TUHH
Technische
Universität
Hamburg

3. Übungsblatt : Informatik für Ingenieur*Innen

Ausgabe: Freitag, 31. Oktober 2025
Bearbeitung: 03.11. to 09.11 2025
Besprechung: 10.11. to 14.11. 2025

Prof. Dr.-Ing. Görschwin Fey
Dr. rer. nat. Bernhard J. Berger

Wintersemester 2025/2026

Dieser Übungszettel enthält mehrere Programmieraufgaben, die `BitmapPlusPlus` verwenden.

Aufgabe 1: C++-Grammatik

Im letzten Übungsblatt haben Sie die C++-Grammatik zusammengestellt. In der Vorlesung zu Bedingungen und Schleifen haben Sie neue Teile der C++-Grammatik kennengelernt. Vervollständigen Sie ihre Grammatik um die neuen Konstrukte.

Aufgabe 2: Bildbearbeitung mit BitmapPlusPlus

In dieser Aufgabe sollen Sie Programme schreiben, die mit der Bibliothek `BitmapPlusPlus` arbeiten. Bearbeiten Sie die **Aufgaben a) bis e) NICHT alleine zu Hause**, sondern in der Übungsstunde gemeinsam in Dreier- oder Vierergruppen und besprechen Sie Ihre Resultate. Benutzen Sie für diese Aufgaben keine Programmierumgebung oder Compiler, es sei denn, Sie werden explizit dazu aufgefordert.

a) Betrachten Sie den folgenden Farbverlauf:

Abbildung 1: 1D-Farbverlauf

Es soll ein Programm geschrieben werden, welches diesen Farbverlauf generiert. Das Grundgerüst des Programms sieht wie folgt aus:

```
1  #include "BitmapPlusPlus.hpp"
2
3  void create_gradient() {
4      bmp::Bitmap image(25, 1);
5
6
7
8      image.save("colourgradient.bmp");
9  }
10
11 int main() {
12     create_gradient();
13
14     return 0;
15 }
```

Nun soll noch der fehlende Code in den Zeilen 5–7 ergänzt werden. Geben Sie für jede der folgenden Codestücke 1.–7. an, ob dieses das gewünschte Bild in Abbildung 1 generiert. Begründen Sie!
(Hinweis: Es können auch mehrere richtig sein)

1.
```
for(int x = -1; x < 25; x++) {
  image.set(x, 0, bmp::Pixel(x*10,0,0));
}
```

☐ Ja ☐ Nein

Begründung:

2.
```
for(int x = 0; x < 25; x++) {
      image.set(x, 0, bmp::Pixel(x*10,0,0));
}
```

☐ Ja ☐ Nein

Begründung:

3.
```
for(int x = 0; x < 25; ++x) {
    image.set(x, 0, bmp::Pixel(x*10,0,0));
}
```

☐ Ja ☐ Nein

Begründung:

4.
```
for(int x = 0; x > 24; x++) {
    image.set(x, 0, bmp::Pixel(x*10,0,0));
}
```

☐ Ja ☐ Nein

Begründung:

5.

```
1    for(int x = 0; x <= 25; x++) {
2        image.set(x, 0, bmp::Pixel(x*10,0,0));
3    }
```

☐ Ja ☐ Nein

Begründung:

6.

```
1    for(int x = 0; x <= 24; x++) {
2        image.set(x, 0, bmp::Pixel(x*10,0,0));
3    }
```

☐ Ja ☐ Nein

Begründung:

7.

```
1    for(int x = 0; x < 24; x++) {
2        image.set(x, 0, bmp::Pixel(x*10,0,0));
3    }
```

☐ Ja ☐ Nein

Begründung:

b) Geben Sie für die Codestücke, welche nicht zum gewünschten Resultat führen, an, welche der folgenden Bilder beziehungsweise Fehlermeldungen Sie erwarten:

1. Folgendes Bild

   passt zu folgenden Codestücken:

2. Folgendes Bild

   passt zu folgenden Codestücken:

3. Folgende Fehlermeldung
   terminating due to uncaught exception of type bmp::Exception: Bitmap::Set(-1, 0): x,y out of bounds
   passt zu folgenden Codestücken:

4. Folgende Fehlermeldung
   terminating due to uncaught exception of type bmp::Exception: Bitmap::Set(25, 0): x,y out of bounds
   passt zu folgenden Codestücken:

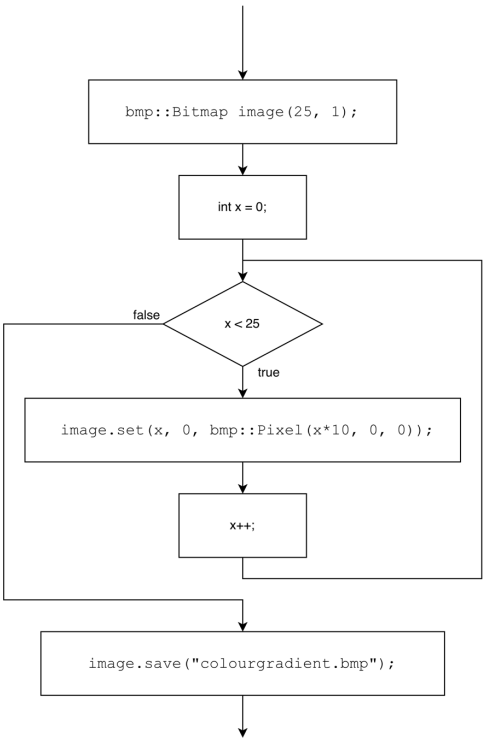
Begründen Sie!

Überprüfen Sie nun Ihre Antworten, in dem Sie die Programme in Ihrer Programmierumgebung ausführen und die Resultate mit Ihren Antworten vergleichen. Versuchen Sie, auftretende Widersprüche aufzulösen.

c) Die Funktion *create_gradient()* soll nun erneut verwendet werden, um den Farbverlauf in Abbildung 1 zu generieren. Benutzen Sie diesmal allerdings eine *while*-Schleife, also keine *for*-Schleife:

```
#include "BitmapPlusPlus.hpp"

void create_gradient() {
    bmp::Bitmap image(25, 1);









    image.save("colourgradient.bmp");
}

int main() {
    create_gradient();

    return 0;
}
```

d) Betrachten Sie nun folgendes Kontrollflussdiagramm:
(Hinweis: Ein Kontrollflussdiagramm ist eine grafische Darstellung, die den Ablauf der einzelnen Programmschritte in einem Programm zeigt. Dabei wird der Richtung der verbindenden Pfeile gefolgt und alle Prozessschritte (Rechtecke) nacheinander ausgeführt. Bei Rauten wird die angegebene Bedingung überprüft und je nach Ergebnis dem 'false'- oder dem 'true'-Pfad gefolgt.)



1. Markieren Sie jedes Rechteck und jede Raute im Flussdiagramm mit je einer anderen Farbe.
2. Markieren Sie nun die entsprechenden Zeilen im folgenden Codestück mit der entsprechenden Farbe.

```
1  void create_gradient() {
2    bmp::Bitmap image(25, 1);
3    int x = 0;
4    while(x < 25){
5      image.set(x, 0, bmp::Pixel(x*10,0,0));
6      x++;
7    }
8    image.save("colourgradient.bmp");
9  }
```

3. Markieren Sie nun im folgenden Stück Code die entsprechenden Programmteile mit der entsprechenden Farbe.

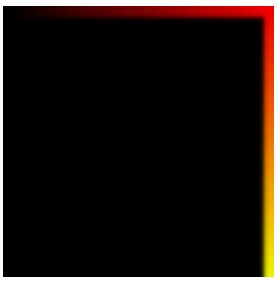
```
1  void create_gradient() {
2    bmp::Bitmap image(25, 1);
3    for(int x = 0; x < 25; x++) {
4      image.set(x, 0, bmp::Pixel(x*10,0,0));
5    }
6    image.save("colourgradient.bmp");
7  }
```

