

# TIFF TO PNG (8BIT GRAYSCALE)

Several thin, parallel white lines of varying lengths and orientations are scattered across the upper right portion of the blue background, creating a dynamic, abstract pattern.

Group Number : 15

Group Members:

IMT2020124 Tavva Akhil  
IMT2020102 Tejdeep Gutta  
IMT2020116 Bhavjyot Singh Chadha  
IMT2020540 Sasank Karamsetty  
IMT2020502 Monjoy Narayan Choudhury  
IMT2020506 Jayam Shanmukha Bhuvan

## Overview

We try to convert a TIFF (Tag Image File Format) to a 8bit-grayscale PNG (Portable Network Graphics) file using the concepts and language construct of C we have learnt during the course ESS-111: Programming in C by Prof. Yashwant Kanetkar.

In the top-level abstraction of our project, we try to read a TIFF file, read and store the necessary information like pixel data, height, width which we convert into an 8-bit grayscale pixel and write in a PNG file using the features of C (without any external library) programing and compression library Zlib (for CRC in PNG).

**sample\_1.tiff**



**TargetImage.png**



## Work distribution:

- IMT2020502-Monjoy Narayan Choudhury:

In File tiff2.c

Understood the basic structure /file organization of a tiff file.

Defined necessary structs and functions like `open_TIFF_File` which is the heart of the whole program on the tiff side

Dealt with endianness check and necessary data conversion. Also created functions which read the data from the file and advanced the file pointer to necessary location.

Provided Bhuvan with the foundation to implement the functions in his part.

Read IFD tags and extracted the image dimension which was used by `png.c` code for output.

Functions/structs written: `open_TIFF_File`,  
`data_read_modify_function`, `IFD_structure(function)`, `IFD_tag_cases`.

Structs are defined in `opentiff.h` (note some declarations also are contributed to Bhuvan)

- IMT2020506- Jayam Shanmukha Bhuvan:  
In File tiff2.c

Copyrgb function:

This function takes the source file and offset value.

After reading endiation part and offset values of the given tiff file, this function is executed to copy the RGB values of given tiff image and store them in a struct containing 3 arrays of R, G, B values of all pixels. After storing R, G, B values of all pixels function returns the struct image\_array to open\_TIFF\_file function.

Convertrgb function:

This function takes a struct image\_array containing 3 arrays of R, G, B values from open\_TIFF\_File function and returns a file which contains grayscale values of all pixels. Grayscale value is calculated by given formula-

$$\text{Grayscale} = (0.3 * R) + (0.59 * G) + (0.11 * B)$$

- IMT2020124 -Tavva Akhil:

My part is about converting the RGB values into grayscale and extracting libz.a from zlib library

As we are getting the RGB values from the tiff part per pixel we are going to change the colour of each pixel by any of these formulas:

1) grayscale =  $(0.3 * R) + (0.59 * G) + (0.11 * B)$  or

2) grayscale =  $(R + G + B) / 3$

We can use a for loop starting with  $i=0$  and we can increment it until  $((\text{offset}-8) * \text{sizeof}(\text{char})/3)-1$  and we can use the above formula for each  $i$

To make the zlib library we open README and there it is given that to compile the program do `./configure` and `make test` then we get the required libz.a file and .so type files.

We can use this libz.a to compress the png files.

- IMT2020540-Sasank Karamsetty:

Makefile creation.

- IMT2020116-Bhavjyot Singh Chadha:

Studied the whole PNG format with Tejdeep and made the appropriate data structures for making the PNG image.

Created the appropriate structures of the header and chunks for the destination format along with Tejdeep.

Got the exact values of the header file from Tejdeep.

Read the PNG file and implemented the above-mentioned data structures.

Wrote the image data into a PNG file from the grayscale pixel data of the source image.

- IMT2020102 Tejdeep Gutta:

Studied the PNG format and gave the exact values of the PNG header.

Created the appropriate structures needed for making the PNG image.

Understood CRCs and helped Bhavjyot to code it.

