\n");
}
```

```
void show_error(char *str, int column)
{
    printf("%s\n", str);
    for(int j = 0; j < column; j++)
        printf(" ");
    printf("^\n");
}
```

Output()

Выводит на экран новые пути после обработки функцией procces().

```
void output(char **str, int size)
{
    int i;
    for(i = 0; i < size; i++)
    {
        printf("%s+", str[i]);
    }
    printf("%s", str[i]);
}
```

Main() представлена в приложении.

ПРИЛОЖЕНИЕ

Lab5.c

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

#include "strings.h"
#define MAX_PATH 260

void free_paths(char **paths, int size)
{
    int i;
    for(i = 0; i < size + 1; i++)
    {
        free(paths[i]);
    }
    free(paths);
}

char *input()
{
    char *str, c;
    int i = 0, k = 2, size = 64;
    str = calloc(sizeof(char), size);
    if(str == NULL)
        return NULL;
    while((c = getchar()) != '\n')
    {
        if(i == size)
        {
            str = realloc(str, sizeof(char) * size * k);
            size *= 2;
            if(str == NULL)
                return NULL;
        }
        str[i] = c;
        i++;
    }
    str[i] = '\0';
    return str;
}

int check(char *str, int *num, char delim)
{
    int i, path_i = 0;
    char *sim = "\"*:<>?\\|";
    if(str == NULL)
        return -4;
    if((i = sspn(str, sim)) != slen(str))
    {
        *num = i;
        return -3;
    }
    i = 0;
    while(str[i] != '\0')
    {
        if(str[i] == delim)
            path_i = 0;
        if(path_i == MAX_PATH)
            return -1;
```

```
57          if(str[i] == ':' && (is_letter(str[i - 1]) == -1 || str[i + 1] !
58  = '/'))
59              return -2;
60          path_i++;
61          *num = ++i;
62      }
63      return 0;
64  }
65
66  char **process(char *dir, char *str, char delim, int *i)
67  {
68      char **new_paths, *copy_dir = NULL, *tok = NULL;
69      int size_str, change, index = 0;
70
71      change = scout(str, delim) + 1;
72
73      new_paths = malloc(sizeof(char *) * change);
74
75      for(int i = 0; i < change; i++)
76      {
77          tok = stok(&str[index], delim);
78          size_str = slen(tok);
79          new_paths[i] = malloc(sizeof(char) * (size_str + 1));
80          new_paths[i] = scpy(new_paths[i], tok);
81          index += size_str + 1;
82      }
83
84      for(int i = 0; i < change; i++)
85      {
86          for(int j = 0; new_paths[i][j] != '\0'; j++)
87          {
88              if(new_paths[i][j] == '~' && new_paths[i][j + 1] == '/')
89              {
90                  copy_dir = malloc(sizeof(char) * (slen(dir) +
-   slen(&new_paths[i][j + 1])));
91                  copy_dir = scpy(copy_dir, dir);
92                  copy_dir = concat(copy_dir, &new_paths[i][j + 1]);
93                  free(new_paths[i]);
94                  new_paths[i] = copy_dir;
95                  copy_dir = NULL;
96              }
97          }
98      }
99      *i = change - 1;
100     return new_paths;
101 }
102
103 void output(char **str, char delim, int size)
104 {
105     int i;
106     for(i = 0; i < size; i++)
107     {
108         printf("%s%c", str[i], delim);
109     }
110     printf("%s", str[i]);
111 }
112
113 void show_error(char *str, int column)
114 {
115     printf("%s\n", str);
116     for(int j = 0; j < column; j++)
117         printf(" ");
```

```
118     printf("^\n");
119 }
120
121 void output_error(char *str, int mode, int column)
122 {
123     if(mode == -1)
124         printf("Path length limit exceeded ( < 260)\n");
125     if(mode == -2)
126     {
127         show_error(str, column);
128         printf("Use rule violated for ':' in %d column\n", column);
129     }
130     if(mode == -3)
131     {
132         show_error(str, column);
133         printf("Illegal character for filename\n");
134     }
135     if(mode == -4)
136         printf("Error in memory allocation function\n");
137 }
138
139 int main()
140 {
141     char *name, *dir, *paths, **new_paths;
142     char delim;
143     int i, check_str, size;
144
145     printf("delim: ");
146     delim = getchar();
147     while(getchar() != '\n')
148         continue;
149
150     printf("user name: ");
151     name = input();
152
153     printf("dir: ");
154     dir = input();
155     if((check_str = check(dir, &i, delim)) != 0)
156     {
157         output_error(dir, check_str, i);
158         return -1;
159     }
160
161     printf("paths: ");
162     paths = input();
163     if((check_str = check(paths, &i, delim)) != 0)
164     {
165         output_error(paths, check_str, i);
166         return -1;
167     }
168
169     new_paths = process(dir, paths, delim, &size);
170     for(int j = 0; j < size; j++)
171     {
172         if((check_str = check(new_paths[j], &i, delim)) != 0)
173         {
174             output_error(new_paths[j], check_str, i);
175             return -1;
176         }
177     }
178
179
```

```
180     printf("new paths: ");
181     output(new_paths, delim, size);
182
183     free(name);
184     free(dir);
185     free(paths);
186     free_paths(new_paths, size);
187     printf("\nEnd.\n");
188
189     return 0;
190 }
```

