Hinweis:
Die Anmeldung zur Klausur ist nun im Müsli freigeschaltet und für zwei Wochen bis zum **Donnerstag, den 30.01.2020** offen. Bitte melden Sie sich zur Teilnahme an der Klausur fristgerecht an.

Übung 1 *Polynomschablone*

In der Vorlesung (Kapitel 9) wurde die Klasse `Polynomial` durch Vererbung aus `SimpleFloatArray` erzeugt. Man kann diese Konstruktion durch die Verwendung von Templates vom unterliegenden Datentyp `float` lösen und dadurch flexibler machen; dadurch wird es zum Beispiel möglich `double`-Arithmetik zu verwenden. Grundlage der neuen Hierarchie ist die Template-Klasse `SimpleArray` aus Kapitel 11 der Vorlesung. Davon kann man eine templatisierte Klasse `Polynomial` ableiten. Die Klassendefinition sieht dann so aus:

```
template <class T>
class Polynomial :
  public SimpleArray<T>
{
public:
  Polynomial (int n);
  // ...
};
```

In den C++-Beispielprogrammen finden Sie das Hauptprogramm `UsePolynomial.cc`, welches die entsprechenden Klassen zur Realisierung von `Polynomial` aus `SimpleFloatArray` beinhaltet. Anhand dieser Vorlage verallgemeinern Sie die Klasse `Polynomial`, sodass es eine Schablone ist, die für verschiedene unterliegende numerische Datentypen funktioniert. Testen Sie Ihr Programm für mindestens zwei verschiedene Datentypen, z.B. `double` und `float`, wie in diesem Programmfragment:

```
Polynomial<float>  p(2);
p[0] = 1.0;
p[1] = 1.0;
p.print();
p = p*p;
p.print();

Polynomial<double> q(3);
q[0] = 2.0;
q[1] = -1.0;
q[3] = 4.0;
q.print();
```

(5 Punkte)

Übung 2 *Die Standard Template Library (STL)*

Die C++ Standard Library, die in der Regel mit dem Akronym ihrer Vorgängerin STL bezeichnet wird, stellt Container und Algorithmen zur Verfügung. Fast jedes größere Programm profitiert von der Verwendung der STL. Nicht nur erspart man sich auf diese Weise viel Programmieraufwand,

weil die abstrakten Datentypen nicht selbst implementiert werden müssen, man reduziert auch die Wahrscheinlichkeit, subtile Bugs zu produzieren.

Schreiben Sie eine Klasse für einen Vektor im mathematischen Sinne. Den Typ der Elemente wie die Länge des Vektors sollen als Templateparameter übergeben werden. Zur Verwaltung der Einträge wird ein `std::vector` innerhalb der Klasse verwendet. Die Klasse soll Methoden bereit stellen, die die folgenden Operationen erlauben:

- Addition zweier Vektoren
- Multiplikation mit einem Skalar
- Skalarprodukt
- Anwendung einer beliebigen Funktion (als Funktor) auf die Einträge
- Berechnung von Maximum, Minimum und Mittelwert

Das genaue Interface ist Ihnen überlassen. Rechenoperationen wie die skalare Multiplikation oder das Skalarprodukt sollten auch mit numerischen Datentypen funktionieren, die nicht dem Typ der Elemente des Vektors entsprechen. Dies können Sie programmieren, indem Sie die Methoden der Klasse wie Funktionen mit Templateparameter versehen. Am Beispiel des Skalarprodukts (als Methode einer solchen Klasse) kann der Code dafür wie folgt aussehen:

```
template<class K2>
K operator*(const Vector<K2,n>& y)
{
  K res(0.0);
  // ... Code hier
  return res;
}
```

Das Skalarprodukt zweier Vektoren $x \in (K)^n$, $y \in (K2)^n$ kann dann mit `x*y` berechnet werden, wobei zu beachten gilt, dass hier der überladene `operator*` nur für Vektoren gleicher Länge gegeben ist.
Implementieren Sie die oben genannten Methoden und verwenden Sie dabei

- den Elementzugriff über `operator[]`, durch den sich der `std::vector` wie ein C-Array verwenden lässt
- den Zugriff über die Containeriteratoren, der auch mit anderen Containern (z.B. `std::list`) funktioniert
- die vordefinierten STL-Algorithmen, die ebenfalls mit anderen Containern funktionieren

Schreiben Sie außerdem eine Main-Funktion, die Ihre Klasse testet.

