Homework 01 Instructions

Deadline: 13th April 2021 - 23:59:59

Bugs and Fixes

In this lab, your task is to review the code, find bugs with triggers for them, and write the fixes for these bugs. You will get 10 points per bug report for a new type of bug. Bugs from the same family as previous ones will grant you a maximum of 7 points per bug report. There is at least 10 bugs but less than 15.

Finding Bugs

1. Go through the code and identify locations where faulty behavior can occur.
2. For each identified location, create a test case that triggers the bug.
3. File a bug report for each bug, following the sample format below.

Only bugs in the following files will count towards the grade in this lab:

- `checkerboard.c`
- `circle.c`
- `filter.c`
- `rect.c`
- `resize.c`
- `solid.c`

However, any bugs found in the png parsing library (`pngparser.c`) may come in useful later in the course. Example bugs will not be graded, but students are encouraged to fix them on their own.

Make sure all source files are formatted using `clang-format` and LLVM style before submission. `clang-format` can be obtained on Ubuntu by installing the package `clang-format`.

The code that isn't linted properly and isn't compilable will not be graded

Fixing Bugs

For each discovered bug, identify the culprit (e.g., uninitialized variable), and replace it with code that produces results which match expected behavior. Expected behavior can be inferred from the description of different functions and tools. Exercise common sense and write the simplest fix you can think of for each bug.

Your fixes will be verified for functionality.

Sample Bug Report

Name

Uninitialized local variables

Description

The loop iteration counters, `i` and `j`, are not initialized and the behavior of the loop is thus undefined.

Affected Lines

In `filter.c:20` and `filter.c:21`

Expected vs Observed

We expect that the loops process over all the pixels in the image by iterating over every row, and every pixel in that row, starting from index 0.The loop counters are not initialized and are thus not guaranteed to start at 0. This makes the behavior of the grayscale filter undefined.

Steps to Reproduce

Command

```
./filter poc.png out.png grayscale
```

Proof-of-Concept Input (if needed)

(attached: poc.png)

Suggested Fix Description

Initialize the `i` and `j` counters to 0 in the loop setup. This allows the loop to iterate over all the image pixels to apply the grayscale filter.

Deliverables

The code needed for this assignment is provided as hw1.zip with a single top-level directory, `src/`. 1. Extract hw1.zip 2. Apply your fixes directly in `src/`. 3. Add a new sibling directory, `reports/` 4. For every discovered bug, save a file, named `bug_X.txt`, where X is a bug identifier of your choosing (e.g., decimal numbers starting 0). 5. Create a new zip file with the two top-level directories, `src/` and `reports/` and submit your .zip file on blackboard. Please make your file name as ”StudentID_YourName.zip’.

PNG Parser Interface

PNG Parser interface consists of functions:

- `int load_png(char *filename, struct image ** img)` loads a png file denoted by the filename and writes a pointer to struct image into the memory pointed to by img. Please remember to free both img and img->px after you are finished using the image. Also remember to do it in the correct order. This function returns 0 on success and a non-zero value on failure.
- `int store_png(char *filename, struct image *img, struct pixel *palette, unsigned palette_length)` which stores an image into the file. If the palette argument is NULL, the image is stored in the RGBA format. Otherwise, the palette is used to index colors.
- `struct image` stores the height and the width of an image, as well as a flattened array of pixels.

```
struct image {
    uint16_t size_x;
    uint16_t size_y;
    struct pixel *px;
};
```

- `struct pixel` represents a pixel as a RGBA tuple.

```
struct pixel {
    uint8_t red;
    uint8_t green;
    uint8_t blue;
    uint8_t alpha;
};
```

Utility Programs

Description

Several programs have been implemented to showcase capabilities of the Y0L0 PNG format:

- `checkerboard`
- `circle`
- `filter`
- `rect`
- `resize`
- `solid`

Solid

Usage: `solid output_file height width hex_color`

Output: A PNG file consisting of a solid block of color.

Checkerboard

Usage: `checkerboard output_file height width square_width hex_color1 hex_color2`

Output: A PNG file consisting of squares of alternating colors.

Filter

Usage: `filter input_file output_file filter_name [filter_arg]`

Output: Processes the input image using the specified filter.

Grayscale

Usage: `filter input_file output_file grayscale`

Output: The grayscale version of the input image.

Negative

Usage: `filter input_file output_file negative`

Output: The negative version of the input image.

Blur

Usage: `filter input_file output_file blur blur_radius`

Output: The blurred version of the image where every pixel is replaced by an average value of its neighborhood.

Alpha

Usage: `filter input_file output_file alpha hex_alpha`

Output: Sets the alpha channel of every pixel to the value of `hex_alpha`.

Circle

Usage: `circle input_file output_file center_x center_y radius hex_color`

Output: Draws a specified circular line over the input image.

Rect

Usage: `rect input_file output_file top_left_x top_left_y bottom_right_x bottom_right_y hex_color`

Output: Draws the specified rectangle over an input image.

Resize Image

Usage: `resize input_file output_file float_factor`

Output: Resizes the input by the specified factor.

Building

PNG Parser library can be built by running `make libpngparser`. It depends on zlib for compressions and on `build-utils` for building. On Ubuntu zlib can be obtained by installing the `zlib1g-dev` and `build-utils` packages.

Utility programs can be compiled by running `make program_name` or `make all`.

