HW4 DESIGN DOC
Feiyu Lu & Xu Liu on Feb 20, 2015

## 1 Overall Architecture:

Part A: Compress/decompress:

### a. type:

```
 Pnm_ppm; //store rgb pixel.
 Uarray2; //store decompressed result;

 extern A2Methods_T uarray2_methods_blocked;// read in original image for compressor;
/* following three type is for store intermediate results during compressing and decompressing;
     typedef struct Cie {
            float Y, Pr, Pb;
      } *Cie;
      typedef struct float_pixel {
            float a, b, c, d, Pr_ave, Pb_ave;
      } *float_pixel;
      typedef struct int_pixel {
            uint64_t a, Pr_ave, Pb_ave;
            int64_t b, c, d;
      } *int_pixel;
```

### b. functions:

```
  /*read in data inside input file, trim the matrix to even row and column then call pack_print to compress the file*/
 compress40(FILE *input)
        /* read in original data to compress*/
          Pnm_ppmread()
        / *compress original data and print out the compressed data */
        void pack_print(Pnm_ppm original);
                    /* following functions are called to help converting, compressing ,packing and printing */
                    /* convert rgb pixel to Y/P/P pixel */
                   void rgb_to_cie(Pnm_rgb src, Cie des, int dnm);
                    /*compress four pixels to one pixel*/
                   void pixels_ave(Cie src, float_pixel des);
                   /*quantize each component of  compressed pixel from float type to int type*/
                   void quantize(float_pixel src, int_pixel des);
                   /* pack each component of one pixel into 32 bit space*/
                   uint64_t pack_into_words(int_pixel src);
                             /* following two functions will be called to pack the pixel data into 32 bit
                                                                                              space */
                             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
                                                         ,  int64_t value)

                   /* print out the 32 bit number in big endian order*/
                   void print_result(uint64_t *result);
                   /* helper function for quantization*/
                   static inline int quantize_a(float a);
                   static inline int quantize_bcd(float a);


         /*read in data inside input file, then decompress the file*/
         decompress40(FILE *input)
                    /* following functions will be called to read in data, extract pixel information and
                                                         convert them into rgb data, finally write into to
         file*/
                    /*read in original data to decompress*/
                    fscanf()
                    /* extract the pixel information from a 32 bit number*/
                   uint64_t extract_from_words(uint64_t words, int_pixel des);
                   /*convert the signed or unsigned data to float number by reversing map against
                     the compressing procedure */
                   void int_to_float(int_pixel src, float_pixel des);
                   /* expand one pixel to a block of 4 pixel;*/
                   void pixel_to_block(float_pixel src, float_pixel des);
                   /*convert cie pixel to rgb pixel*/
                   void cie_to_rgb(float_pixel src, Pnm_ppm des);
                    /*print out*/
                    Pnm_ppmwrite();
```

Part B: Bitpack:

- **type:**

```
/* for raising exception of bit overflow*/
Except_T Bitpack_Overflow = { "Overflow packing bits"};
```

- **function:**

```
/* shift left operation*/
uint64_t l_shift(uint64_t words, int n)
```

```
 /*shift right on signed number*/
int64_t r_shifts(int64_t words, int n)
 /*shift right on unsigned number */
uint64_t r_shiftu(uint64_t words, int n)

 /*check if a number could fit in given space*/
 bool Bitpack_fitsu(uint64_t n, unsigned width)
 bool Bitpack_fitss(int64_t n, unsigned width)
 /*extract a field from given space */
uint64_t Bitpack_getu(uint64_t word, unsigned width, unsigned lsb)
 int64_t Bitpack_gets(uint64_t word, unsigned width, unsigned lsb)
 /* put a given field into a given 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
                                       ,  int64_t value)
```

