Derek Sims and Dylan Park's Design Document

Modules:

- 40image.c
  - Contains the main function
  - Using compress40.h, either calls compress40 or decompress40 with the given image file pointer depending on user input
- compress40
  - `void compress40(FILE *fp)`
    - Writes compressed 32 bit codeword of given image to standard output in big-endian order
  - `void decompress40(FILE *fp)`
    - Writes decompressed image to standard output (using Pnm_ppmwrite)
- Process_image
  - `UArray2 trim_dim(Pnm_ppm ppm_struct)`
    - Trims the height and width of the UArray2 to even dimensions and returns trimmed UArray2 if necessary
  - `void read_header(char *header, int *width, int *height)`
    - Reads the header of compressed file with fscanf and ensures it is in right format, gets height and width
  - `void print_codewords(uint64_t word)`
    - Prints each 32-bit code word written in big-endian order
- Rgb_vid_conversion
  - ```
    typedef struct {
            float y;
            float pb;
            float pr;
    } *vid_color;
    ```
  - `UArray2b_T rgb_to_vid(Pnm_ppm ppm_struct)`
    - Given the ppm_struct which has a UArray2 of Pnm_rgb structs representing pixels in RGB color space, convert each pixel to video color space by creating our own struct with the Y, Pr, and Pb values and placing these structs into a new UArray2b_T
    - Blocksize will be 2 so that the cells of each 2x2 block (to be accessed later) are stored close together in memory
    - Returns new UArray2b_T
  - `UArray2 vid_to_rgb(UArray2b uarray2b)`
    - Transforms pixels from a component video color to RGB color, quantizes the RGB values to integers in range of 0-255, and puts RGB values into array in pixmap struct
- Pack_or_unpack
  - `struct coded_elements`
    - Contains `uint64_t a, int64_t b, int64_t c, int64_t d, float index_pb, float index_pr`
  - `uint64_t pack_block_at(UArray2b uarray2b, int i , int j)`
    - Packs the pixels from the block of the UArray2b (with its top-left element at (i, j)) into a 64 bit word, returns that word
  - `void avg_chroma(UArray2b uarray2b, int i, int j, float *pb_avg, float *pr_avg)`
    - Calculates and updates the avg_pb and avg_pr of the block starting at the given index
    ```

    - `coded_elements dct(UArray2b uarray2b, int i, int j)`
        - Calculates the values for a, b, c, d and index(Pr) and index(Pb) at the given index, returns a pointer to a struct containing that info
    - `int64_t code_float(float f)`
        - Codes given float into a five bit signed int, returns that int
    - `Float uncode_int(int64_t i)`
        - Uncodes given int into a float, returns that float
    - `coded_elements unpack_values(uint64_t word)`
        - Unpacks the values for a, b, c, d (using Bitpack_get) and coded Pb and Pr into local variables
        - Converts the chroma codes for Pb and Pr into the avg values for Pb and Pr
    - `void inverse_dct(UArray2b uarray2b, coded_elements)`
        - Calculates Y1, Y2, Y3, and Y4 from the elements in the coded_elements struct
        - Creates a new struct containing Y, Pb, and Pr for each pixel and places it into uarray2b
- Bitpack
    - `bool Bitpack_fitsu(uint64_t n, unsigned width)`
    - `bool Bitpack_fitss(int64_t n, unsigned width)`
        - Returns true if n can be represented in width bits, false if it cannot
    - `uint64_t Bitpack_getu(uint64_t word, unsigned width, unsigned lsb)`
    - `int64_t Bitpack_getss(uint64_t word, unsigned width, unsigned lsb)`
        - Extracts a field from a word given the width of the field and location of the field's least significant bit
        - Checked run-time error if called with a width less than 0 or greater than 64 or if w + lsb is not less than or equal to 64
    - `uint64_t Bitpack_newu(uint64_t word, unsigned width, unsigned lsb, uint64_t value)`
    - `uint64_t Bitpack_news(uint64_t word, unsigned width, unsigned lsb, uint64_t value)`
        - Returns a new word identical to original word, except field of width `width` has been replaced by a `width`-bit representation of `value`
        - Checked run-time error if called with a width less than 0 or greater than 64 or if w + lsb is not less than or equal to 64
        - Raises exception if `value` does not fit in `width` signed bits

Interactions:

- 40image.c will call either compress40 or decompress40 within compress40, depending on user input
- Compress40
    - Uses functions within Rgb_vid_conversion file to convert between arrays of RGB pixels and arrays of video color space pixels
