Thema: Zeiger, Strukturen & dynamische Speicherverwaltung

Letzter Abgabetermin: Im vierten Praktikumstermin

Aufgabe:[1]
In dieser Aufgabe wird eine schnelle Anwendung erstellt, die eine Bitmap Bilddatei einlesen, auswerten, verändern und speichern kann. Diese Anwendung kann unter Linux, OS-X oder Mircosoft Windows ausgeführt werden. Die Entwicklung kann mit Eclipse, Netbeans, dem MinGW C Compiler o. ä. erfolgen. Es dürfen keine Funktionen zur Manipulation von Bilddateien verwendet werden.

Im Detail führt die Anwendung folgende Schritte aus:

- Einlesen der Bilddatei
  Die Bilddateien sind entsprechend dem Microsoft .BMP-Dateiformat aufgebaut. Sie haben folgende Eigenschaften:
  - 8 Bit pro Pixel
  - Verwendung einer Palette
  - RLE-Kodierung
- Zur weiteren Bearbeitung sollen die Pixeldaten der Dateien eingelesen und in einem dynamisch angelegten Array (variable length array) mit nrows * ncols Elementen geschrieben werden. Die Handhabung von variable length arrays kann dem unter Emil abgelegten Beispielprogramm entnommen werden.
- Auswertung der Datei
  Die Dateien enthalten grüne und rote Rechtecke, die sich nicht überlappen. Bestimmen Sie die Größe und die Position dieser Rechtecke.
- Verändern der Datei
  Zeichnen Sie einen blauen Rahmen ein, so dass die Rechtecke innerhalb des Rahmens liegen. Jede Kante des blauen Rahmens soll mindestens an einer Kante eines Rechtecks liegen.
- Speichern der Datei
  Speichern Sie die Datei im 24-Bit Modus ohne Palette.

Wichtig ist eine möglichst schnelle Abarbeitung der einzelnen Schritte. Nutzen Sie Profiling. Optimieren Sie die notwendigen Algorithmen. Messen Sie die benötigten Zeiten und geben Sie diese auf dem Bildschirm aus.
Achten Sie insbesondere darauf, dass bei allen von Ihnen implementierten Algorithmen die vorgesehenen Feldgrenzen nicht überschritten werden.

Hinweise:
- Lesen und Schreiben von Binärdateien
  Zum Lesen und Schreiben der Bilddateien sollten Sie folgende Funktionen verwenden:
  ```
  FILE* fopen(char const *fname, char const *mode);
  int fclose(FILE *stream);
  ```

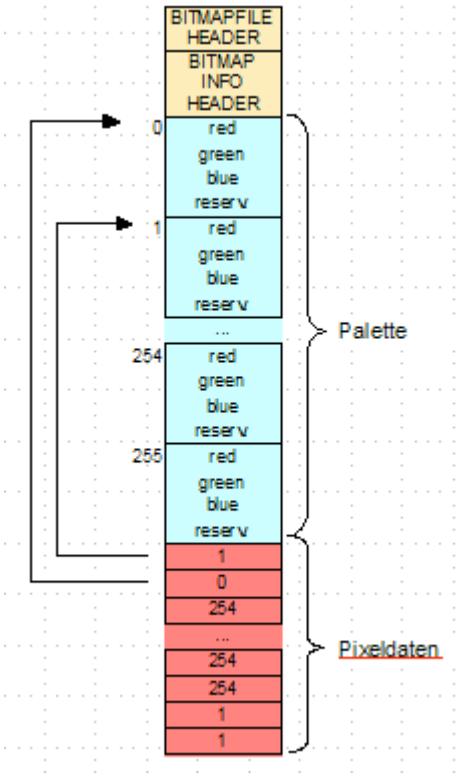
[1] Diese Aufgabe basiert auf Unterlagen zum GSP Praktikum von Prof. Dr. Heiner Heitmann, HAW Hamburg.

```
size_t fread(void *buf, size_t size, size_t count, FILE *stream);
size_t fwrite(void const *buf, size_t size, size_t count, FILE *stream);
int fseek(FILE *stream, long offset, int whence);
```

Eine Beschreibung dieser Funktionen finden Sie zum Beispiel unter http://pronix.linuxdelta.de/C/standard_C/c_programmierung_19_3.shtml#17.

