COS284 Practical Assignment 5: Working with Files and Structures in YASM Assembly Language

University of Pretoria
Department of Computer Science

Online Publication: 14 October 2024

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Plagiarism Policy

- All work submitted must be your own. You may not copy from classmates, textbooks, or other resources unless explicitly allowed.
- You may discuss the problem with classmates, but you may not write or debug code with other classmates, other than those in your group.
- If you use any external material, you must cite these sources.

Assignment Overview

In this assignment, you will demonstrate your practical knowledge of YASM assembly language, focusing on reading from and to file sas well as the use of structures (`structs`). You will complete three tasks that involve defining structs, manipulating data within them, and implementing a 2D Array linked list. Each task builds upon the previous one, enhancing your understanding of how structs operate at the assembly level.

General Data Structures

Before diving into each function, it's crucial to understand the data structures used:

- **PixelNode**: A struct representing a pixel in the image, organized in a 2D linked list to facilitate easy traversal in all directions (up, down, left, right).

```
typedef struct PixelNode {
    unsigned char Red;
    unsigned char Green;
    unsigned char Blue;
    unsigned char CdfValue;
    struct PixelNode* up;
    struct PixelNode* down;
    struct PixelNode* left;
    struct PixelNode* right;
} PixelNode;
```

- **Histogram Arrays**: Arrays used to store the frequency of grayscale intensities and their cumulative distribution.

1 Task1: `readPPM` Function

Purpose

Reads a binary P6 PPM image file and constructs a 2D linked list (`PixelNode`) representing the image pixels.
The file will look something like this:

Inputs

- `const char* filename`: Path to the PPM image file.

Outputs

- Returns a pointer to the head of the 2D linked list representing the image pixels.

Approach

1. **Open the File**:
   - Use system calls to open the file in binary read mode.
2. **Read and Parse the Header**:
   - PPM P6 files have a header with the following format:

     ```
     P6
     width height
     maxColorValue
     RGBRGBRGB...
     ```

   - The header may contain comments starting with `#`. These comments should be skipped.
   - Read the header byte by byte, checking for whitespace and newline characters (`\n`).
   - Use a buffer to store header data up to a defined maximum size (e.g., 512 bytes).
3. **Extract Image Metadata**:
   - Parse the header buffer to extract:
     - Image format (should be `P6`).
     - Image width and height.
     - Maximum color value (usually 255).
   - Handle cases where comments or additional whitespace complicate parsing.
4. **Validate Metadata**:
   - Ensure that width, height, and `maxColorValue` are positive integers.
   - If any validation fails, close the file and return NULL.
5. **Read Pixel Data**:
   - For each pixel, read three bytes corresponding to the Red, Green, and Blue components.
   - Allocate memory for a new `PixelNode`.
   - Store the RGB values in the `PixelNode`.

- Initialize the `CdfValue` to 0.
- Set up the linked list pointers (`up`, `down`, `left`, `right`) to connect each pixel appropriately:
  - Pixels in the same row are connected via `left` and `right`.
  - Pixels in the same column are connected via `up` and `down`.

6. **Construct the 2D Linked List**:
   - Use nested loops to iterate over the height (rows) and width (columns) of the image.
   - Keep track of the start of each row to connect the `up` and `down` pointers between rows.
7. **Close the File and Return**:
   - After reading all pixel data, close the file.
   - Return the head of the linked list.

2 Task2: `computeCDFValues` Function

Purpose

Computes the histogram of grayscale intensities, calculates the cumulative distribution function (CDF), and normalizes it. Updates each pixel's `CdfValue` with the normalized CDF value.

Inputs

- `PixelNode* head`: Pointer to the head of the 2D linked list representing the image.

Outputs

- Updates each `PixelNode`'s `CdfValue` with the normalized CDF value.