4. Sind die beiden Programme äquivalent? Gibt es Gründe, wieso Sie das eine Programm dem anderen vorziehen würden?

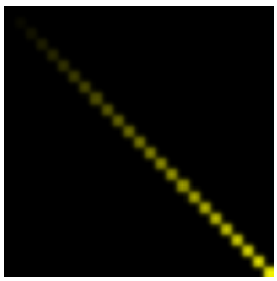
e) Betrachten Sie zuerst folgendes Programm und versuchen Sie seine Funktionsweise zu verstehen.

```
#include "BitmapPlusPlus.hpp"

void create_gradient() {
    bmp::Bitmap image(25, 25);
    for(int x = 0; x  < 25 ; ++x) {
        for(int y = 0; y < 25; ++y) {
            image.set(x, y, bmp::Pixel(x*10, y*10, 0));
        }
    }
    image.save("colourgradient.bmp");
}
int main() {
    create_gradient();

    return 0;
}
```

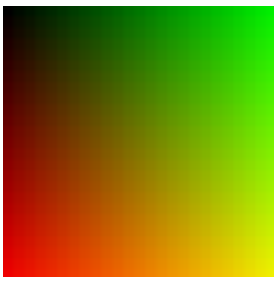
Drei Studierende haben unterschiedliche, falsche Vorstellungen, was das Programm macht. Erläutern Sie, wo der jeweilige Denkfehler liegt.



(a) Anja



(b) Mo



(c) Ali

Anja: *'Die Funktion enthält zwei Schleifen. Die erste Schleife zählt von 0 bis 24 hoch. Also bewegen wir uns auf der x-Achse von ganz links nach ganz rechts und die Pixel werden zunehmend röter. Dann wird die zweite Schleife ausgeführt, die ebenfalls von 0 bis 24 hochzählt. Hier bewegen wir uns von oben nach unten mit einem zunehmenden Grünanteil, was zusammen mit dem rot gelb ergibt.'*

Mo: *'Nein, die Schleifen sind ja verschachtelt. Das heisst, sie werden gleichzeitig ausgeführt. Wir fangen also bei den Koordinaten (0,0) an. Dann wird x um 1 erhöht, dann wird y um 1 erhöht, wir sind also bei (1,1). Dann werden wieder beide erhöht, also sind wir bei (2,2). Dann bei (3,3), dann bei (4,4) usw.. Es ergibt sich also eine Diagonale die zunehmend gelber wird.'*

Ali: *'Nein, die Verschachtelung hat zur Folge, dass für jede Wiederholung der äußeren Schleife, die innere Schleife 25 Wiederholungen durchmacht. Es wird also Zeile für Zeile komplett ausgefüllt, wobei die Pixel nach rechts immer grüner werden und die Zeilen nach unten immer röter. Unten rechts ergibt sich dann gelb.'*

Überprüfen Sie nun Ihre Antworten, in dem Sie das Programm in Ihrer Programmierumgebung ausführen und das Resultat mit Ihren Antworten vergleichen. Versuchen Sie, auftretende Widersprüche aufzulösen.

WICHTIG: Füllen Sie bitte die folgende kurze Umfrage aus:
`https://dual.tuhh.de/dual_limesurvey/426154?lang=de`

f) Schreiben Sie das Programm (`grayscale.cpp`). Die `main`-Funktion soll eine Funktion`void to_gray()` aufrufen, die das Bild `kittens.bmp`, lädt, in ein Schwarz-Weiß-Bild umwandelt und wieder speichert. Um das Bild umzuwandeln, müssen die Farbwerte von jedem Pixel wie folgt gesetzt werden:

$$px_{(x,y)}.* \quad = \quad \frac{(px_{(x,y)}.r + px_{(x,y)}.g + px_{(x,y)}.b)}{3}$$

Hinweis: Die Breite und die Höhe des Bildes können über `image.width()` und `image.height()` abgefragt werden.