- Format einer BMP-Bilddatei

Die Pixel einer BMP-Bilddatei können durch unterschiedliche Anzahl Bits beschrieben werden, üblich sind 24-Bit pro Pixel oder 8-Bit pro Pixel. Bei 24-Bit pro Pixel werden pro Pixel drei 8-Bit Zahlen verwendet, die den Rot-, Grün- und Blauanteil der Farbe angeben. D.h. jedes Pixel wird durch die Struktur




```
typedef struct tagRGBTRIPLE {
  unsigned char rgbtBlue;
  unsigned char rgbtGreen;
  unsigned char rgbtRed;
} RGBTRIPLE;
```

beschrieben.
So ist z. B. Schwarz durch RGBTRIPLE black = {.rgbtBlue=0, .rgbtGreen=0, .rgbtRed=0};
und Weiss durch RGBTRIPLE white = {.rgbtBlue=255, .rgbtGreen=255,. rgbtRed=255};
definiert.

Eine Bilddatei, die pro Pixel 8-Bit verwendet, benutzt zur Farbdefinition eine sogenannte Palette. Die Palette ist ein Array mit 256 Elementen, das die verwendeten Farben enthält. Jedes Element dieses Arrays wird durch folgende Struktur beschrieben:

```
typedef struct tagRGBQUAD {
  unsigned char    rgbBlue;
  unsigned char    rgbGreen;
  unsigned char    rgbRed;
  unsigned char    rgbReserved;
} RGBQUAD;
```

Die 8-Bit Pixeldaten sind Indizes, die auf die entsprechende Farbe innerhalb der Palette verweisen.

Eine Bilddatei besteht aus folgenden Komponenten (s. Abbildung):

1. BITMAPFILEHEADER
   Identifiziert die Datei und enthält einige Informationen bezüglich dem Aufbau der Datei.
2. BITMAPINFOHEADER
   Enthält Angaben über die Größe des Bildes, das verwendete Format und die Art der Komprimierung.
3. Palette (nicht vorhanden bei 24-Bit pro Pixel)
4. Pixeldaten

Eine genaue Beschreibung des Formats findet man unter https://msdn.microsoft.com/en-us/library/dd183391(VS.85).aspx. Microsoft verwendet dabei folgende Definitionen für die Basistypen:

```
typedef char                CHAR;
typedef short               SHORT;
typedef long                LONG;
typedef unsigned long       DWORD;
typedef int                 BOOL;
typedef unsigned char       BYTE;
typedef unsigned short      WORD;
```

Nachfolgend eine vereinfachte Darstellung der verwendeten Strukturen.

Die Struktur BITMAPFILEHEADER ist wie folgt definiert:

```
typedef struct tagBITMAPFILEHEADER {
  WORD    bfType;
  DWORD   bfSize;
  WORD    bfReserved1;
  WORD    bfReserved2;
  DWORD   bfOffBits;
} BITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

wobei

- bfType: Specifies the file type, must be BM (0x4d42).
- bfSize: Specifies the size, in bytes, of the bitmap file.
- bfReserved1: Reserved; must be zero.
- bfReserved2: Reserved; must be zero.
- bfOffBits: Specifies the offset, in bytes, from the beginning of the BITMAPFILEHEADER structure to the bitmap bits.

Die Struktur BITMAPINFOHEADER ist wie folgt definiert:

```
typedef struct tagBITMAPINFOHEADER{
  DWORD  biSize;
  LONG   biWidth;
  LONG   biHeight;
  WORD   biPlanes;
  WORD   biBitCount;
```

```
  DWORD  biCompression;
  DWORD  biSizeImage;
  LONG   biXPelsPerMeter;
  LONG   biYPelsPerMeter;
  DWORD  biClrUsed;
  DWORD  biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

wobei

- biSize: Specifies the number of bytes required by the structure (==40).
- biWidth:Specifies the width of the bitmap, in pixels.
- biHeight: Specifies the height of the bitmap, in pixels.
- biPlanes: Specifies the number of planes for the target device. This value must be set to 1.
- biBitCount: Specifies the number of bits-per-pixel. The biBitCount member of the BITMAPINFOHEADER structure determines the number of bits that define each pixel and the maximum number of colors in the bitmap. This member must be one of the following values.