Approach

W

1. **Initialize Histogram and Variables**:
   - Create an array `histogram[256]` to store the frequency of each grayscale intensity (0–255).
   - Initialize all elements to zero.
2. **First Pass - Compute Histogram**:
   - Traverse the 2D linked list row by row.
   - For each pixel:

- Convert the RGB values to a grayscale intensity using the **luminosity method**:

$$\text{gray} = 0.299 \times \text{Red} + 0.587 \times \text{Green} + 0.114 \times \text{Blue}$$

- The result is a floating-point value. Convert it to a single byte value (to store a range of 0–255).
- Store the grayscale intensity in the `CdfValue` field of the `PixelNode` (temporarily).
- Increment the corresponding histogram bin:

$$\text{histogram}[\text{gray}] \mathrel{+}= 1$$

3. **Compute the Cumulative Histogram (CDF)**:

- Create an array `cumulativeHistogram[256]` to store the cumulative frequencies.
- Initialize a variable `cumulative` to zero.
- For each intensity level i from 0 to 255:
  - Update the cumulative frequency:

$$\text{cumulative} \mathrel{+}= \text{histogram}[i]$$

  - Store it in the cumulative histogram:

$$\text{cumulativeHistogram}[i] = \text{cumulative}$$

4. **Find the Minimum Non-Zero CDF Value (cdfMin)**:

- Iterate over the cumulative histogram to find the smallest non-zero cumulative frequency:

$$\text{cdfMin} = \min \{\text{cumulativeHistogram}[i] \mid \text{histogram}[i] \neq 0\}$$

5. **Second Pass - Normalize CDF and Update Pixels**:

- Traverse the 2D linked list again.
- For each pixel:
  - Retrieve the grayscale intensity stored earlier in `CdfValue`.
  - Use the normalized histogram equalization formula to compute the new intensity:

$$\text{cdfValue} = \left(\frac{\text{cumulativeHistogram}[\text{intensity}] - \text{cdfMin}}{\text{totalPixels} - \text{cdfMin}} \right) \times 255$$

  - Ensure the `cdfValue` is clamped between 0 and 255.
  - Store the normalized `cdfValue` back into the `CdfValue` field of the `PixelNode`.

Notes

- Pay attention to floating-point operations and conversions between integer and floating-point values.
- Handle division carefully to avoid division by zero (ensure `totalPixels` −cdfMin is not zero).
- Clamping ensures that the new intensity values stay within valid bounds (0–255).

3 Task3: `applyHistogramEqualisation` Function

Purpose

Applies the normalized CDF values to adjust the pixel intensities, effectively performing histogram equalization on the image.

Inputs

- `PixelNode* head`: Pointer to the head of the 2D linked list representing the image.

Outputs

- Updates each pixel's RGB values to reflect the histogram-equalized grayscale intensity.

Approach

1. **Traverse the 2D Linked List**:
   - Iterate over each pixel in the image.
2. **Update Pixel Intensities**:
   - For each pixel:
     - Retrieve the normalized `CdfValue` from the `PixelNode`.
     - Round the `CdfValue` to the nearest integer:

$$\text{newPixelValue} = \text{round}(\text{CdfValue}) = \lfloor \text{CdfValue} + 0.5 \rfloor$$

     - Clamp `newPixelValue` between 0 and 255 to ensure it's a valid intensity.
     - Set the pixel's RGB values to `newPixelValue`, converting the image to grayscale:

```
pixel->Red = newPixelValue;
pixel->Green = newPixelValue;
pixel->Blue = newPixelValue;
```

   - This step ensures that the histogram equalization effect is applied, enhancing the contrast of the image.

Notes

- Rounding can be performed by adding 0.5 to the floating-point number and then truncating it to an integer.
- Clamping after rounding ensures that any slight overflows due to rounding are corrected.

4 Task4: `writePPM` Function

Purpose

Writes the processed image data to a new PPM P6 file using the 2D linked list of `PixelNode`s.

Inputs

- `const char* filename`: Path to the output PPM file.
- `const PixelNode* head`: Pointer to the head of the 2D linked list representing the image.

Outputs

- Creates a new PPM file with the histogram-equalized image.

Approach

1. **Open the File**:
   - Use system calls to open the file in binary write mode.
   - Handle errors if the file cannot be opened.
2. **Determine Image Dimensions**:
   - Since the image dimensions are not directly stored in the `PixelNode`s, you need to calculate them:
     - **Width**: Traverse the first row (using `right` pointers) and count the number of pixels.
     - **Height**: Traverse the first column (using `down` pointers) and count the number of pixels.
3. **Write the PPM Header**:
   - Use the following format for the header:

```
P6
width height
maxColorValue
RGBRGBRGB...
```

- Typically, `maxColorValue` is 255.

