Thema: Dateiformate, Strukturen und Felder
Letzter Abgabetermin: Im Laufe des vierten Praktikumstermins

Aufgabe: In dieser Aufgabe wird eine Anwendung für das ITS-Board erstellt, die ein Bild über die serielle Schnittstelle empfängt, gegebenenfalls auf die Größe des LCD-Displays verkleinert und auf diesem darstellt. Das Bild ist im Microsoft .BMP-Format kodiert. Zur Vereinfachung ist die Aufgabe in drei aufeinander aufbauende Teilaufgaben unterteilt. Die letzte Teilaufgabe ist optional.

Teilaufgabe a: Die .BMP-Datei wird über die serielle Schnittstelle (USB-UART) eingelesen, dekodiert und Pixel für Pixel auf dem LCD-Display ausgegeben. Das Display ist 320 Zeilen hoch und 480 Spalten breit. **Pixel, die außerhalb des Bildbereichs liegen, werden nicht dargestellt.**

Die .BMP-Dateien liegen auf Ihrem PC und werden über USB-UART an das ITS-Board übertragen. Der Code für die Datenübertragung wird zur Verfügung gestellt. Unter MS-Teams finden Sie eine einfache Python GUI, die mehrere Bilder nacheinander an das ITS-Board überträgt. Weiterhin finden Sie unter MS-Teams das C Modul `input.c`. Es liest die .BMP-Dateien ein. Über eine Funktion können Sie die Übertragung der nächsten .BMP-Datei starten. Sie können sowohl ganze Blöcke (analog zu `fread`) der aktuellen Bilddatei als auch einzelne Zeichen einlesen.

Draw

Im Modul `LCD_GUI` finden Sie die Zeichenfunktionen für das LCD-Display. In dieser Teilaufgabe benötigen Sie die Funktionen `GUI_clear` und `GUI_drawPoint`. Beim Aufruf von `GUI_drawPoint` setzen Sie den Parameter `dotPixel` auf `DOT_PIXEL_1X1` und den Parameter `dotStyle` auf `DOT_FILL_AROUND`.

Das Microsoft .BMP-Format unterstützt mehrere Kodierungen, wobei hier nur folgende Kodierung eingelesen werden muss:

8 Bit pro Pixel mit RLE-Kodierung unter Verwendung einer Farbpalette

Error →

Bei Fehlern im Dateiformat oder einer anderen Kodierung, erzeugt Ihr Programm eine entsprechende Fehlermeldung. Das .BMP-Format ist am Ende der Aufgabenstellung beschrieben.

Header

Angehende Informatiker*innen sollten Dateiformate lesen und bearbeiten können. Das .BMP-Format ist relativ einfach und somit eine gute Übung. Üblicherweise findet man am Anfang eines Dateiformats mehrere Header, die C-`structs` entsprechen. Ein C Modul mit diesen C Typen finden Sie unter MS-Teams. Da diese Header zentrale Informationen enthalten, sind elementare Konsistenzprüfungen der Header sinnvoll. Eine entsprechende C-Funktion finden Sie ebenfalls unter MS-Teams.

Error

Die Fehlerausgabe können Sie einfach halten. Geben Sie im Fehlerfall eine entsprechende Textmeldung auf dem LCD-Display aus. Ein entsprechendes C-Makro, das einen Fehlertext zusammen mit der Zeile und dem Dateinamen, wo der Fehler aufgetreten ist, ausgibt finden Sie

ebenfalls unter MS-Teams.

Damit mehrere Bilder hintereinander angezeigt werden können, ist es ausreichend, wenn über den Druck auf einen Taster des ITS-Boards der User die Darstellung des nächsten Bilds anfordert.

Im .BMP-Format ist ein Pixel in drei Byte kodiert (s. Beschreibung des .BMP-Formats). Sie beschreiben die Farbanteile von Blau, Grün und Rot.
Für das LCD-Display wird ein Pixel in einem 16 Bit Wert kodiert, wobei die drei Farbanteile wie folgt auf die 16 Bit verteilt sind:

Hinweise

- Gehen Sie schrittweise vor und überprüfen Sie einzelne Teilfunktionen. Fangen Sie frühzeitig möglichst viele Fehlersituationen ab und nutzen das Makro zur Ausgabe von Fehlern. **Im Endeffekt spart dies viel Zeit**.
- Der Speicherplatz des ITS-Boards reicht nicht zur Speicherung des gesamten Bilds als großes zweidimensionales Feld aus. Dies ist auch nicht nötig, sie können Pixel für Pixel ausgeben.
- Neben der Umsetzung selbst ist ein Aspekt dieser Aufgabe, dass Sie das .BMP-Format verstehen und im Programm umsetzen. Arbeiten Sie das Format genau durch. Diskutieren Sie es mit anderen Studierenden aus Ihrem Semester. **Das spart viel Zeit.** Schauen Sie sich die Datei eines kleinen komprimierten Bilds im HEX Editor an. Bilden Sie die Werte auf die unten dargestellten Strukturen ab.
- Die Datenübertragung der Bilder findet über den USB-UART statt. Die Ausgabe der Funktion printf ist auf den USB-UART umgelenkt. Das bedeutet, dass Sie in dieser Aufgabe printf nicht benutzen dürfen, sonst treten Fehler bei der Datenübertragung auf.
- Machen Sie **erste Tests** mit kleinen Bildern, das erleichtert die Fehlersuche.
- Typischer Ablauf der Software auf dem ITS-Board:
  - Einmaliger Aufbau der Verbindung über `initInput`
  - Pro File:
    - `openNextFile` // Startet die Übertragung
    - Lesen der Daten mit `COMread` und / oder `nextChar`
- Typischer Ablauf in der Python GUI:
  - Erstellung der Verbindung mit **Connect**
  - Auswahl einer Menge von Bilddateien mit **Load File**
  - Start der Übertragung der selektierten Files mit **Start**

Teilaufgabe b – Beschleunigung der Ausgabe: Aufgrund des relativ langsamen SPI-Busses, über den die Daten an das LCD-Display übertragen werden, ist die Lösung von Aufgabe (a) langsam.

Verwendet man wie in Teilaufgabe (a) die Funktion `GUI_drawPoint`, werden für jedes Pixel des Bilds jeweils die Koordinaten des Pixels und der Farbwert übertragen.
Da das .BMP-Format im Prinzip die Pixel einer Datei zeilenweise hintereinander liefert, kann man alternativ die Farbwerte einer Bildzeile in einem Feld speichern und dann die gesamte Bildzeile über die Funktion `GUI_WriteLine` ausgeben. Damit werden deutlich weniger Daten über den SPI-Bus übertragen und somit das Programm beschleunigt. Diese Beschleunigung müssen Sie in dieser Teilaufgabe durchführen.

Teilaufgabe c: In dieser Teilaufgabe werden große Bilder auf die Größe des LCD-Displays (480 x 320) reduziert. Es werden Bilder bis zu einer Breite von 5 * 480 Pixeln und einer Höhe von 5 * 320 Pixeln unterstützt. Somit muss maximal auf 1/5 verkleinert werden.

Mit Blick auf die Rechenzeit ist zur Verkleinerung der Bilder ein Box Algorithmus gut geeignet. Wird zum Beispiel ein Bild auf 1/5 verkleinert, so werden immer Kacheln von 5 x 5 Pixeln auf ein Pixel reduziert. Dazu wird für die einzelnen Farbanteile der 25 Pixel der Kachel jeweils der Mittelwert gebildet. Hier der entsprechende an Python angelehnte Pseudo Code:

```
scale_x= 480 / width_picture
scale_y = 320 / height_picture
scale = min(scale_x, scale_y)  # Damit keine Verzerrung auftritt
box_width = int(np.ceil(1/scale))
box_height = int(np.ceil(1/scale))
for y in range(320):
    for x in range(480):
        # Koordinaten im Bild
        x_ = int(np.floor(x/scale_x))
        y_ = int(np.floor(y/scale_y))
        x_end = min(x_ + box_width, 480-1) # keine Bereichsüberschreitung
        y_end = min(y_ + box_height, 320-1) # keine Bereichsüberschreitung
        Mittelwert der Pixel der Kachel (x_,y_) bis (x_end,y_end) bestimmen
        Gebe Farbwert an der Stelle (x,y) auf dem LCD Display aus
```

Liegt das ganze Bild im Speicher, wozu der Speicherplatz des ITS-Boards nicht ausreicht, ist der Algorithmus einfach umsetzbar. Wir brauchen eine effizientere Speicherverwaltung. Da maximal um den Faktor 5 verkleinert wird, ist eine Box maximal 5 Zeilen hoch. Somit ist es ausreichend, wenn die „letzten“ 5 Zeilen im Speicher gehalten werden.
Die zeilenweise Ausgabe kann aus Aufgabe (b) übernommen werden.

