Scheinaufgabe 1 (15%): Stack, Heap, Global (Thema: „storage duration“)?
Welche Zahlen (gelb hervorgehoben) liegen in welchem Speicher (ein “x” pro Spalte)?

	1	2	3	4	5	6
Stack			X	X		
Heap		X			X	X
Global	X					

```
#include <iostream>
#include <memory>

int a{1};
int *b=nullptr;

void f() {
     b = new int{2};
     int c{3};

     std::cout << a << "\n";
     std::cout << *b << "\n";
     std::cout << c << "\n";
}

int main() {
     f();
     int d{4};
     std::shared_ptr<int> e = std::make_shared<int>(5);
     std::unique_ptr<int> f = std::make_unique<int>(6);
     std::cout << d << "\n";
     std::cout << *e << "\n";
     std::cout << *f << "\n";
     delete b;
}
```

Scheinaufgabe 2 (10%): Was läuft hier falsch (Thema: „storage duration“)?

```
#include <iostream>
int *cnt;
void f() { int i{99}; cnt = &i; } // On my computer:
int main() {                      // (may differ on your computer)
     f();
     std::cout << *cnt << "\n";   // → 99
     std::cout << *cnt << "\n";   // → 1 ?????!?!?!?
}
```

Die Funktion f() wird aufgerufen, in der die LOKALE Variable i mit dem Wert 99 initialisiert wird.
Da es sich um eine LOKALE Variable handelt, wird dieser auf dem Stack gespeichert bis zur Ausführung der Funktion. Nach dem Verlassen, wird sein Speicherbereich als löschbar markiert und dieser kann eine Weile noch den Wert 99 beinhalten, es wird jedoch nicht garantiert.

Scheinaufgabe 3 (15%): Klausuraufgabe (Thema: „storage duration“)

```
#include <iostream>
#include <memory>

struct Point {
  float x,y;
  Point(float _x, float _y) {
    x=_x; y=_y;
  }
  ~Point() {
    std::cout << "destroy " << x << "," << y << "\n";
  }
};

struct A {
  Point *p;
};
struct B {
  std::shared_ptr<Point> p;
};
struct C {
  std::unique_ptr<Point> p;
};

int main() {
  A a1,a2;    B b1,b2;    C c1,c2;

  a1.p = new Point(11,11);                //                11
  a2.p = new Point(12,12);                //             12  |
  b1.p = std::make_shared<Point>(21,21); //          21  |  |
  b2.p = std::make_shared<Point>(22,22); //       22  |  |  |
  c1.p = std::make_unique<Point>(31,31); //    31  |  |  |  |
  c2.p = std::make_unique<Point>(32,32); // 32  |  |  |  |  |
                                          //  |  |  |  |  |  |
  a1.p = a2.p;                            // [ ][ ][ ][ ][ ][ ]
  b1.p = b2.p;                            // [ ][ ][ ][X][ ][ ]
  // c1.p = c2.p;  // kompiliert nicht!
}                                         // [X][X][X][ ][X][X]
```

Es werden sechs Punkte („Point“-Objekte) auf dem Heap erzeugt. Wann werden diese freigegeben? Kreuzen Sie das Kästchen in der Zeile an, in der der entsprechende Punkt (Koordinate 11,12,21,22,31,32) freigegeben wird.

Warum kompiliert die letzte (auskommentierte) Zeile nicht (siehe Code)?

c1 und c2 sind Unique Pointer, die nur einen einzigen Besitzer haben dürfen, d.h. diese 2 Pointer dürfen nicht auf das selbe Objekt zeigen. Im Bsp. ist c2.p nicht kopierbar!

_____ relevante Foliennummer aus 03_slides_classes_and_objects1.pdf: 30

Scheinaufgabe 4 (60%): Programmieraufgabe (Abgabe: Code)

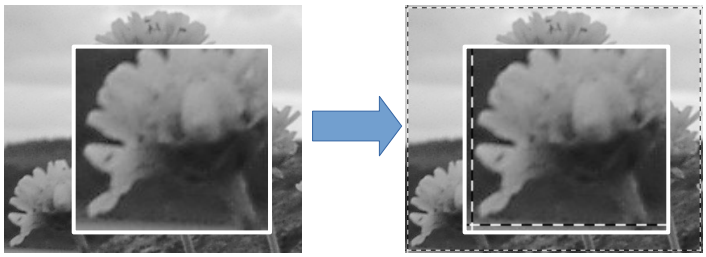
In dieser Übung soll die Bibliothek https://github.com/nothings/stb verwendet werden um PNG Bilder zu laden und zu speichern. Die Verwendung ist beschrieben in den Headerdateien

- https://github.com/nothings/stb/blob/master/stb_image.h
- https://github.com/nothings/stb/blob/master/stb_image_write.h

oder alternativ in den **Beispielarchiven stb_image_simple1.zip bzw. stb_image_simple2.zip**.

❗ Sie können (wie in stb_image_simple2.zip) die relevanten Dateien der Bibliothek einfach in Ihr Projekt kopieren, oder (wie in stb_image_simple1.zip) die Dateien installieren (beispielsweise mit https://github.com/microsoft/vcpkg) und von CMake finden lassen.

