Day 17 Documentation

Md. Mahfuj Hasan Shohug

BDCOM0019

(Reading time 6.15 hour and problem solving, debug time and doc 2 hour 30 min)

1. Exercise 5-11:

Problem: Modify the program entab and detab (written as exercises in Chapter 1) to accept a list of tab stops as arguments. Use the default tab settings if there are no arguments.

Analysis for Entab:

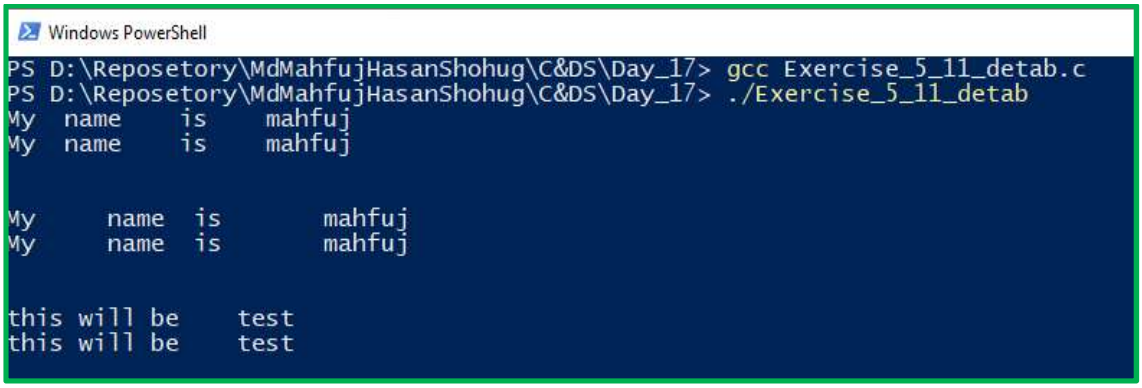
I am doing the Entab and also Detab program in two different program file, with the help of the book I am using their provided function name and variable for better learning.

- In the entab function counts the number of consecutive spaces (space_count) by scanning each character in the input line.
- It substitutes the tab character (\t) for each subsequent space. This makes sure that spaces in the output that are not a part of a tab-width sequence are preserved.

Analysis for Detab:

- The tab width and tab stop arguments in my dtab program in command-line arguments will be executed.
- It utilizes the default tab width of 4 if no input for tab width is provided. Based on the tab width and tab stop.
- In the program reads the input lines and replaces the tabs with the necessary number of spaces.

Test Case (Detab):



```
^Z
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17> ./Exercise_5_11_detab 4 4 8
This is    mahfuj  hasan        shohug
This is    mahfuj  hasan    shohug
```

```
^Z
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17> ./Exercise_5_11_detab.exe 4 0 2 6
Invalid tab stop. Ignoring.
This   is   spaceing
This   is   spaceing
```

Test for (Entab):

```
^Z
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17> ./Exercise_5_11_entab.exe
This    is    a    line    with    multiple    spaces.
This    is      a       line    with    multiple        spaces.


    Leading spaces are preserved.
        Leading spaces are preserved.


^Z
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17> ./Exercise_5_11_entab.exe 0
Invalid tab width. Using default.
```

Three different test case I used for this program.

Source code (Entab):

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

#define MAXLINE 1000
#define DEFAULT_TAB_WIDTH 4

/*******************************************************************************
 * Function: get_line                                                          *
 * Description: Reads a line of input from the user.                           *
 *                                                                             *
 * Inputs:                                                                     *
 *   - line: An array to store the line read from input.                       *
 *   - maxline: The maximum length of the line that can be stored.             *
 *                                                                             *
 * Outputs:                                                                    *
 *   - Returns the number of characters read, including the newline character. *
 *******************************************************************************/
int get_line(char line[], int maxline)
{
    int c, i;
    for (i = 0; i < maxline - 1 && (c = getchar()) != EOF && c != '\n'; ++i)
        line[i] = c;
```

```
    if (c == '\n')
            {
        line[i] = c;
        ++i;
    }
    line[i] = '\0';
    return i;
}

/*****************************************************************************************
 * Function: entab                                                                       *
 * Description: Replaces spaces with tabs and spaces according to the given tab width.  *
 *                                                                                       *
 * Inputs:                                                                               *
 *   - line: An array representing the input line to be modified.                        *
 *   - tab_width: The desired width of each tab in spaces.                               *
 *                                                                                       *
 * Outputs:                                                                              *
 *   - Modifies the 'line' array to contain the modified line with tabs and spaces.     *
 *****************************************************************************************/
void entab(char line[], int tab_width)
{
    int i, j, space_count, tab_count;
    char entab_line[MAXLINE];
    i = j = space_count = tab_count = 0;

    while (line[i] != '\0')
            {
        if (line[i] == ' ')
                        {
            space_count++;
            if (space_count == tab_width)
                                {
                tab_count++;
                space_count = 0;
            }
        } else
                        {
            // Convert accumulated spaces to tabs
            for (; tab_count > 0; tab_count--)
                                {
                entab_line[j] = '\t';
                j++;
            }

