Day 21 Documentation

Md. Mahfuj Hasan Shohug

BDCOM0019

1. **Exercise 1-16:**

Problem: Revise the main routine of the longest-line program so it will correctly print the length of arbitrary long input lines, and as much as possible of the text.

Analysis: Here in my updated code, I am using pointer instated of array and get input the string to print the max longest line of input.

In the program reads the remaining characters until a newline character or EOF is detected when a line is longer than or equal to the maximum permitted length (MAXLINE - 1). This guarantees that the entire line will be received.

Here I am take some sample test case on this program:

```
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_21\Exercise_1_16.exe
Enter any string:
This is a sample line.
This is another line.
And this is the longest line in this input.
^Z

String length: 44
The longest string is: And this is the longest line in this input.


--------------------------------
Process exited after 4.29 seconds with return value 0
Press any key to continue . . .
```

```
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_21\Exercise_1_16.exe
Enter any string:
12345
123
1234567
12345
^Z

String length: 8
The longest string is: 1234567


--------------------------------
Process exited after 16.84 seconds with return value 0
Press any key to continue . . .
```

```
D:\Repository\MdMahfujHasanShohug\C&DS\Day_21\Exercise_1_16.exe
Enter any string:


^Z

String length: 1
The longest string is:


--------------------------------
Process exited after 3.142 seconds with return value 0
Press any key to continue . . .
```

```
D:\Repository\MdMahfujHasanShohug\C&DS\Day_21\Exercise_1_16.exe
Enter any string:
1234567890
12345678901234567890
123456
1234567890123
^Z


Storage limit exceeded by: 11
String length: 21
Please enter a maximum 10 length string

--------------------------------
Process exited after 46.86 seconds with return value 0
Press any key to continue . . .
```

On this test case my string maximum limit was 10.

Source Code:

```
#include <stdio.h>

#define MAXLENGTH 100

int getLine(char *, int);
void copy_long_str(char *, char *);

/*****************************************************
 * Function Name: main
 * Description: Reads lines of input from the user, finds the longest line, and prints it.
 * Parameters:
 *   - argc: Number of command-line arguments
 *   - argv: Array of command-line arguments
 * Returns: 0 on successful execution
 *****************************************************/
int main(int argc, char *argv[])
{
   int len, max = 0;
   char line[MAXLENGTH], longest[MAXLENGTH];
   char *linePtr = line;       // Pointer to the line array
   char *longestPtr = longest;  // Pointer to the longest array

   printf("Enter any string:\n");
   while ((len = getLine(linePtr, MAXLENGTH)) > 0)  // Read lines until len becomes 0
   {
      if (len > max)  // Check if the current line is longer than the previously stored longest line
      {
         max = len;  // Update the maximum length
         copy_long_str(longestPtr, linePtr);  // Copy the current line to the longest line buffer
      }
   }
```

```
    if (max > 0)  // Check if there is a longest line
    {
      if (max > MAXLENGTH)  // Check if the longest line exceeds the maximum allowed length
      {
        printf("\n\nStorage limit exceeded by: %d", max - MAXLENGTH);
        printf("\nString length: %d", max);
        printf("\nPlease enter a maximum %d length string", MAXLENGTH);
      }
      else
      {
        printf("\nString length: %d", max);
        printf("\nThe longest string is: %s", longest);
      }
    }
    printf("\n");
    return 0;
  }

  /****************************************************
   * Function Name: getLine
   * Description: Reads a line of input and stores it in the provided buffer.
   * Parameters:
   *   - line: Pointer to the buffer for storing the line
   *   - limit: Maximum length of the line buffer
   * Returns: The actual length of the line read
   ****************************************************/
  int getLine(char *line, int limit)
  {
    int i, c;
    for (i = 0; i < limit - 1 && (((c = getchar()) != EOF) && (c != '\n')); i++)
      *(line++) = c;  // Store the character in the current pointer position and move the pointer

    if (i == (limit - 1))  // If the loop exited due to reaching the limit
    {
      while ((c = getchar()) != '\n')  // Continue reading characters until newline
      {
        ++i;
      }
    }

    if (c == '\n')
    {
      *(line++) = c;  // Store the newline character and move the pointer
      ++i;
    }

    *line = '\0';
    return i;  // Return the actual length of the line
  }

  /****************************************************
   * Function Name: copy_long_str
   * Description: Copies a string from the source to the destination.
   * Parameters:
   *   - to: Pointer to the destination string
   *   - from: Pointer to the source string
   ****************************************************/
  void copy_long_str(char *to, char *from)
  {
    while ((*to = *from) != '\0')  // Copy characters until the null terminator is encountered
    {
      ++to;
      ++from;
    }
}
```