# Prerequisite to open C project

## Zlib:

1. Zlib is attached with the submission as zlib-master.zip
2. Extract the zip
3. Open terminal in the folder zlib-master
4. Type the following command

./configure; make test

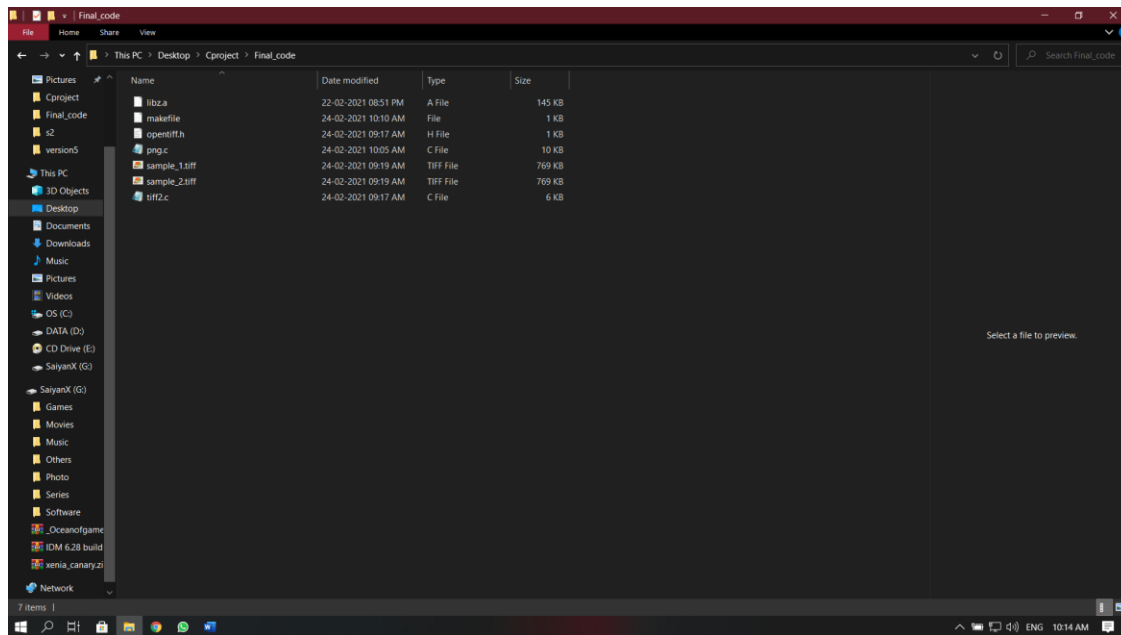
```
mnc@LAPTOP-P9LGWF8: /mnt/c/Users/ASUS/Desktop/Cproject/Final_code/zlib-master$ ./configure; make test
gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o obj/gzwrite.o gzwrite.c
gcc -shared -Wl,-soname,libz.so.1,-version-script,zlib.map -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o libz.so.1.2.11 adler32.lo crc32.lo deflate.lo
inffast.lo inflate.lo inftrees.lo trees.lo zutil.lo compress.lo uncompress.lo gzclose.lo gzlib.lo gzread.lo gzwrite.lo -lc
rm -f libz.so libz.so.1
ln -s libz.so.1.2.11 libz.so
ln -s libz.so.1.2.11 libz.so.1
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o examplesh example.o -L. libz.so.1.2.11
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o minigzipsh minigzip.o -L. libz.so.1.2.11
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -I. -D_FILE_OFFSET_BITS=64 -c -o example.o test/example.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o example64 example64.o -L. libz.a
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -I. -D_FILE_OFFSET_BITS=64 -c -o minigzip64.o test/minigzip.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o minigzip64 minigzip64.o -L. libz.a
hello world
zlib version 1.2.11 = 0x12b0, compile flags = 0xa9
uncompress(): hello, hello!
gzread(): hello, hello!
gzgets() after gzseek: hello!
inflate(): hello, hello!
large_inflate(): OK
after inflateSync(): hello, hello!
inflate with dictionary: hello, hello!
*** zlib test OK ***
hello world
zlib version 1.2.11 = 0x12b0, compile flags = 0xa9
uncompress(): hello, hello!
gzread(): hello, hello!
gzgets() after gzseek: hello!
inflate(): hello, hello!
large_inflate(): OK
after inflateSync(): hello, hello!
inflate with dictionary: hello, hello!
*** zlib shared test OK ***
hello world
zlib version 1.2.11 = 0x12b0, compile flags = 0xa9
uncompress(): hello, hello!
gzread(): hello, hello!
gzgets() after gzseek: hello!
inflate(): hello, hello!
large_inflate(): OK
after inflateSync(): hello, hello!
```

```
mnc@LAPTOP-P9LGWF8: /mnt/c/Users/ASUS/Desktop/Cproject/Final_code/zlib-master$ ./configure; make test
Checking for gcc...
Checking for shared library support...
Building shared library libz.so.1.2.11 with gcc.
Checking for size_t... Yes.
Checking for off64_t... Yes.
Checking for fseeko... Yes.
Checking for strerror... Yes.
Checking for unistd.h... Yes.
Checking for stdarg.h... Yes.
Checking whether to use vs[n]printf() or s[n]printf()... using vs[n]printf().
Checking for vsnprintf() in stdio.h... Yes.
Checking for return value of vsnprintf()... Yes.
Checking for attribute(visibility) support... Yes.
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -I. -c -o example.o test/example.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o adler32.o adler32.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o crc32.o crc32.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o deflate.o deflate.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o inffast.o inffast.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o inflate.o inflate.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o inftrees.o inftrees.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o trees.o trees.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o zutil.o zutil.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o compress.o compress.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o uncompress.o uncompress.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o gzclose.o gzclose.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o gzlib.o gzlib.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o gzread.o gzread.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -c -o gzwrite.o gzwrite.c
ar rc libz.a adler32.o crc32.o deflate.o inffast.o inflate.o inftrees.o trees.o zutil.o compress.o uncompress.o gzclose.o gzlib.o gzread.o gzwrite.o
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o example example.o -L. libz.a
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -I. -c -o minigzip.o test/minigzip.c
gcc -O3 -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -o minigzip minigzip.o -L. libz.a
gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o obj/adler32.o adler32.c
gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o obj/crc32.o crc32.c
gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o obj/deflate.o deflate.c
gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o obj/inffast.o inffast.c
gcc -O3 -fPIC -D_LARGEFILE64_SOURCE=1 -DHAVE_HIDDEN -DPIC -c -o obj/inflate.o inflate.c
```

