

CS 218 – Assignment #11, Part A

Purpose: Become more familiar with operating system interaction, file input/output operations, and file I/O buffering.

Points: 300 (grading will include functionality, documentation, and coding style)

Assignment:

Write a program that will create a thumbnail¹ (small, 300x200 image). The program will read the source file and output file

names from the command line. The program must perform basic error checking on the command line file names. For example, given an image file, **image.bmp**, to create a thumbnail image named **thumb.bmp**, the command line might be as follows:

```
./makeThumb image.bmp thumb.bmp
```

The required functions include the following:

- Function **getImageFileNames()** to read and verify the file names from the command line. The function should check the file names by attempting to open the files. If both files open, the function should return TRUE and the file descriptors. The function should verify the ".bmp" extension. If there is an error, the function should display an appropriate error message and return FALSE.
- Function **setImageInfo()** to read (original image file), verify, and write (thumbnail file) the header information. The function must check the signature, color depth, and bitmap size consistency. Refer to the BMP File Format section. If the original image file information is correct, the header should be updated to reflect the height and width (passed) and the new smaller thumbnail file size ($width * height * 3 + \text{HEADER_SIZE}$). Once updated, the revised header can be written to the output file. Additionally, the original width and original height should be returned to the main (via reference). If all parameters were correct, the function should return TRUE. If there is an error, the function should display an appropriate error message (predefined) and return FALSE.
- Function **readRow()** should return one row of pixels. If a row can be returned, the function should return the row and return TRUE. If a row can not be returned (because there is no more data) the function should return FALSE. If there is a read error, the function should display an appropriate error message (predefined) and return FALSE. To ensure overall efficiency, the function **must** perform buffered input with a buffer size of **BUFF_SIZE** (originally set to 750,000). *Note*, this will be changed for part B.



¹ For more information, refer to: <https://en.wikipedia.org/wiki/Thumbnail>

- Void function ***writeRow()*** should write one row of pixels to the output file. If successful, the function should return TRUE. If there is a write error, the function should display an appropriate error message (predefined) and return FALSE. This function is not required to buffer the output. As such, each row may be written directly to the output file.

Provided Main

The provided main program calls the various functions. ***The provided main program must not be changed in any way.*** All your functions must be in a separate, independently assembled source file. *Note*, to help with the initial program testing, it might be best to temporarily skip (comment out) the calls to the image manipulation statements (in the main).

Assemble and Linking Instructions

You will be provided a main function (**`makeThumb.cpp`**) that calls the functions. Your functions should be in a separate file (**`allprocs.asm`**). The files will be assembled individually and linked together.

When assembling, and linking the files for assignment #10, use the provided **makefile** to assemble, and link. *Note*, **only** the functions file, **`allprocs.asm`**, will be submitted. The submitted functions file will be assembled and linked with the provided main. As such, do not alter the provided main.

Debugging -> Command Line Arguments

When debugging a program that uses command line arguments, the command line arguments must be entered after the debugger has been started. The debugger is started normally (**`ddd <program>`**) and once the debugger comes up, the initial breakpoint can be set. Then, when you are ready to run the program, enter the command line arguments. This can be done either from the menu (Program -> Run) or on the GDB Console Window (at bottom) by typing **`run <commandLineArguments>`** at the (gdb) prompt (bottom window).

Buffering

Since many image files can be very large, it would be difficult to pre-allocate enough memory to hold a complete large image. Further, that memory would be mostly unused for processing smaller images. To address this, the program will perform *buffered input*. Specifically, the program should read a full buffer of **`BUFF_SIZE`** bytes (originally set to 750,000). *Note*, this will be changed for part B.

From that large buffer, the ***readRow()*** function would return one row, ***width* × 3** bytes. The next call to the ***readRow()*** function would return the next row. As such, the ***readRow()*** function must keep track of where it is in the buffer. When the buffer is depleted, the ***readRow()*** function must re-fill the buffer by reading **`BUFF_SIZE`** bytes from the file. Only *after* the last row has been returned, should the ***readRow()*** function return a FALSE status.

24-bit Color Depth

For 24-bit color depth, the pixel color is stored as three bytes, a red value (0-255), a green value (0-255), and a blue value (0-255). The color values are stored in that order. Thus, each pixel is 3 bytes (or 24-bits). So, for a 800 pixel row, there are 2,400 bytes (800 * 3).

The main sets a row value maximum as 5,000 pixels (or 15,000 bytes). The provided main will perform this verification and display an appropriate error message as required.