2. Exercise 1-18:

Problem: Write a program to remove trailing blanks and tabs from each line of input, and to delete entirely blank lines.

Analysis: Here on this program I am take the books code function and after that add remove_trailing_blanks() function for remove trailing blanks and tabs from each line of input.

The getLine() function reads a line of input using the getter and stores it in the given buffer. It ends the line with a null character and returns the actual length of the line.

After that the remove_trailing_blanks() function iterates over the characters of the line starting from the end and until the first non-blank character is found. It then adds newline characters and ends the line after that.

Here I show some test case for this program:

```
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_21\Exercise_1_18.exe
abc
Input: abc
Input Length: 7
Output: abc
Output Length: 4

1234
Input: 1234
Input Length: 4
Output: 1234
Output Length: 5


Input:
Input Length: 10

Input:
Input Length: 0
a
Input: a
Input Length: 4
Output: a
Output Length: 2

Hello BDCOM!!
Input: Hello BDCOM!!
Input Length: 17
Output: Hello BDCOM!!
Output Length: 14

^Z

--------------------------------
Process exited after 97.32 seconds with return value 0
Press any key to continue . . .
```

Source code:

```
#include <stdio.h>
#include <string.h>

#define MAXLINE 100 /* maximum input line length */

int getline(char line[], int maxline);
void remove_trailing_blanks(char line[], int length);

/*****************************************************
 * Function Name: main
 * Description: Reads lines of input from the user, removes trailing blanks, and prints the result.
 * Parameters:
 *   - argc: Number of command-line arguments
 *   - argv: Array of command-line arguments
 * Returns: 0 on successful execution
 *****************************************************/
int main(int argc, char *argv[])
{
   int len; /* current line length */
   char line[MAXLINE]; /* current input line */

   while ((len = getline(line, MAXLINE)) > 0)
   {
      printf("Input: %s", line);
      printf("Input Length: %d\n", len - 1); // Subtract 1 to exclude newline character

      remove_trailing_blanks(line, len);

      if (line[0] != '\n')
      {
         printf("Output: %s", line);
         printf("Output Length: %d\n\n", len - 1 - (len - 1 - strlen(line))); // Subtract trailing blanks length
      }
   }

   return 0;
}

/*****************************************************
 * Function Name: getline
 * Description: Reads a line of input and stores it in the provided buffer.
 * Parameters:
 *   - s: Pointer to the buffer for storing the line
 *   - lim: Maximum length of the line buffer
 * Returns: The actual length of the line read
 *****************************************************/
int getline(char s[], int lim)
{
   int c, i;

   for (i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; ++i)
   {
      s[i] = c;
   }

   if (c == '\n')
   {
      s[i] = c;
      ++i;
   }

   s[i] = '\0';

   return i;
```

```
}

/**************************************************
 * Function Name: remove_trailing_blanks
 * Description: Removes trailing blanks and tabs from the end of a line.
 * Parameters:
 *   - line: Pointer to the line string
 *   - length: Length of the line
 **************************************************/
void remove_trailing_blanks(char line[], int length)
{
    int i;

    // Start from the end of the line and move backward
    for (i = length - 2; i >= 0; --i)
    {
        if (line[i] != ' ' && line[i] != '\t')
        {
            break;
        }
    }

    line[i + 1] = '\n'; // Add back the newline character
    line[i + 2] = '\0'; // Terminate the line after the newline character
}
```