

### Learning summary:

**Study: 7 Hours (I have to take more time in learning to understand the topic as Pointer is a sensitive topic in C)**

**Exercises: 4 Hours**

### Documentation of Day 17

#### Exercise 5-11:

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.

#### Source Code:

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

#define MAX_TAB_STOPS 100

void entab(int tab_stops[], int num_stops) {
    int c;
    int current_pos = 0;
    int space_count = 0;

    while ((c = getchar()) != EOF) {
        if (c == ' ') {
            space_count++;
            current_pos++;

            if (current_pos % tab_stops[0] == 0) {
                putchar('\t');
                space_count = 0;
            }
        } else {
            while (space_count > 0) {
                putchar(' ');
                space_count--;
            }

            putchar(c);
            current_pos++;

            if (c == '\n')
                current_pos = 0;
        }
    }
}
```

```

        if (current_pos >= tab_stops[0])
            tab_stops++;
    }
}

void detab(int tab_stops[], int num_stops) {
    int c;
    int current_pos = 0;
    int i;
    while ((c = getchar()) != EOF) {
        if (c == '\t') {
            int next_tab_stop = tab_stops[0] - (current_pos % tab_stops[0]);

            for (i = 0; i < next_tab_stop; i++) {
                putchar(' ');
                current_pos++;
            }

            tab_stops++;
        } else {
            putchar(c);
            current_pos++;

            if (c == '\n')
                current_pos = 0;

            if (current_pos >= tab_stops[0])
                tab_stops++;
        }
    }
}

```

```

int main(int argc, char *argv[]) {
    int tab_stops[MAX_TAB_STOPS];
    int num_stops = 0;
    int i;
    if (argc > 1) {
        for (i = 1; i < argc && num_stops < MAX_TAB_STOPS; i++) {
            tab_stops[num_stops] = atoi(argv[i]);
            num_stops++;
        }
    } else {
        // Default tab settings
    }
}

```

```
    tab_stops[0] = 8;
    num_stops = 1;
}
entab(tab_stops, num_stops);
// detab(tab_stops, num_stops);

return 0;
}
```

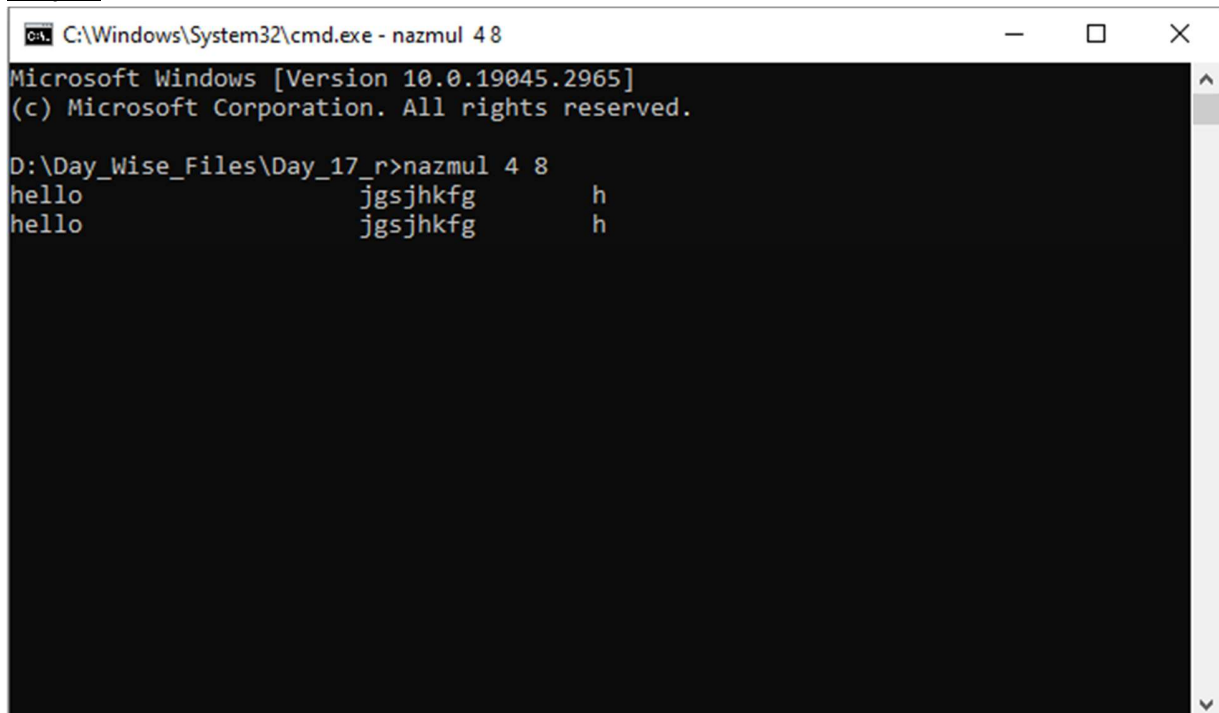
The modified program accepts a list of tab stops as command-line arguments. If no arguments are provided, it uses default tab settings. The program has two functions, entab and detab.

In entab, the program reads input and replaces consecutive spaces with tabs whenever possible based on the tab stops. It tracks the current position and the number of consecutive spaces.

In detab, the program reads input and replaces tabs with spaces based on the tab stops. It calculates the number of spaces needed to reach the next tab stop and inserts them.

The program loops through the input character by character, printing the modified output as it goes.

#### Output:



```
C:\Windows\System32\cmd.exe - nazmul 4 8
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

D:\Day_Wise_Files\Day_17_r>nazmul 4 8
hello          jgsjhkfg      h
hello          jgsjhkfg      h
```

**Exercise 5-13:**

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

**Source Code:**

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

#define MAX_LINE_LENGTH 1000
#define DEFAULT_LINES 10

void print_last_lines(FILE *file, int num_lines) {
    char **lines = (char **)malloc(num_lines * sizeof(char *));
    if (lines == NULL) {
        fprintf(stderr, "Error: Insufficient memory.\n");
        exit(1);
    }

    int line_count = 0;
    char line[MAX_LINE_LENGTH];

    // Store the last n lines in the circular buffer
    while (fgets(line, MAX_LINE_LENGTH, file) != NULL) {
        lines[line_count % num_lines] = strdup(line);
        line_count++;
    }

    int start = line_count > num_lines ? line_count - num_lines : 0;
    int end = line_count;
    int i;
    for (i = start; i < end; i++) {
        printf("%s", lines[i % num_lines]);
    }
}
```

```

        free(lines[i % num_lines]);
    }

    free(lines);
}

int main(int argc, char *argv[]) {
    int num_lines = DEFAULT_LINES;

    if (argc > 1) {
        num_lines = atoi(argv[1]);
        if (num_lines <= 0) {
            fprintf(stderr, "Error: Invalid number of lines.\n");
            return 1;
        }
    }

    print_last_lines(stdin, num_lines);

    return 0;
}

```

The code defines the `print_last_lines` function, which takes a file and the number of lines to print as parameters. It creates a dynamic array to store the last `n` lines.

The function reads input from the file and stores each line in the dynamic array, overwriting older lines as needed to maintain a circular buffer of size `n`.

After reading all the lines, the function determines the starting and ending indices of the lines to print based on the line count and the specified number of lines.

It then loops through the lines to be printed and outputs them to the console.

The main function reads the number of lines from the command-line argument (or uses the default value) and calls `print_last_lines` with the standard input as the file parameter, effectively printing the last `n` lines from the input.

#### **Inputs:**

Line 1  
 Line 2  
 Line 3  
 Line 4  
 Line 5  
 Line 6

Line 7  
Line 8  
Line 9  
Line 10  
Line 11  
Line 12

### Output:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

D:\Day_Wise_Files\Day_17_r>gcc Ex_5-13.c -o 5-13

D:\Day_Wise_Files\Day_17_r>5-13 4
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
Line 11
Line 12

^Z
Line 10
Line 11
Line 12

D:\Day_Wise_Files\Day_17_r>5-13 4
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
Line 11
Line 12
^Z
Line 9
Line 10
Line 11
Line 12

D:\Day_Wise_Files\Day_17_r>
```