Value	Meaning
8	The bitmap has a maximum of 256 colors, and the bmiColors member of BITMAPINFO contains up to 256 entries. In this case, each byte in the array represents a single pixel.
24	The bitmap has a maximum of 2^24 colors, and the bmiColors member of BITMAPINFO is NULL. Each 3-byte triplet in the bitmap array represents the relative intensities of blue, green, and red, respectively, for a pixel.

- biCompression: Specifies the type of compression for a compressed bottom-up bitmap (top-down DIBs cannot be compressed). This member can be one of the following values.

Value	Meaning
BI_RGB (=0)	An uncompressed format.
BI_RLE8 (=1)	A run-length encoded (RLE) format for bitmaps with 8 bpp. The compression format is a 2-byte format consisting of a count byte followed by a byte containing a color index. For more information, see Bitmap Compression.

- biSizeImage: Specifies the size, in bytes, of the image. This may be set to zero for BI_RGB bitmaps.
- biXPelsPerMeter: Specifies the horizontal resolution, in pixels-per-meter, of the target device for the bitmap. An application can use this value to select a bitmap from a resource group that best matches the characteristics of the current device.
- biYPelsPerMeter: Specifies the vertical resolution, in pixels-per-meter, of the target device for the bitmap.

- biClrUsed: Specifies the number of color indexes in the color table that are actually used by the bitmap. If this value is zero, the bitmap uses the maximum number of colors corresponding to the value of the biBitCount member for the compression mode specified by biCompression.
- biClrImportant: Specifies the number of color indexes that are required for displaying the bitmap. If this value is zero, all colors are required.

- Bitmap Compression
  Windows versions 3.0 and later support run-length encoded (RLE) formats for compressing bitmaps that use 4 bits per pixel and 8 bits per pixel. Compression reduces the disk and memory storage required for a bitmap.

  Compression of 8-Bits-per-Pixel Bitmaps
  When the biCompression member of the BITMAPINFOHEADER structure is set to BI_RLE8, the DIB is compressed using a run-length encoded format for a 256-color bitmap. This format uses two modes: encoded mode and absolute mode. Both modes can occur anywhere throughout a single bitmap.

  Encoded Mode
  A unit of information in encoded mode consists of two bytes. The first byte specifies the number of consecutive pixels to be drawn using the color index contained in the second byte. The first byte of the pair can be set to zero to indicate an escape that denotes the end of a line, the end of the bitmap, or a delta. The interpretation of the escape depends on the value of the second byte of the pair, which must be in the range 0x00 through 0x02. Following are the meanings of the escape values that can be used in the second byte:

Second byte	Meaning
0	End of line.
1	End of bitmap.
2	Delta. The two bytes following the escape contain unsigned values indicating the horizontal and vertical offsets of the next pixel from the current position.

  Absolute Mode
  Absolute mode is signaled by the first byte in the pair being set to zero and the second byte to a value between 0x03 and 0xFF. The second byte represents the number of bytes that follow, each of which contains the color index of a single pixel. Each run must be aligned on a word boundary.

Following is an example of an 8-bit RLE bitmap (the two-digit hexadecimal values in the second column represent a color index for a single pixel):

Compressed data	Expanded data
03 04	04 04 04
05 06	06 06 06 06 06
00 03 45 56 67 00	45 56 67
02 78	78 78
00 02 05 01	Move 5 right and 1 down
02 78	78 78
00 00	End of line
09 1E	1E 1E 1E 1E 1E 1E 1E 1E 1E
00 01	End of RLE bitmap

Unter Emil finden Sie zwei BMP Übungsdateien.

Unterlagen, die vor dem Praktikum verschickt werden müssen:
Erstellen Sie ein schriftliches Konzept. **Dieses Konzept muss spätestens am Abend vor dem ersten Versuchstag via e-Mail an Silke Behn und mich geschickt werden**. Dieses Konzept muss folgende Themen beinhalten:

- Analyse der Aufgabenstellung
- Sinnvolles Modulkonzept

Abnahme der Aufgabe im Praktikum:
- Das Konzept muss vorliegen.
- Kommentierter Quellcode, der dem C-Coding Style (s. Emil) entspricht, muss vorliegen.
- Testfälle
- Sie müssen Ihr Programm erklären können.