1 Testataufgabe: BMP Funktionsplotter

In dieser Übung sollen Rastergrafiken mit dem Windows Bitmap (BMP)[1] Format genutzt werden. Laden Sie sich die Datei `libBMP.h` (→ StudIP) herunter und machen Sie sich mit dem gegebenen Quellcode und mit den Grundlagen der Dateiein- und -ausgabe (`FILE*`, `fopen`, `fclose`)[2] vertraut. Rastergrafiken können nur diskrete Elemente zeichnen, z.B. die Parabel in Abbildung 1 sieht bei einem groben Raster sehr „pixelig“ aus. Je feiner das Raster gewählt wird, umso weniger fällt dem Betrachter auf, dass es sich nicht wirklich um eine kontinuierliche Funktion handelt.

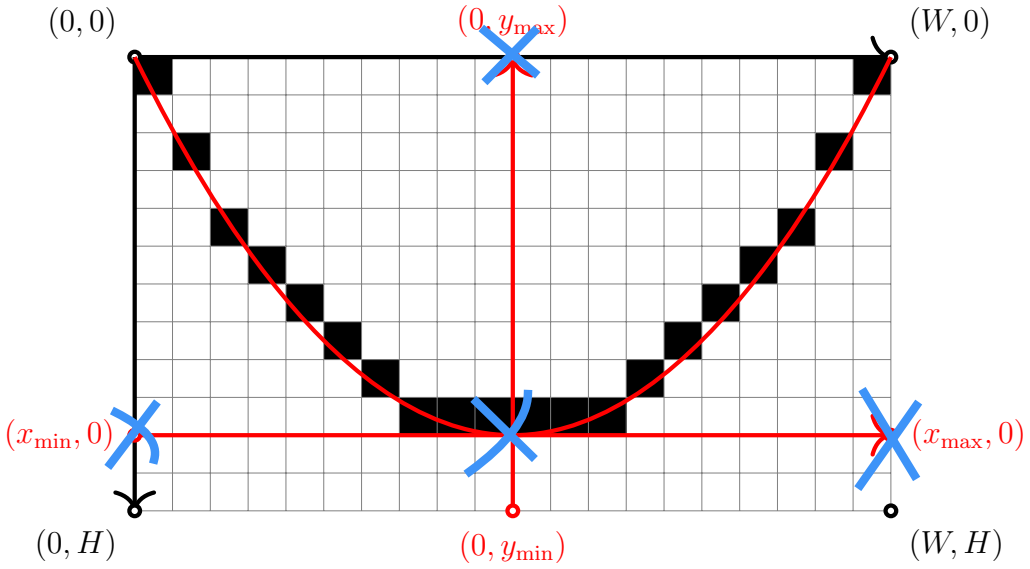

Abbildung 1: Mathematische Koordinaten und Funktionsgraph (rot), BMP-Koordinaten und gerasterter Funktionsgraph (schwarz).

1.1 Weißer Hintergrund **(2 Punkte)**

Erstellen Sie ein eindimensionales dynamisches Array der Größe $W \times H$ mit dem Datentyp `uint32_t`, was den Hintergrund der Rastergrafik repräsentiert. Füllen Sie jeden Pixel des Arrays mit der Farbe `COLOR_WHITE` (→ `libBMP.h`) und erstellen mithilfe der `bmp_create`-Funktion (→ `libBMP.h`) ein komplett weißes BMP-Bild. Geben Sie bei Programmende den dynamisch allozierten Speicher des Arrays wieder frei.

1.2 Übersetzung BMP- zu mathematischen Koordinaten **(4 Punkte)**

In Abbildung 1 wird deutlich, dass sich das BMP- und das mathematische Koordinatensystem unterscheiden. Schreiben Sie eine Funktion `toMath` welche als Eingabeparameter einen BMP-Punkt (X, Y) mittels *call-by-value* und einen mathematischen Punkt (x, y) mittels *call-by-*

[1] `https://de.wikipedia.org/wiki/Windows_Bitmap`
[2] `http://openbook.rheinwerk-verlag.de/c_von_a_bis_z/016_c_ein_ausgabe_funktionen_005.htm`

reference übergeben bekommt und letzteren mithilfe der Formeln (1) berechnet.

$$\begin{aligned} x &:= x_{\min} + \frac{X \cdot (x_{\max} - x_{\min})}{W} \\ y &:= y_{\min} + \frac{Y \cdot (y_{\max} - y_{\min})}{H} \end{aligned} \tag{1}$$

Überlegen Sie sich, welche Datentypen für die Ein- und Ausgabe sinnvoll sind und wann in der Berechnung eine explizite Typumwandlung verlustfrei ist. Bei der Formel (1) wurde noch nicht berücksichtigt, dass die BMP- und mathematische und Y-Achse entgegengesetzt sind!
Stellen Sie analoge Überlegungen für die Übersetzung von mathematischen zu BMP-Koordinaten an und implementieren Sie die Funktion `toBMP`.