    - Uses functions within pack_or_unpack file to perform its calculations which are completed by calling its own local functions
- Pack_or_unpack
    - Calls functions within Bitpack file to return either the code word

Testing:

- 40image.c
    - Pass in normal PPM image
        - Should open it with no exceptions raised

    - Pass in non-PPM image
        - Should raise exception
- Compress40
    - `UArray2 trim_dim(UArray2 uarray2)`
        - Pass image with odd dimensions
            - Should trim edges making height and width even
        - Pass image with even dimensions
            - Trim_dim should not be called
    - `void read_header(char *header, int *width, int *height)`
        - Pass in correct test compressed file
            - Should not raise exception for incorrect header format
        - Pass in incorrect compressed file (extraneous information or not enough information in header)
            - Should raise exception
    - Pass in sample code words to print
        - Should print out in big-endian order
- Rgb_vid_conversion
    - `UArray2b rgb_to_vid(UArray2 uarray2)`
        - Pass in correct UArray2 with struct containing Pnm_rgb structs
            - Should return UArray2b with struct filled with correct Y, Pb, and Pr values
        - Pass in incorrect UArray2 (does not contain correct struct)
            - Should raise exception
    - `UArray2 vid_to_rgb(UArray2b uarray2b)`
        - Pass in correct UArray2b with struct containing Y, Pb, and Pr
            - Should return UArray2 with correct RGB values
        - Pass in incorrect UArray2b
            - Should raise exception
- Pack_or_unpack
    - `uint64_t pack_block_at(UArray2b uarray2b, int i , int j)`
        - Pass in correct UArray2b
            - Should calculate avg_pb and avg_pr of the block starting at the given index
            - Should calculate a, b, c, and d correctly
            - Should return correct code word
        - Trying to access something out of UArray2b
            - Should raise exception for incorrect retrieval request
    - `void unpack_values(uint64_t word, uint64_t *a, int64_t *b, int64_t *c, int64_t *d, float *pb_avg, float *pr_avg)`
        - Pass in accurate code word
            - Should successfully retrieve a, b, c, d, Pb_avg, and Pr_avg
- Bitpack
    - `bool Bitpack_fitsu(uint64_t n, unsigned width)`
    - `bool Bitpack_fitss(int64_t n, unsigned width)`
        - Ensure that difference between unsigned and signed is recognized
            - Ex/ Bitpack_fitsu(4, 3) == true, but Bitpack_fitss(4, 3) == false
    - `uint64_t Bitpack_getu(uint64_t word, unsigned width, unsigned lsb)`
    - `int64_t Bitpack_getss(uint64_t word, unsigned width, unsigned lsb)`
        - Call with width less than 0 or greater than 64
            - Raises exception

        - Call with w + lsb being greater than 64
            - Raises exception
        - Correct word, width, and lsb
            - Should accurately return field from word
    - `uint64_t Bitpack_newu(uint64_t word, unsigned width, unsigned lsb, uint64_t value)`
    - `uint64_t Bitpack_news(uint64_t word, unsigned width, unsigned lsb, uint64_t value)`
        - Call with width less than 0 or greater than 64
            - Raises exception
        - Call with w + lsb being greater than 64
            - Raises exception
        - Call with correct arguments
            - Returns a word with field of width `width` having been replaced by a `width`-bit representation of `value`

How will design enable us to do well on the challenge problem?

- Due to the fact that we have pretty good modularity and we break up our code into small steps, it should be relatively easy to update our code to account for the change to the code word.
- Additionally, due to the fact that we have a struct containing the values a, b, c, d, index of pr, and index of pb should make it easy to alter should the challenge problem change the values placed into the code word or the manner in which you calculate these values.

Loss of Information

- After the first compression/decompression, we lose the Pb and Pr values of each of the pixels and are left with just the average Pb and Pr for each 2x2 block. If we were to compress/decompress again, we would not lose any more information because the averages would not change.
- Additionally, through quantization, the b/c/d values will lose information on its first compression/decompression to encode and decode these values. In additional compressions/decompressions, however, these values do not change again because they will code/decode to the same values.