# Instructions to open C project

1.Extract the project zip file

2. The folder must contain the following files (note it also has zlib-master.zip and readme.pdf)

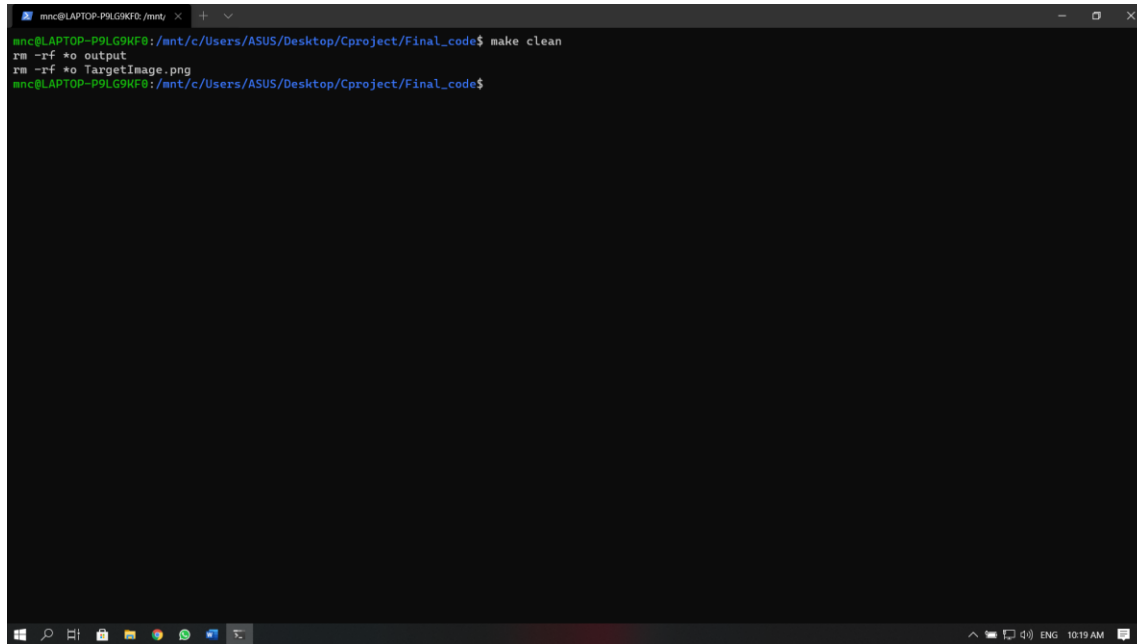


3.Open the folder in Terminal

(Please note all the demonstration will be done on WSL (windows subsystem on linux))



4.Type in make clean . This helps to remove any intermediate files of compilation.