Die Bibliothek bietet eine Lesefunktion für Bilddateien (**stbi_load**) und eine Schreibfunktion (**stbi_write_png**). Bilder werden wie im Foliensatz 02_slides_hello_world.pdf beschrieben als 1D Array dargestellt (Folie 34 „Dynamic memory management: alloc/access/free array data“). Speicher, den die Bibliothek anlegt, muss mit „**stbi_image_free**“ freigegeben werden (statt mit delete). Will man Speicher im selben Stil anlegen, verwendet man „**stbi__malloc**“ (statt new; vgl. **stb_image_simple*.zip**). Das Beispiel malt einen gestrichelten Rahmen in das Bild.



Aufgabe:
Implementieren Sie eine Klasse „Image“ mit folgenden Eigenschaften. Sie soll den angegebenen Beispielcode in **stb_image_exercise4.zip** ermöglichen.

- **Information Hiding**: Die Klasse Image soll die Details der „stb“-Bibliothek verbergen (insbesondere den Pointer auf die Pixeldaten und die Bildgröße).
- Die Klasse Image soll einen **Konstruktor** anbieten, um ein PNG Bild als **Grauwertbild** zu **laden**.
- Die Klasse Image soll einen **Konstruktor** anbieten, um **Grauwertbild** einer bestimmten Größe **anzulegen**.
- Die Klasse soll im **Destruktor** angelegten **Speicher** wieder **freigeben**.
- Ein Image Objekt soll eine ***effiziente* Zugriffsfunktion** „get“ anbieten (zero-overhead durch Funktionsaufruf), um **Grauwerte** an einer bestimmten Koordinate **auszulesen**.
- Ein Image Objekt soll eine ***effiziente* Zugriffsfunktion** „set“ anbieten (zero-overhead durch Funktionsaufruf), um einen **Grauwert** an einer bestimmten Koordinate zu **setzen**.
- Ein Image Objekt soll es erlauben, die **Bildgröße abzufragen** („getWidth“, „getHeight“).
- Ein Image Objekt soll eine Funktion zum **Speichern von Bildern** (write_png) anbieten.
- Die Klasse wird von vorgegebenen Algorithmen verwendet (achten Sie dabei auf „const“-Parameter dieser Algorithmen). Achten Sie auch auf **const-correctness**!
- Optional: Was ist die Bedeutung von „`STB_IMAGE_IMPLEMENTATION`“ und „`STB_IMAGE_WRITE_IMPLEMENTATION`“ im Code?

❗ Hinweis: Der Vorgabe Code (siehe unten) ist in der Datei **stb_image_exercise4.zip** zu finden. Dort wird davon ausgegangen, dass die „stb“ Bibliothek installiert ist (wie in stb_image_simple1.zip). Sie können auch wie in stb_image_simple2.zip vorgehen, und die entsprechenden „stb“-Dateien Dateien in Ihr Projekt kopieren.

```
#include <iostream>
#include <cassert>

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"
```

```
class Image {
	int c,w,h;
	stbi_uc* data;
	// TODO
};

/**
* @brief simple "draw border" example.
* @param inp: input image (read only)
* @param b: border position
* @param d: dash size
* @param out: output image (must have same size as input image)
*/
void border(const Image& inp, int b, int d, Image &out) {
	assert(inp.getWidth() == out.getWidth());
	assert(inp.getHeight() == out.getHeight());
	for (int y=0;y<inp.getHeight();y++) {
		for (int x=0;x<inp.getWidth();x++) {
			if (x==b || y==b
				|| x==inp.getWidth()-b-1
				|| y==inp.getHeight()-b-1) {
				out.set(x,y, 255*(((x+y)/d)%2) );
			}
			else {
				out.set(x,y, inp.get(x,y));
			}
		}
	}
}

/**
* @brief trivial box filter implementation.
* @param inp: input image (read only)
* @param b: blur size
* @param out: output image (must have same size as input image)
*/
void blur(const Image& inp, int b, Image &out) { // Eingabe = const, Aausgabe = read/write
	assert(inp.getWidth() == out.getWidth());
	assert(inp.getHeight() == out.getHeight());
	for (int y=0;y<inp.getHeight();y++) {
		for (int x=0;x<inp.getWidth();x++) {
			int x1 = x-b/2;
			int x2 = x1+b;
			int y1 = y-b/2;
			int y2 = y1+b;
			if (x1<0) x1=0;
			if (y1<0) y1=0;
			if (x2>=inp.getWidth()) x2=inp.getWidth();
			if (y2>=inp.getHeight()) y2=inp.getHeight();
			int sum=0;
			int total=0;
			for(int yy=y1;yy<y2;yy++) {
				for(int xx=x1;xx<x2;xx++) {
					sum += inp.get(xx,yy);
					total++;
				}
			}
			out.set(x,y, sum / total);
		}
	}
}

int main() {
	Image inp{ "../blume2.png" }; // Vorsicht beim Pfad! Das müssen Sie ggf. anpassen...
	Image out{ inp.getWidth(), inp.getHeight() };
	border(inp, 5, 1, out);
	out.write_png("output_border_01.png");
	border(inp, 5, 6, out);
	out.write_png("output_border_06.png");
	border(inp, 5, 24, out);
	out.write_png("output_border_24.png");
	blur(inp, 5, out);
	out.write_png("output_blur_05.png");
	blur(inp, 15, out);
	out.write_png("output_blur_15.png");
}
```