BMP File Format²

The BMP format supports a wide range of different types, including possible compression and 16, 24, and 32 bit color depths. For our purposes, we will only support uncompressed, 24-bit color BMP files. Under these restrictions, the header (or initial information) in the file is a 138 byte block containing a series of data items as follows:

Size	Description
2 bytes	Signature. Must be "BM". <i>Note</i> , must ensure is "BM" for this assignment.
4 bytes	File size (in bytes).
4 bytes	Reserved.
4 bytes	Size of header. Varies based on compression and color depth. For an uncompressed, 24-bit color depth, the header is 54 bytes.
4 bytes	Offset to start of image data in bytes. May vary based on compression and color depth.
4 bytes	Image width (in pixels).
4 bytes	Image height (in pixels).
2 bytes	Number of planes in image (typically 1).
2 bytes	Number of bits per pixel. Typically 16, 24, or 32. <i>Note</i> , must ensure is 24 for this assignment.
4 bytes	Compression Type (0=none). <i>Note</i> , must ensure is 0 for this assignment.
4 bytes	Size of image in bytes (may vary based on compression types).
100 bytes	Miscellaneous (not used for uncompressed, 24-bit color depth).

Since the remaining parts of the program will only work for uncompressed, 24-bit color depth, these must be verified before the program can perform image manipulations. Specifically, the following items should be verified:

- File signature is valid (must be "BM" for signature).
- Color depth is 24-bits.
- Image data is not compressed (i.e., compression type must be 0).
- File bitmap block size consistency (file size = size of image in bytes + header size).

Appropriate error message strings are provided in the functions template. Once these are verified, the header information should be written to the output file.

Example Executions:

The following execution, which includes some errors, will read file **image0.bmp**, create the thumbnail image, and place the output image in a file named **tmp.bmp**.

```
ed-vm% ./makeThumb
Usage: ./makeThumb <inputFile.bmp> <outputFile.bmp>
ed-vm%
ed-vm% ./makeThumb img0.bp thm.bmp
Error, invalid source file name. Must be '.bmp' file.
ed-vm% ./makeThumb none .bmp tmp.thm
Error, too many command line arguments.
```

² For more information, refer to: http://en.wikipedia.org/wiki/BMP_file_format

```



ed-vm%
ed-vm% ./makeThumb none.bmp none. bmp
Error, too many command line arguments.
ed-vm%
ed-vm% ./makeThumb img1.bmp thm.bmpp
Error, invalid output file name. Must be '.bmp' file.
ed-vm%
ed-vm% ./makeThumb tmp.bmp
Error, incomplete command line arguments.
ed-vm%
ed-vm% ./makeThumb img4.bmp tmpImg4.bmp
ed-vm%
ed-vm% ./makeThumb test/imgBad1.bmp tmp.bmp
Error, unsupported color depth. Must be 24-bit color.
ed-vm%
ed-vm% ./makeThumb test/imgBad2.bmp tmp.bmp
Error, only non-compressed images are supported.
ed-vm%
ed-vm% ./makeThumb test/imgBad3.bmp tmp.bmp
Error, bitmap block size inconsistent.
ed-vm%
ed-vm% ./makeThumb test/imgBad0.bmp tmp.bmp
Error, invalid file signature.
ed-vm%

```

Note, a series of image files, including **img4.bmp**, are provided. However, the program will work on any uncompressed, 24-bit BMP format file.

Example Ouptut Images

The following table provides some examples of the expected final output.

Example Output	
Original Image img4.bmp Original Image Size: 5.8 MB	Image thumbnail tmpImg4.bmp Original Image Size: 180.1 KB
	

Submission:

- All source files must assemble and execute on Ubuntu with **yasm**.
- Submit source files
 - Submit a copy of the program source file via the on-line submission.
 - Note, only the functions file (**allprocs.asm**) will be submitted.
- Once you submit, the system will score the project and provide feedback.
 - If you do not get full score, you can (and should) correct and resubmit.
 - You can re-submit an unlimited number of times before the due date/time (at a maximum rate of 5 submissions per hour).
- Late submissions will be accepted for a period of 24 hours after the due date/time for any given assignment. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, ... , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0.

Program Header Block

All source files must include your name, section number, assignment, NSHE number, and program description. The required format is as follows:

```
; Name: <your name>
; NSHE ID: <your id>
; Section: <section>
; Assignment: <assignment number>
; Description: <short description of program goes here>
```

Failure to include your name in this format will result in a loss of up to 3%.

Scoring Rubric

Scoring will include functionality, code quality, and documentation. Below is a summary of the scoring rubric for this assignment.

Criteria	Weight	Summary
Assemble	-	Failure to assemble will result in a score of 0.
Program Header	3%	Must include header block in the required format (see above).
General Comments	7%	Must include an appropriate level of program documentation.
Program Functionality (and on-time)	90%	Program must meet the functional requirements as outlined in the assignment. Must be submitted on time for full score.