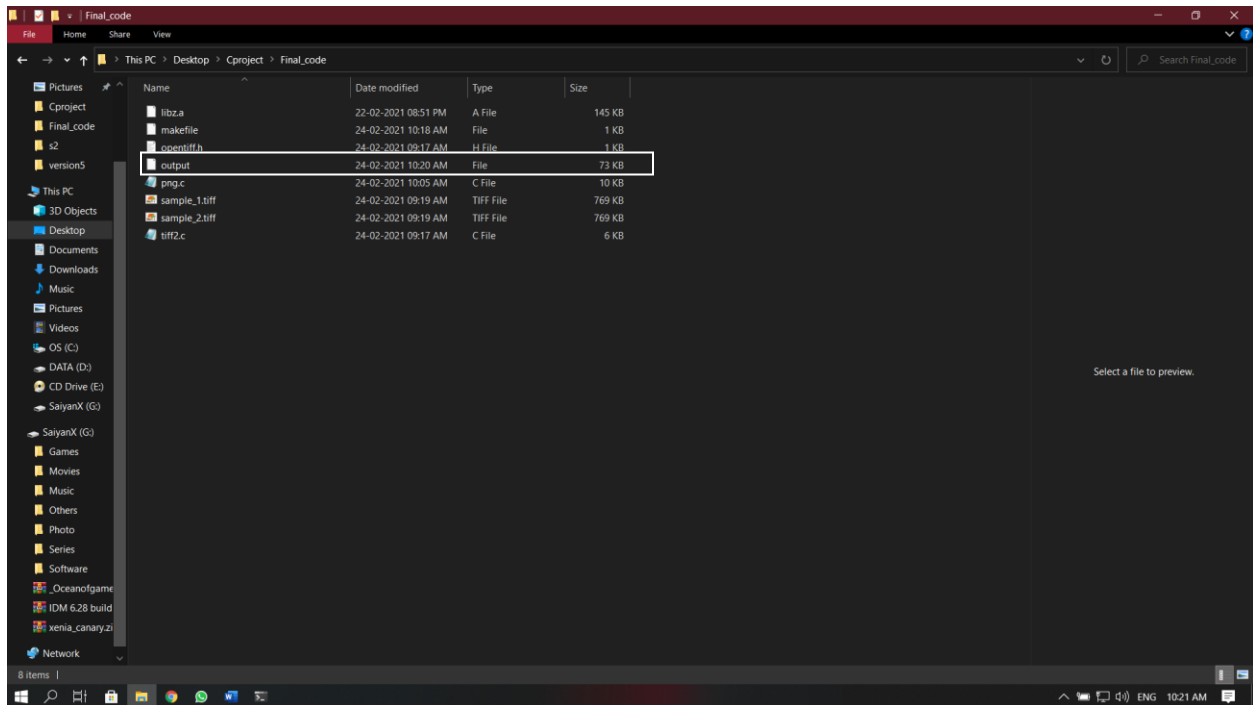
A screenshot of a Windows terminal window. The title bar shows the file name 'mnc@LAPTOP-P9LG9KF0: mnt/'. The terminal text shows the user running 'make clean' in the directory '/mnt/c/Users/ASUS/Desktop/Cproject/Final\_code'. The output shows two lines: 'rm -rf \*o output' and 'rm -rf \*o TargetImage.png'. The prompt then returns to the user's shell.

```
mnc@LAPTOP-P9LG9KF0:/mnt/c/Users/ASUS/Desktop/Cproject/Final_code$ make clean
rm -rf *o output
rm -rf *o TargetImage.png
mnc@LAPTOP-P9LG9KF0:/mnt/c/Users/ASUS/Desktop/Cproject/Final_code$
```

5. Type make

This would build an executable called output.

```
mnc@LAPTOP-P9LG9KF0: /mnt/ ...$ make clean
rm -rf *o output
rm -rf *o TargetImage.png
mnc@LAPTOP-P9LG9KF0: /mnt/...$ make
gcc -o output tiff2.c png.c libz.a -lm
mnc@LAPTOP-P9LG9KF0: /mnt/...$ |
```



6.Type in ./output “filename.tiff”

If the commands are correct the program will give an output  
TargetImage.png