4. **Write Pixel Data**:
   - Iterate over each row:
     - For each pixel in the row:
       * Write the `Red`, `Green`, and `Blue` values to the file.
       * Each component is one byte.
   - Ensure that you write the data in binary mode without any additional formatting.
5. **Close the File**:
   - After writing all pixel data, close the file.

Notes

- Be careful with file I/O operations to ensure data integrity.
- Ensure that you traverse the linked list correctly to write all pixels.

After successful writing of your converted file, the image will look something like this:

Formulas and Algorithms Summary

- **Grayscale Conversion (Luminosity Method)**:

$$\text{gray} = 0.299 \times \text{Red} + 0.587 \times \text{Green} + 0.114 \times \text{Blue}$$

- **Histogram Equalization Formula**:

$$\text{cdfValue} = \left(\frac{\text{cumulativeHistogram[intensity]} - \text{cdfMin}}{\text{totalPixels} - \text{cdfMin}} \right) \times 255$$

- **Rounding to Nearest Integer**:

$$\text{newPixelValue} = \text{round}(\text{cdfValue}) = \lfloor \text{CdfValue} + 0.5 \rfloor$$

Implementation Tips

- **Assembly Language Considerations**:
  - Pay attention to calling conventions when interfacing with the C `main` function.
  - Use appropriate registers for passing arguments and returning values.
  - Manage the stack carefully to preserve return addresses and local variables.
  - Allocate memory using system calls or C library functions (e.g., `malloc`), ensuring you handle pointers correctly.
- **Error Handling**:
  - Return appropriate error codes or `NULL` pointers when operations fail.
- **Testing**:
  - Use small PPM images for initial testing to simplify debugging.
  - Compare your output with the expected result to verify correctness.

Expected Outcome

By completing this assignment, you will have:

- Understood how to perform file I/O operations, memory management, and data structure manipulation in assembly.
- Learned how image processing techniques like histogram equalization are implemented at a low level.
- Improved your problem-solving skills by working with pointers, linked lists, and numerical algorithms in assembly.

Mark Distribution

The total marks for this assignment are 20, distributed as follows:

- **Task 1: `readPPM` Function** (10 marks)
  - Correctly opening and reading the PPM file header: **2 marks**
  - Proper parsing of image metadata (width, height, maxColorValue): **2 marks**
  - Handling comments and whitespace in the PPM header: **1 mark**
  - Correct construction of the 2D linked list of `PixelNode`s: **5 marks**
- **Task 2: `computeCDFValues` Function** (6 marks)
  - Accurate computation of the grayscale histogram: **2 marks**
  - Correct calculation of the cumulative distribution function (CDF): **2 marks**
  - Proper normalization and updating of `CdfValue`s in `PixelNode`s: **2 marks**
- **Task 3: `applyHistogramEqualisation` Function** (4 marks)
  - Correct traversal of the 2D linked list: **2 mark**
  - Accurate application of normalized CDF values to update pixel intensities: **2 marks**
- **Task 4: `writePPM` Function** (5 marks)
  - Correct determination of image dimensions (width and height): **2 mark**
  - Proper writing of the PPM header and pixel data to the output file: **3 marks**

Submission

- Submit your assembly source code files for each task:
  - `task1_read_ppm_file.asm`
  - `task2_compute_cdf_values.asm`
  - `task3_histogram_equalisation.asm`
  - `task4_write_ppm.asm`

Assessment Criteria

- **Correctness:** The program functions as specified in each task.
- **File I/O:** Proper reading from and writing to files.
- **Use of Structs:** Proper definition and manipulation of structs.

Additional Resources

- You can view PPM files online at: Netpbm Viewer
- YASM Documentation: https://yasm.tortall.net/
- YASM tutorial playlist: Playlist

Academic Integrity

Remember to adhere to the university's policies on academic integrity. Any form of plagiarism or academic dishonesty will be dealt with according to university regulations.