1 Mutex-Guard

In dieser Aufgabe sollen Sie das Konzept RAII an einem einfachen Beispiel anwenden. Und zwar soll eine kleine Klasse geschrieben werden, welche den Destruktor ausnutzt, um automatisch einen Mutex zu unlocken. Dies ist sehr sinnvoll, da häufig Deadlocks entstehen, weil Entwickler nicht in allen Fällen einen gelockten Mutex freigeben. Z.B. beim vorzeitigen `return;` in einer Funktion.

1.1 Aufgabe

a) Schreiben Sie eine Klasse `MutexGuard` welche den folgenden `Mutex` automatisch freigeben kann.
b) Schreiben Sie einen Unit-Test welcher das Verhalten testet.

mutex.h

```
1  #pragma once
2
3  struct Mutex {
4    void lock() {
5      locked = true;
6    }
7    void unlock() {
8      locked = false;
9    }
10   bool locked;
11 };
```

2 Punkte-Array

Sie sollen eine eigene C++ Klasse zur Verwaltung eines halb-dynamischen Punkte-Arrays entwickeln. Das Attribut `size_` gibt die aktuelle Anzahl Punkt-Objekte im Array an und `capacity_` zeigt an, wie viele Punkt-Objekte maximal im Array Platz finden.

Die Klasse `Point` und das Interface der Klasse `PointArray` ist nachfolgend gegeben:

point.h

```
1  #pragma once
2
3  class Point {
4   public:
5    explicit Point(int x = 0, int y = 0) {
6      x_ = x;
```

```
    y_ = y;
  }

  int GetX() const { return x_; }
  int GetY() const { return y_; }

  void SetX(const int x) { x_ = x; }
  void SetY(const int y) { y_ = y; }

 private:
  int x_;
  int y_;
};
```

point_array.h

```
#pragma once

#include <cstddef>

#include "point.h"

class PointArray {
 public:
  PointArray();
  PointArray(Point const * const points, const size_t size);
  PointArray(const PointArray& other);
  ~PointArray();

  void Clear();
  size_t Size() const { return size_; }

  void Print() const;

  void PushBack(const Point& p);
  void Insert(const size_t pos, const Point& p);
  void Remove(const size_t pos);

  bool Get(const size_t pos, Point* p) const;
  Point* At(const size_t pos);
  const Point* At(const size_t pos) const;

 private:
  void Resize(size_t capacity);

  size_t size_;
  size_t capacity_;
  Point* points_;
};
```

2.1 Aufgabe

a) Implementieren Sie `PointArray`.
b) Schreiben Sie Unit-Tests um die Klasse zu testen. Sie können sich an folgendem Testprogram orientieren. Sinnvoll wäre z.B. alle hier gezeigten Test-Fälle in einzelne Unit-Tests zu migrieren und eventuell noch weitere Tests hinzuzufügen. Folgend das existerende Testprogramm.

```cpp
#include <iostream>

#include "point_array.h"

int main() {
  Point p1(1, 2);
  Point p2(2, 3);
  Point p3(3, 4);
  Point p4(4, 5);

  static constexpr size_t kArraySize = 4;
  Point points[kArraySize] = {p1, p2, p3, p4};

  PointArray pa1;
  PointArray pa2(points, kArraySize);
  PointArray pa3(pa2);
  pa2.Clear();

  std::cout << "size test" << std::endl;
  std::cout << std::boolalpha << (pa1.Size() == 0) << std::endl;
  std::cout << std::boolalpha << (pa2.Size() == 0) << std::endl;
  std::cout << std::boolalpha << (pa3.Size() == 4) << std::endl;

  std::cout << "push-back test" << std::endl;
  pa3.PushBack(Point(5, 6));
  pa3.Print();

  std::cout << "remove test" << std::endl;
  pa3.Remove(5);
  pa3.Remove(4);
  pa3.Remove(0);
  pa3.Remove(1);
  pa3.Print();

  std::cout << "insert test" << std::endl;
  pa3.Insert(0, p1);
  pa3.Insert(2, p3);
```

```
38    pa3.Insert(4, Point(5, 6));
39    pa3.Print();
40  }
```