            // Add remaining individual spaces
            if (space_count > 0)
                                {
                for (; space_count > 0; space_count--)
                                        {
                    entab_line[j] = ' ';
                    j++;
                }
            }

            // Add the non-space character
            entab_line[j] = line[i];
            j++;
```

```
        }
        i++;
    }

    // Terminate the modified line
    entab_line[j] = '\0';

    // Copy the modified line back to the original line array
    for (i = 0; entab_line[i] != '\0'; i++)
        line[i] = entab_line[i];
    line[i] = '\0';
}
/*****************************************************************************************
 * Function: main                                                                        *
 * Description: Entry point of the program. Reads input lines and performs entab operation.*
 *                                                                                       *
 * Inputs:                                                                               *
 *   - argc: Number of command-line arguments passed to the program.                     *
 *   - argv: Array of strings containing the command-line arguments.                     *
 *                                                                                       *
 * Outputs:                                                                              *
 *   - Returns 0 to indicate successful execution of the program.                        *
 *****************************************************************************************/
int main(int argc, char *argv[])
{
    int tab_width = DEFAULT_TAB_WIDTH;

    if (argc > 1)
            {
        tab_width = atoi(argv[1]);
        if (tab_width <= 0)
                        {
            printf("Invalid tab width. Using default.\n");
            tab_width = DEFAULT_TAB_WIDTH;
        }
    }

    char line[MAXLINE];
    while (get_line(line, MAXLINE) > 0)
            {
        entab(line, tab_width);
        printf("%s", line);
    }

    return 0;
}
```

Source code (Detab)

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

#define MAXLINE 1000
#define DEFAULT_TAB_WIDTH 4

/*****************************************************************************
```

```
 * Function: get_line                                          *
 * Description: Reads a line of input from the user.                    *
 *                                                             *
 * Inputs:                                                     *
 *   - line: An array to store the line read from input.                *
 *   - maxline: The maximum length of the line that can be stored.            *
 *                                                             *
 * Outputs:                                                    *
 *   - Returns the number of characters read, including the newline character.  *
 ********************************************************************************/
int get_line(char line[], int maxline)
{
   int c, i;
   for (i = 0; i < maxline - 1 && (c = getchar()) != EOF && c != '\n'; ++i)
      line[i] = c;
   if (c == '\n')
            {
      line[i] = c;
      ++i;
   }
   line[i] = '\0';
   return i;
}

/*************************************************************************
 * Function: detab                                             *
 * Description: Replaces tab characters with spaces in the given line.  *
 *                                                             *
 * Inputs:                                                     *
 *   - line: The input line to detab.                                 *
 *   - tab_width: The width of each tab character.                        *
 *   - tab_stops: Array of custom tab stops.                          *
 *   - num_stops: The number of custom tab stops.                         *
 *                                                             *
 * Outputs:                                                    *
 *   - Modifies the 'line' array in-place by replacing tabs with spaces.*
 *************************************************************************/
void detab(char line[], int tab_width, int *tab_stops, int num_stops)
{
   int i, j, k, tab_count, stop_index;
   char detab_line[MAXLINE];
   i = j = tab_count = stop_index = 0;

   while (line[i] != '\0')
            {
      if (line[i] == '\t')
                        { // If a tab character is found
         if (num_stops > 0)
                                 { // If custom tab stops are specified
            while (j % tab_stops[stop_index] != 0) // Fill with spaces until the next custom tab stop
               detab_line[j++] = ' ';
            stop_index = (stop_index + 1) % num_stops; // Move to the next custom tab stop
         } else
                                 { // If no custom tab stops are specified, use the default tab width
            while (j % tab_width != 0) // Fill with spaces until the next default tab stop
               detab_line[j++] = ' ';
         }
      } else
```

```
                        { // If a non-tab character is found, copy it to the detabbed line
        detab_line[j++] = line[i];
      }
      i++;
    }
    detab_line[j] = '\0';

    // Copy the detabbed line back to the original line array
    for (k = 0; detab_line[k] != '\0'; k++)
      line[k] = detab_line[k];
    line[k] = '\0';
}

/*****************************************************************************************
 * Function: main                                                    *
 * Description: Entry point of the program. Reads input lines and performs detab operation.*
 *                                                                   *
 * Inputs:                                                           *
 *   - argc: Number of command-line arguments passed to the program.                 *
 *   - argv: Array of strings containing the command-line arguments.                 *
 *                                                                   *
 * Outputs:                                                          *
 *   - Returns 0 to indicate successful execution of the program.                    *
 *****************************************************************************************/
int main(int argc, char *argv[])
{
    int tab_width = DEFAULT_TAB_WIDTH; // Default tab width
    int tab_stops[MAXLINE]; // Array to store custom tab stops
    int i, num_stops = 0; // Number of custom tab stops

    if (argc > 1)
            {
      tab_width = atoi(argv[1]); // Set tab width from command line argument
      if (tab_width <= 0)
                        {
        printf("Invalid tab width. Using default.\n");
        tab_width = DEFAULT_TAB_WIDTH;
      }
    }

    for (i = 2; i < argc; i++)
            {
      tab_stops[num_stops] = atoi(argv[i]); // Store custom tab stops from command line arguments
      if (tab_stops[num_stops] <= 0)
                        {
        printf("Invalid tab stop. Ignoring.\n");
      } else
                        {
        num_stops++;
      }
    }

    char line[MAXLINE]; // Array to store input line
    while (get_line(line, MAXLINE) > 0)
            { // Read input lines until end of file
      detab(line, tab_width, tab_stops, num_stops); // Replace tabs with spaces
      printf("%s", line); // Print detabbed line
    }
```

```
    return 0;
}
```