Aufgabe 3: Zeichenbibliothek

Im Folgenden programmieren Sie den Anfang einer Zeichenbibliothek. Diese besteht aus der Header-Datei `drawing.hpp`, welche Deklarationen für das Hauptprogramm enthält und der Datei `drawing.cpp`, welche Sie um die Implementierung ergänzen müssen.
In dieser Aufgabe müssen Sie mehrere `.cpp`-Dateien bauen und zu einem Programm linken. Hierfür ist es notwendig, dass sie ihre `.vscode/tasks.json` anpassen. In dieser findet Sie einen Abschnitt, der die Parameter des `g++`-Aufrufs steuert:

```
"args": [
    ...
    "${file}",
    ...
],
```

Diesen Abschnitt bedeutet, dass der Compiler nur die aktuelle Datei übersetzt. Passen Sie diese Zeile so an, dass dort alle Dateien des aktuellen Ordners zusammen übersetzt werden:

```
  "args": [
      ...
      "${fileDirname}/drawing.cpp",
      "${fileDirname}/main.cpp",
      ...
  ],
```

Hinweis: Durch diese Änderung können wir nur noch ein Programm in einem Workspace erstellen. Dies bedeutet, dass sie für jedes neue Programm, welches aus mehreren Dateien besteht einen neuen Ordner anlegen müssen, welcher die ensprechende Konfiguration enthält.

a) Implementieren Sie die Funktion `circle` zum Zeichnen eines Kreises. Die Parameter sind in der Datei `drawing.hpp` dokumentiert und das Programm `main.cpp` verwendet diese Funktion zum Zeichnen eines Kreises. Die Implementierung müssen Sie in der Datei `drawing.cpp` vornehmen.

Implementierungshinweise:

- Die Standard-Header-Datei `cmath` enthält viele vordefinierte Funktionen aus der Mathematik. Hierzu gehört auch die Funktion `std::sqrt`, die dabei hilft, die Wurzel einer Zahl zu bestimmen. Die offizielle Dokumentation zu C++-Funktionen finden Sie auf *cppreference*[1]. Hier finden Sie auch die vollständige Dokumentation von `sqrt`[2].

[1] `https://en.cppreference.com/w/`
[2] `https://en.cppreference.com/w/cpp/numeric/math/sqrt`

- Sie können einen Viertelkreis berechnen, indem Sie den Satz des Pythagoras verwenden ($a^2+b^2=c^2$). Inkrementieren Sie hierfür über die `x`-Werte von `0` bis (einschließlich) `radius`. Stellen Sie die Gleichung nach b um, ersetzen Sie a durch `x` und c durch `radius`. Die Wert b entspricht dem `y`-Wert. Beachten Sie, dass Sie `x` und `y` zu `cx` und `cy` addieren müssen.
- Die weiteren drei Viertelkreise können Sie erzeugen, wenn Sie die Werte `x` und `y` entsprechend von `cx` und `cy` abziehen oder addieren.

b) Implementieren Sie die Funktion `line` zum Zeichnen einer Linie. Die Parameter sind in der Datei `drawing.hpp` dokumentiert und das Programm `main.cpp` verwendet diese Funktion zum Zeichnen mehrerer Linien. Auch hier müssen Sie die Implementierung in der Datei `drawing.cpp` umsetzen.

Implementierungshinweise:

1. Berechnen Sie die Breite (`dx`) und Höhe (`dy`) der Linie.
2. Berechnen Sie zunächst das Maximum aus der Breite und der Höhe der Linie (`upper`). Verwenden Sie hierfür die absoluten Werte. Der berechnete Wert entspricht der Anzahl der Pixel, die Sie berechnen werden.
3. Inkrementieren Sie einen Wert `c` von 0 bis einschließlich der berechneten Obergrenze.
4. Setzen Sie für jeden Wert einen Pixel:

$$\begin{aligned} x &= \frac{c}{upper} \cdot dx + sx \\ y &= \frac{c}{upper} \cdot dy + sy \end{aligned}$$

5. Stellen Sie sicher, dass Sie eine Fließkommadivision verwenden. Hierfür müssen Sie einen `cast` verwenden. Die Syntax sieht wie folgt aus: Wert $\rightarrow$ (Typ) $Wert$

   Beispiel : `int i = (int) 3.1415;` Diese Programmzeile konvertiert die Fließkommazahl in eine Ganzzahl.