1.3 Koordinatenachsen **(2 Punkte)**

Zur Orientierung im Funktionsplot sollen die X- und Y-Achse als einfache Linien eingezeichnet werden. Den mathematischen Koordinatenursprung $(x, y) = (0, 0)$ als BMP Koordinaten (X, Y) können Sie z.B. mit dem Funktionsaufruf `toBMP(0.0, 0.0, &X, &Y)` erhalten.

1.4 Funktionsplot **(2 Punkte)**

Ermöglichen Sie es, dass ihr Tutor vor dem Kompilieren eine beliebige Funktion, z.B. aus der `<math.h>`, mit Definitionsbereich ($x_{\min}$, $x_{\max}$) in Ihren Programmquelltext eingeben kann. Bei der Programmausführung soll eine BMP-Datei erstellt werden, welche diese Funktion grafisch darstellt.

2 Präsenzaufgaben

1. Schreiben Sie eine Funktion, welche alle Primzahlen zwischen 2 und 100 bestimmt und ausgibt. Gehen Sie dafür folgendermaßen vor: Zuerst wird ein Array mit den Zahlen $2, 3, \ldots, 100$ initialisiert. Beginnend mit der 2 werden alle Vielfachen der zuletzt gefundenen Primzahl markiert (gleich 0 gesetzt). Die neue Primzahl ist die nächste unmarkierte Zahl. Ist das Quadrat der zuletzt gefundenen Primzahl größer als 100, so enthält das Array bereits die gesuchten Primzahlen.

2. Schreiben Sie ein vollständiges Programm mit zwei Funktionen `swap1` und `swap2`, die zwei übergebene Eingabeargumente (`int`) vertauschen. In Funktion `swap1` sollen die Parameter durch *Call By Value* und in `swap2` durch *Call By Reference* übergeben werden. Erklären Sie den Unterschied zwischen *Call By Value* und *Call By Reference*.

3. Schreiben Sie ein Programm, das sechs wöchentliche Lottozahlen (6 aus 49) vorschlägt. Beachten Sie dabei, dass beim Lotto keine Zahl doppelt vorkommen darf. Verwenden Sie dafür den Zufallszahlengenerator aus der Standardbibliothek `<stdlib.h>`.

   Tipp: `rand()` liefert eine Pseudo-Zufallszahl aus dem Bereich von `0` bis `RAND_MAX`. Zur Projektion der Zufallszahl auf ein Intervall `[a,b]` berechne den ganzzahligen Rest bei Division durch `b-a+1` und addiere `a` dazu.

4. Schreiben Sie ein Programm, das dem Benutzer folgendes Auswahlmenü anzeigt:

   ```
   (1) a + b        (2) a - b        (5) Programmende
   (3) a * b        (4) a / b
   ```

   Das Programm soll die Tastatureingabe einer Ziffer zwischen 1 und 5 erwarten und eine entsprechende Operation auf zwei einzulesende `int`-Zahlen anwenden. Eine Fehlerbehandlung bei Division durch 0 wird erwartet.

5. Schreiben Sie ein vollständiges Programm, das eine ganze Zahl $n > 0$ von der Tastatur einliest und anschließend zwei `double`-Arrays `v,w` der Länge n erzeugt und diese ebenfalls von der Tastatur einliest. In einer Funktion `dotprod` soll das Skalarprodukt beider Arrays $s = \sum_{k=0}^{n-1} v[k] \cdot w[k]$ berechnet und im Hauptprogramm ausgegeben werden.

6. Schreiben Sie ein Makro `mymac`, das 100 mal `"Hello world"` ausgibt. Die einzige Befehlszeile der `main`-Funktion sei der Makroaufruf `mymac`.

7. Makros können auch Parameter enthalten:

   ```
   #define makroname(param1,param2,...) makrobefehl(e)
   ```

   Schreiben Sie ein Makro `max`, welches das Maximum zweier Zahlen berechnet.