2. Exercise 5-13.

Problem: Write the program tail, which prints the last n lines of its input. By default, n is set to 10, let us say, but it can be changed by an optional argument so that tail –n prints the last n lines. The program should behave rationally no matter how unreasonable the input or the value of n. Write the program so it makes the best use of available storage; lines should be stored as in the sorting program of Section 5.6, not in a two-dimensional array of fixed size.

Analysis:

This code is a program that reads lines of input from the user and prints the last '-n' lines, where '-n' is either specified as a command-line argument or defaults to 10 if no argument is provided. It defines three functions:

- ReadLines: This function reads lines from standard input and stores them in an array of pointers. It returns the number of lines read. We used it before
- print_tail: This function prints the last '-n' lines from the given array of pointers.
- main: The main function is the entry point of the program. It processes the command-line arguments, calls the readline function to read the input lines, and then calls the print_tail function to print the desired lines.

Test case:

```
PS D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_17> ./Exercise_5_13.exe -n 2
1
2
3
4
5
6
^Z
5
6
PS D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_17>
```

```
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17> ./Exercise_5_13.exe -n
1
2
3
4
5
6
7
8
9
0
^Z
1
2
3
4
5
6
7
8
9
0
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17>
```

Default is 10.

```
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17> ./Exercise_5_13.exe -n 3
a
b
c
d
e
f
g
^Z
e
f
g
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17>
```

```
PS D:\Repository\MdMahfujHasanShohug\C&DS\Day_17> ./Exercise_5_13.exe -n -1
Invalid input. Using default value.
```

Source code:

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

#define MAX_LINES 1000
#define MAX_LINE_LENGTH 100

/*******************************************************************************
 * Function: readlines                                                         *
 * Description: Reads lines from stdin and stores them in an array of pointers. *
 *          It returns the number of lines read.                               *
 * Inputs:                                                                     *
 *   - lines: An array of pointers to store the lines.                         *
 *   - maxlines: The maximum number of lines that can be stored.               *
```

```
 * Outputs:                                                    *
 *   - Returns the number of lines read.                       *
 ********************************************************************************/
int readlines(char *lines[], int maxlines) // Given this function on the book
{
   int num_lines = 0;
   char line[MAX_LINE_LENGTH];
   while (fgets(line, MAX_LINE_LENGTH, stdin) != NULL)
   {
      lines[num_lines] = malloc(strlen(line) + 1);
      strcpy(lines[num_lines], line);
      num_lines++;
      if (num_lines >= maxlines)
      {
         printf("Input exceeds maximum line limit. Only storing %d lines.\n", maxlines);
         break;
      }
   }
   return num_lines;
}

/********************************************************************************
 * Function: print_tail                                        *
 * Description: Prints the last 'n' lines from the given array of pointers.     *
 * Inputs:                                                     *
 *   - lines: An array of pointers containing the lines.                      *
 *   - n: The number of lines to print.                        *
 * Outputs:                                                    *
 *   - None. Prints the lines to stdout.                       *
 ********************************************************************************/
void print_tail(char *lines[], int n)
{
            int i;
   for (i = 0; i < n; i++)
   {
      printf("%s", lines[i]);
      free(lines[i]); // Free the memory allocated for each line
   }
}

/*************************************************************************
 * Function: main                                              *
 * Description: The entry point of the program.                *
 * Inputs:                                                     *
 *   - argc: The number of command-line arguments.             *
 *   - argv: An array of strings containing the command-line arguments. *
 * Outputs:                                                    *
 *   - Returns 0 on successful execution.                      *
 *************************************************************************/
int main(int argc, char *argv[])
{
   int n = 10; // Default number of lines to print

   if (argc > 2 && strcmp(argv[1], "-n") == 0) // for -n
   {
      n = atoi(argv[2]); // Convert the argument to an integer
      if (n <= 0)
      {
```

```
            printf("Invalid input. Using default value.\n");
            n = 10; // Invalid input, use the default
        }
    }

    char *lines[MAX_LINES]; // Array to store lines
    int num_lines = readlines(lines, MAX_LINES);

    if (num_lines > 0)
    {
        int start = (num_lines > n) ? num_lines - n : 0;
        print_tail(&lines[start], num_lines - start);
    }

    return 0;
}
```