Y0L0 PNG Format

Y0L0 PNG format is a subset of the PNG file format. It consists of a PNG file signature, followed by mandatory PNG chunks:

- IHDR chunk
- Optional PLTE chunk
- One or more IDAT chunks
- IEND chunk

All multibyte data chunk fields are stored in the big-endian order (e.g. 4-byte integers, CRC checksums). IHDR must follow the file signature. If the palette is used to denote color, PLTE chunk must appear before IDAT chunks. IDAT chunks must appear in an uninterrupted sequence. The image data is the concatenation of the data stored in all IDAT chunks. IEND must be the last chunk in an image. All other chunk types are simply ignored.

Y0L0 PNG File Signature

Y0L0 PNG files start with the byte sequence: `137 80 78 71 13 10 26 10`

Y0L0 PNG Chunks

Y0L0 PNG chunks have the following structure:

- Length (4 bytes) denotes the length of the data stored in the chunk
- Chunk type (4 bytes) identifies the chunk type (IHDR, PLTE, IDAT or IEND). The type is encoded as a 4 byte sequence.

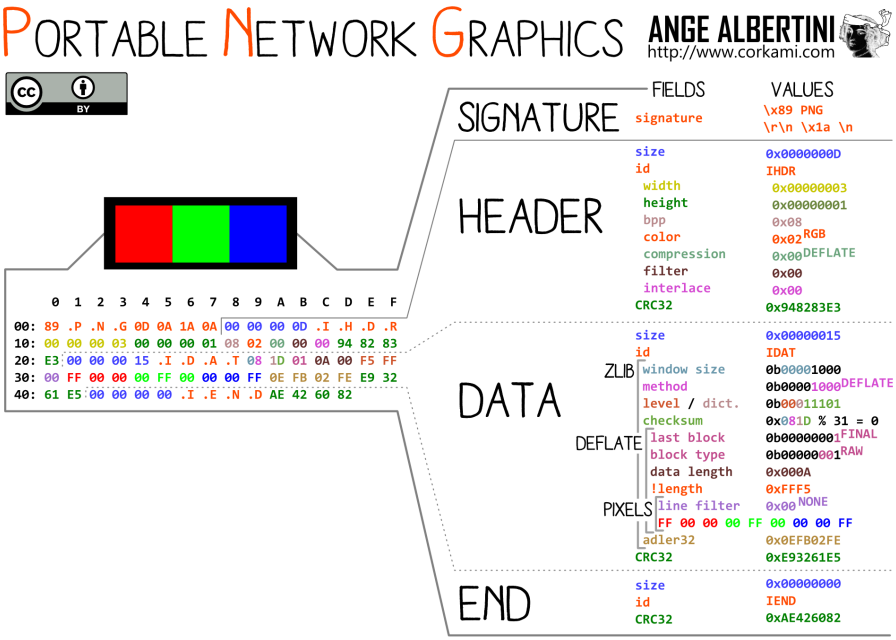

Figure 1: PNG format explanation

- Chunk data (Length bytes) stores the actual chunk data
- CRC code (4 bytes) is a checkcode that is calculated over chunk type and chunk data

All fields are consecutive and in the given order.

IHDR Chunk

IHDR chunk must appear as the first chunk following the file signature. It has the type `IHDR` and the following structure of the chunk data:

- Width (4 bytes)
- Height (4 bytes)
- Bit depth (1 byte)
- Color type (1 byte)
- Compression method (1 byte)
- Reserved (2 bytes)

All fields are consecutive and in the given order.

Bit Depth

The only supported bit-depth is 8 bits (1 byte). This refers to 1 byte per color

channel in the RGBA color mode, and to 1 byte per palette index in PLTE color mode.

Color Type

Color type field denotes the way color data is stored in the image. The only supported values are 3 (palette) and 6 (RGBA).

- Palette denotes that we expect to find a PLTE chunk in the image. In the IDAT chunk data colors are not represented as RGBA tuples, but as indices in the palette table. Every offset has the length of bit-depth. If the pixel has the color of 0x123456, and the palette has the color {R:0x12, G:0x34, B: 0x56} at the position 5, the value 5 will be stored in the image.
- RGBA mode represents colors in IDAT data as a sequence of 4 values per pixel, each one bit-depth in size. Every value corresponds to the intensity of a RGBA (red-green-blue-alpha) channel.

Compression Method

The only supported compression method is the deflate algorithm signified by value 0.

Reserved

Reserved for future use.

PLTE Chunk

PLTE chunk must appear before the IDAT chunk if the palette is used to encode color. Its type is encoded with `PLTE`. The chunk data is an array of PLTE entries which are defined as:

- Red (1 byte)
- Green (1 byte)
- Blue (1 byte)

The length field of the PLTE chunk needs to be divisible by 3.

IDAT Chunk

The type of the IDAT chunk is encoded by the bytes `IDAT`. If multiple IDAT chunks exist, they must all occur in sequence. The image data is the concatenation of the data stored in all the IDAT chunks in the order in which they appeared. It is compressed using the deflate algorithm and is stored in the zlib file format:

- Compression type and flags (1 byte)
- Flags and check bits (1 byte)
- Compressed data (n bytes)
- Adler-32 checksum (4 bytes)

This data is used as an input for the inflate algorithm and is handled by zlib. Inflated data is stored left-to-right, top-to-bottom. Every row (scanline) begins with a byte denoting the start of the line. This byte must be 0. Pixel information for the scanline follows this byte. It consists either of the palette index (1 byte), or of the RGBA pixel value (4 bytes), depending on the color format.

All fields of structures are consecutive and in the given order.

IEND Chunk

IEND chunk appears as the last chunk in the file. Its type is `IEND`. It stores no data and the length field is 0.