March 8th, 2018

CSE344 – System Programming - Homework #1

Submission deadline: 23:55, March 22th, 2018

The **Tagged Image File Format** or **Tiff** is a format for storing digital images. A digital image can be thought of as a rectangular matrix of non-negative integers, where every point (or pixel) of the image is denoted by a single value. Tiff is a powerful file format that can store multiple scenes in a single file and compress them quite effectively.

This homework's objective is to write a POSIX compatible C program that admits as command line argument the path of a black & white, uncompressed locally stored tiff file and prints its pixels on the standard output using 0's for black and 1's for white.

Example input image (25x25) (test.tif):

Expected output on screen:

```
$./tiffProcessor test.tif
Width : 25 pixels
Height : 25 pixels
Byte order: Intel
11111111111111111111111110
11111111111111111111111110
11111111111111111111111101
11111111111111111111111011
11111111111111111111110111
11111111111111111111101111
11111111111111111111011111
11111111111111111110111111
11111111111111111101111111
11111111111111111011111111
11111111111111110111111111
11111111111111101111111111
11111111111111011111111111
11111111111110111111111111
11111111111101111111111111
11111111110111111111111111
11111111101111111111111111
11111110111111111111111111
11111101111111111111111111
11111011111111111111111111
11110111111111111111111111
11101111111111111111111111
11011111111111111111111111
00111111111111111111111111
```

Rules:

a) You are not allowed to use any 3rd party libraries for loading tiff files. Use POSIX and Standard C libraries. You can write your program using C11 standards.

b) Your program must handle eventual errors gracefully according to the POSIX way.

c) If the command line argument is missing your program must print usage information.

d) Your program must work with black & white uncompressed tiff files in either Intel of Motorola byte order.

e) Make sure your program doesn't have any memory leak and runs clean. You can check that and many other things using Valgrind tool. Just run your program from shell like this :

valgrind ./tiffprocessor test.tif

It will output whether your program has memory leak or not. Also it will tell you about invalid memory accesses, uninitialized memory and much more. For example, if your program gives segmentation fault, most of the time Valgrind will tell you the reason. So it is also a very powerful tool for debugging.
A program with no errors or leaks will give you an output like this:

```
gtusvy@debian:~/Downloads$ valgrind ./example deneme.txt  asd
==1619== Memcheck, a memory error detector
==1619== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==1619== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright info
==1619== Command: ./example deneme.txt asd
==1619==
*****************************
File name: deneme.txt
-----------------------------
Line 3 : ,sadjhgdsa asdhlksajhd asdkhsadlsahlkjdsa
Line 3 : ,sadjhgdsa asdhlksajhd asdkhsadlsahlkjdsa
Line 4 : asdhgjsahd askjdhlksa sadkjhsalkd
Line 5 : ashdgsahd asd asd asd asd
Line 5 : ashdgsahd asd asd asd asd
Line 5 : ashdgsahd asd asd asd asd
Line 5 : ashdgsahd asd asd asd asd
Line 6 : asd asd asd
Line 6 : asd asd asd
Line 6 : asd asd asd
Line 8 : üasd asdasd
Line 8 : üasd asdasd
Line 8 : üasd asdasd
Line 9 : asd
Line 10 : ads asd asd
Line 10 : ads asd asd

Total occurance of asd: 16
-----------------------------
==1620==
==1620== HEAP SUMMARY:
==1620==     in use at exit: 0 bytes in 0 blocks
==1620==   total heap usage: 2 allocs, 2 frees, 1,152 bytes allocated
==1620==
==1620== All heap blocks were freed -- no leaks are possible
==1620==
==1620== For counts of detected and suppressed errors, rerun with: -v
==1620== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==1619==
==1619== HEAP SUMMARY:
==1619==     in use at exit: 0 bytes in 0 blocks
==1619==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==1619==
==1619== All heap blocks were freed -- no leaks are possible
==1619==
==1619== For counts of detected and suppressed errors, rerun with: -v
==1619== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

We will be expecting your homeworks to run just like the one above. For each process the output should be: “all heap blocks are freed – no leaks are possible” and “0 errors” when we run it with Valgrind. A few errors is OK if they are not harmful.

f) *Your program MUST not give “segmentation fault” or any other fault that causes your program to end in a bad way. It is better for you to send a little part of the homework with clean and executing code instead of full homework with faults like this.*

g) **Provide a makefile to compile your homework. Do not run your program inside makefile. Just compile it.**

h) Think smart and try to do the job with as short as possible and easily understandable code. Do not send homeworks with 1000 lines that can be done in 200 lines. When you find a solution to a problem, think like this: “What could be a better solution...”.
i) Test your homework using the Virtual Machine given to you.
j) No late submissions.

Tip for this homework: You can produce uncompressed tiff files using Gimp and ImageJ.

Homework format:
StudentID_HW1_CSE344.tar.gz
|→ Makefile
|→ StudentID_main.c
|→ … (Any other source files)

Teaching assistant: Ahmet Soyyiğit
e-mail: asoyyigit@gtu.edu.tr

Good luck.