## 2 Architecture Sections:

```
  /*compress40() is reposible for making sure we correctly read in original data*/
 compress40(FILE *input)
         /* read in original data to compress*/
           Pnm_ppmread()

         / *pack_print iterate each block of the orignial image and apply the following functions
             on each block;
                    convert rgb to Y/P/P  by  rgb_to_cie(Pnm_rgb src, Cie des, int dnm)
                    compress a block into  a pixel by pixels_ave();
                    quantize each component of the pixel by quantize();
                    pack a pixel into a 32 bit spacef by pack_into_words();
                    print out the word by print_result();
               */
         void pack_print(Pnm_ppm original);

                     /* convert rgb pixel to Y/P/P pixel according to given converting matrix from
                                                                                  assignment
*/
                    void rgb_to_cie(Pnm_rgb src, Cie des, int dnm);
                     /*compress four pixels to one pixel according to relationship given in the
                                                                                 assignment*/
                    void pixels_ave(Cie src, float_pixel des);
                    /*quantize each component of  compressed pixel from float type to int type
```

```
                                                  according to given relationship in the assignment*/
                void quantize(float_pixel src, int_pixel des);




                /* call interface in bitpack.h to pack each component of one pixel into 32 bit

        space*/
                uint64_t pack_into_words(int_pixel src);
                /* print out the 32 bit number in big endian order*/
                void print_result(uint64_t *result);
                /* helper function for quantization*/
                static inline int quantize_a(float a);
                static inline int quantize_bcd(float a);


/*decompress40 executes an inverse procedure from compress40
  it organizes the following functions*/
decompress40(FILE *input)
                 /*read in original data to decompress*/
                 fscanf()
                 /* call interface in bitpack.h to extract the pixel information from a 32 bit
number*/
                uint64_t extract_from_words(uint64_t words, int_pixel des);
                /*convert the signed or unsigned data to float number according to the
        relationship
                                                                                  given in the
        assignment*/
                void int_to_float(int_pixel src, float_pixel des);
                /* expand one pixel to a block of 4 pixel according to the relationship given in
        the

        assignment*/
                void pixel_to_block(float_pixel src, float_pixel des);
                /*convert cie pixel to rgb pixel according given transferring matrix*/
                void cie_to_rgb(float_pixel src, Pnm_ppm des);
                 /*print out*/
                 Pnm_ppmwrite();
```

**3 Test:**

**a. test bitpack**

-------------------------------------------

For functions:

```
uint64_t l_shift(uint64_t words, int n)
int64_t r_shifts(int64_t words, int n)
uint64_t r_shiftu(uint64_t words, int n)
```

- **test1:** shift a 64 bit number by random number of bits, to see if the result is right;

--------------------------------------------

For functions:

```
bool Bitpack_fitsu(uint64_t n, unsigned width)
bool Bitpack_fitss(int64_t n, unsigned width)
```

- **test2:** test if we can get the valid result by giving correct and wrong combinations of number and width

- **test3:** check if the following expressions are true:
  Bitpack_fitsu(n, w) == Bitpack_fitsu(n << 4, w + 4)
  Bitpack_fitsu(n, w) == Bitpack_fitsu(n >> 3, w - 3)

-------------------------------------------

For functions:

```
uint64_t Bitpack_getu(uint64_t word, unsigned width, unsigned lsb)
int64_t Bitpack_gets(uint64_t word, unsigned width, unsigned lsb)
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,  int64_t value)
```

- **test4:** test if we can get the original number after putting it into some field of the word

- **test5:** test if the “get” and “new” functions affect the fields outside our target fields.

**b. test compress40 and decompress40**

1. test correctness
   try different size image(from 1 pixel to very large image)
   by printing out small image with several pixel to see if the compress/decompress works correctly
   by displaying image experiencing compress and decompress to see if the result is correct.

2. test bad format file, incomplete file , null file.

3. test efficiency

compress it and then decompress it, compare the final image to the original image to
see how much information is lost.

**4 Handling Part C:**

To prepare for later changing, we have to break down the whole program into as primitive sub functions as possible. Each small function only focuses on a simple task; while we also have to provide proper interface and hide all implementation details from outside. Once one of them has to be change, we only need to change corresponding one.

**5 Points where lose information:**

To compress an image and then decompress it back will experience the following procedure which loss information:

1, trim the original data into even row and column: minor lost;
2, convert RGB to CIE : minor lost on information due to imprecision calculation of floating point data;
3, convert CIE block to float pixel:minor lost on information due to imprecision calculation of floating point data;
4, convert float pixel to int pixel and convert int pixel to float pixel: major lost on information due to the quantization algorithm;
5, convert float pixel to CIE block: major lost on information,since we replace the original Pr and Pb with their average.
6, convert CIE to RGB : minor lost on information due to imprecision calculation of floating point data;