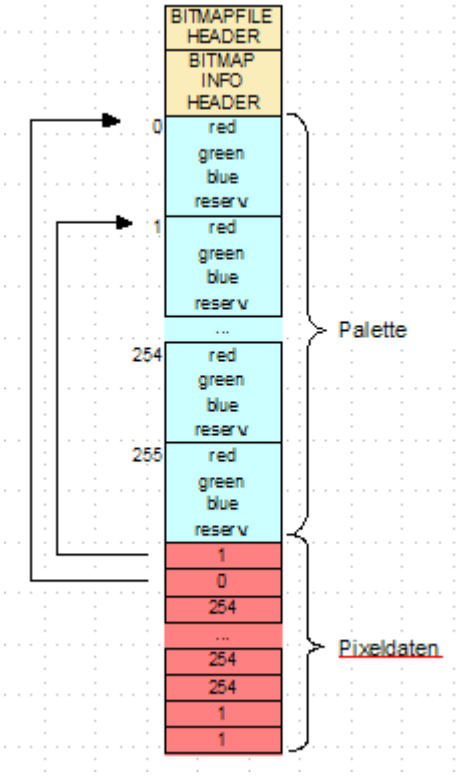
Abnahme der Aufgabe im Praktikum

- Das Modulkonzept. Vor der Codeentwicklung muss ein sinnvolles Modulkonzept ins git Repo auf dem Server hochgeladen und „getagged“ sein.
- Kommentierter Quellcode, der dem C-Coding Style entspricht, muss vorliegen.
- Das Programm muss die in MS-Teams bereitgestellten Testbilder verarbeiten können. Für die fehlerhaften Bilddateien muss eine entsprechende Fehlermeldung auf dem LCD-Display ausgegeben werden.
- Sie müssen Ihr Programm erklären können.

BMP Format

https://msdn.microsoft.com/en-us/library/dd183391(VS.85).aspx beschreibt zum Beispiel das BMP-Format. Hier eine Zusammenfassung der wesentlichen Aspekte.

- Die Pixel einer BMP-Bilddatei können durch eine unterschiedliche Anzahl von 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. Z.B. entspricht das RGBTRIPLE {rgbtBlue=0, rgbtGreen=0, rgbtRed=0} der Farbe Schwarz. Das RGBTRIPLE {rgbtBlue=255, .rgbtGreen=255,. rgbtRed=255} entspricht der Farbe Weiss.

- Eine Bilddatei, die pro Pixel 8-Bit verwendet, benutzt zur Farbdefinition eine sogenannte Palette. Die Palette speichert in einem Feld, das bis zu 256 Elemente groß ist, die verwendeten Farben. Jedes Element dieses Felds 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 Informationen bezüglich des Dateiaufbaus.
  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
     Das erste Pixel aus der Datei ist unten links im Bild platziert.

- Microsoft verwendet folgende Definitionen für Basistypen:

```
typedef int8_t    CHAR;    // Entspricht nicht Original.
typedef int16_t   SHORT;   // Anpassung auf 32 und 64 Bit
typedef int32_t   LONG;    // Umgebung
typedef uint32_t  DWORD;
typedef int32_t   BOOL;
typedef uint8_t   BYTE;
typedef uint16_t  WORD;
```

- 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[1] 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.
Es gibt weitere Werte, die hier nicht beachtet werden.	

[1] bmiColors ist die Farbtabelle

- 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.
Es gibt weitere Werte, die hier nicht beachtet werden.	

- 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

- **Nicht komprimierte BMP Files**

  Bei nicht komprimierten Dateien stehen die Zeilen hintereinander im Speicher. Gemäß dem Eintrag in der Info Struktur wird ein Pixel entweder durch ein Byte, das in die Farbpalette zeigt, oder eine RGBTRIPLE Struktur beschrieben.
  Achtung: Die Anzahl der Bytes, die eine Zeile beschreiben, muss durch 4 teilbar sein. Wenn dies nicht der Fall ist, wird mit padding Bytes aufgefüllt. Die Anzahl Bytes pro Zeile (inklusive padding Bytes) wird wie folgt berechnet:

  (((Breite der Zeile)*(Anzahl Bits pro Pixel Eintrag) + 31) / 32) * 4

Kleine Starthilfen

- Öffnen Sie in einem Hex Editor die Bilddatei

  6x10_8_bit_nicht_komprimiert_mit_padding_bytes.bmp

  Am Anfang der Datei steht die C-`struct` BITMAPFILEHEADER. Sie ist packed und 2 + 4 + 2 + 2 + 4 = 14 Byte groß. Der Hex Editor stellt für die ersten 14 Bytes folgende Werte dar:

0x42	0x4D	0x86	0x04	0x00	0x00	0x00	0x00	0x00	0x00	0x36	0x04	0x00	0x00

  Ein Codestück der Art

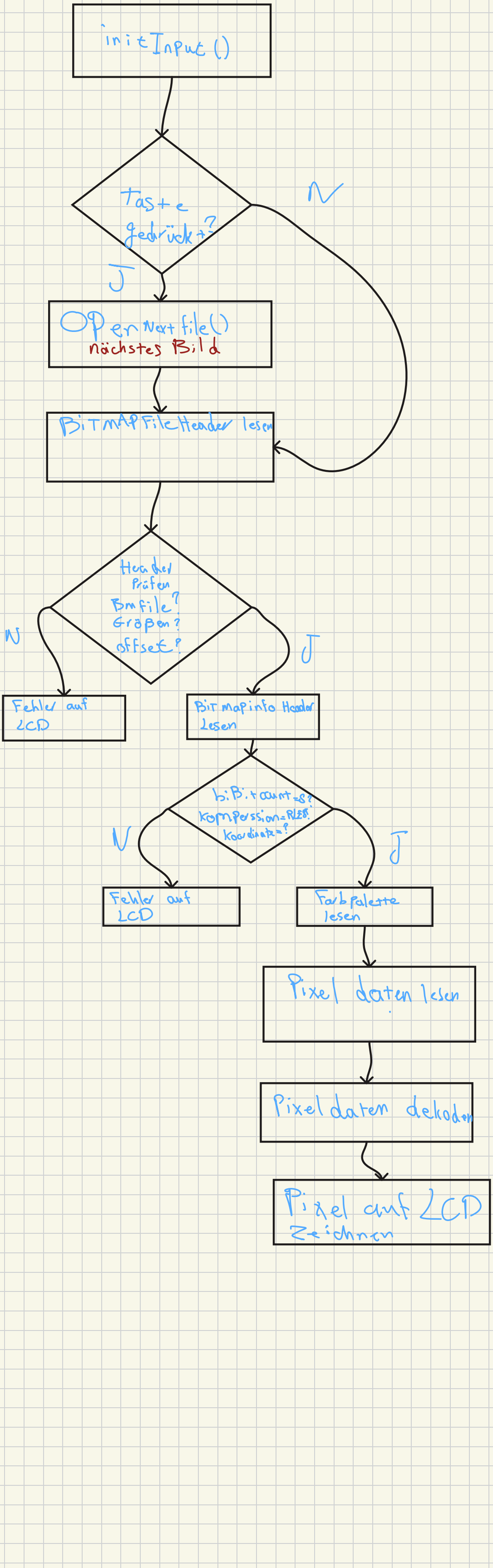
```
static BITMAPFILEHEADER fileHeader;

openNextFile();

ERROR_HANDLER(1!= COMread((char *) &fileHeader,
   sizeof(BITMAPFILEHEADER), 1),
   "readHeaders: Error during read.");

basicChecks(&fileHeader, NULL);
```

  liest den ersten Header byteweise aus dem File in den Speicherbereich, auf den `&fileHeader` zeigt. Da das .BMP-Format und das ITS-Board beide Little-Endian Kodierung verwenden, funktioniert das blockweise Einlesen problemlos. Schauen Sie sich die Darstellung von `fileHeader.bfSize` im Memory Fenster und im Watch Fenster des Keil Debuggers an, dann wird dieser Aspekt klar.
  Nur bei `fileHeader.bfType` hat man fälschlicherweise den Eindruck, dass die Bytes vertauscht sind. Dies ist jedoch nur bei der Signature `"BM"` der Fall, da die einzelnen Bytes `'B'` und `'M'` hintereinander im Speicher stehen und als 2 Byte großer Wert in der C-`struct` dargestellt werden.


initInput()
Taste gedrückt?
N
J
OpenNextFile()
nächstes Bild
BitMAPFileHeader lesen
Header Prüfen
Bmfile?
Größen?
offset?
N
J
Fehler auf LCD
BitmapInfo Header Lesen
biBitCount=8?
kompression=RLE8?
Koordinate=?
N
J
Fehler auf LCD
Farbpalette lesen
Pixel daten lesen
Pixel daten dekodern
Pixel auf LCD Zeichnen

main

error Handler

input

decoder

decodeAndDisplayRLE

decodeRLE8

PutPixelClipped

getColorFromPalette

loadColorPalette

resetDecoderState

headers

BMP_Types

MS_BaseTyp