Strings.c

```
1   #include <stdlib.h>
2
3   int is_letter(char c)
4   {
5       if((c > 172 || c < 141) && (c < 101 || c > 132))
6           return -1;
7       return 0;
8   }
9
10  int slen(char *str)
11  {
12      if(str == NULL)
13          return 0;
14      int i = 0;
15      while(str[i] != '\0')
16          i++;
17      return i;
18  }
19
20  int sspn(char *str1, char *str2)
21  {
22      if(str1 == NULL || str2 == NULL)
23          return -1;
24      int size1 = slen(str1), size2 = slen(str2);
25      for(int i = 0; i < size1; i++)
26      {
27          for(int j = 0; j < size2; j++)
28          {
29              if(str1[i] == str2[j])
30                  return i;
31          }
32      }
33      return size1;
34  }
35
36
37  int scmp(char *str1, char *str2)
38  {
39      int size1 = slen(str1), size2 = slen(str2), size;
40
41      if(size1 > size2)
42          size = size1;
43      else
44          size = size2;
```

```
45
46      for(int i = 0; i < size; i++)
47      {
48          if(str1[i] != str2[i])
49              return str1[i] - str2[i];
50      }
51      return 0;
52  }
53
54  char *scpy(char *dest, char *src)
55  {
56      if(src == NULL || dest == NULL)
57          return NULL;
58
59      int size = slen(src);
60      for(int i = 0; i < size; i++)
61          dest[i] = src[i];
62      dest[size] = '\0';
63      return dest;
64  }
65
66  int scout(char *str, char delim)
67  {
68      if(str == NULL)
69          return -1;
70
71      int cout = 0, size = slen(str);
72      for(int i = 0; i < size; i++)
73      {
74          if(str[i] == delim)
75              cout++;
76      }
77      return cout;
78  }
79
80  char *stok(char *str, char delim)
81  {
82      if(str == NULL)
83          return NULL;
84
85      int size = slen(str);
86      for(int i = 0; i < size + 1; i++)
87      {
88          if(str[i] == delim)
89          {
90              str[i] = '\0';
91              return str;
92          }
93          else if(str[i] == '\0')
94              return str;
95      }
96
97      return NULL;
98  }
99
100 char *concat (char *dest, char *append)
101 {
102     if(dest == NULL || append == NULL)
103         return NULL;
104
105     int size_dest = slen(dest);
106     int size_append = slen(append);
```

```
107     int j = 0;
108     for(int i = size_dest; i < size_dest + size_append; i++)
109     {
110         dest[i] = append[j];
111         j++;
112     }
113     return dest;
114 }
```