Die in der STL definierten Container und die Komplexität ihrer Methoden können Sie z.B. unter `http://www.cplusplus.com/reference/stl/` finden, während die Algorithmen unter `http://www.cplusplus.com/reference/algorithm/` stehen.

(10 Punkte)

Übung 3 *Funktoren und statischer Polymorphismus*

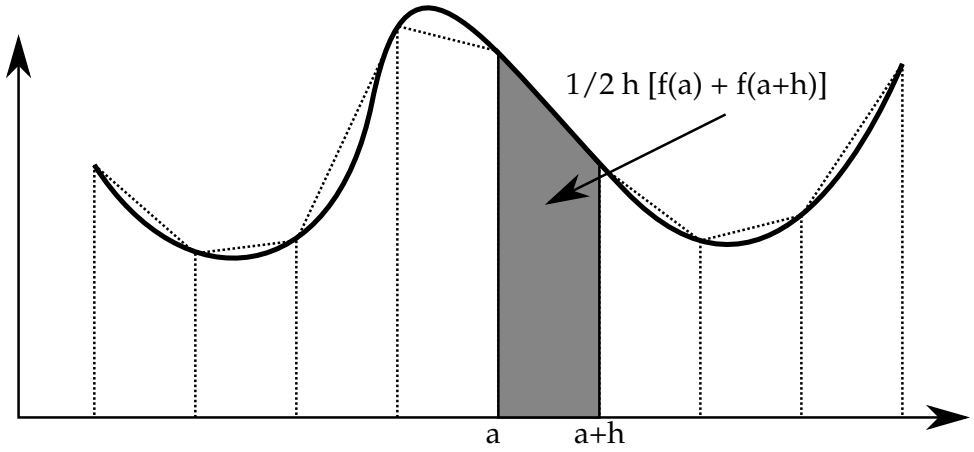
Funktoren sind, wie in der Vorlesung bereits definiert, Klassen, deren `operator()` überladen ist. Sie verhalten sich damit effektiv, wie Funktionen, haben jedoch durch ihre privaten Mitglieder ein "Gedächtnis". Betrachten Sie das folgende Beispiel eines Funktors:

```
class Wurzel
{
public :
  double operator()(double x)
  {
    return std::sqrt(x);
  }
};

int main()
{
  Wurzel f;
  double y = f(3.0); // an dieser Stelle wird Wurzel::operator()(3.0) ausgewertet!
  return 0;
}
```

In der numerischen Mathematik nähert man Integrale mittels sogenannter Quadraturformeln. Dabei wird das Integrationsintervall $[a, b]$ in n gleich große Intervalle unterteilt und die Funktion auf jedem dieser Intervalle durch ein Polynom angenähert. Für $n \to \infty$ konvergiert dieser Wert unter geeigneten Annahmen an f gegen das Integral. Man erhält ein Näherungsverfahren, indem man n gerade so groß wählt, dass der Fehler im Integral klein genug wird. Verwendet man lineare Polynome ergibt sich die sogenannte "summierte Trapezregel" (dabei ist die Intervalllänge $h = (b - a)/n$):

$$I_n(f; a, b) = \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{n-1} f(a + ih) + f(b)\right)$$



a) Schreiben Sie eine Funktion `trapezregel`, welche obige Formel implementiert. Neben der Anzahl an Intervallen `n` und den Integrationsgrenzen `a` und `b` muss dieser Funktion auch die zu integrierende Funktion übergeben werden. Da man die Funktion `trapezregel` für beliebige Funktionen verwenden will, muss man hierfür eine Technik des Polymorphismus verwenden. Im Falle des dynamischen Polymorphismus würde man eine abstrakte Basisklasse `Funktion` einführen und der Funktion `trapezregel` eine Referenz auf ein Objekt dieser Basisklasse übergeben. Wir wollen das Problem allerdings mittels statischem Polymorphismus lösen. Dazu realisieren wir unsere Funktion als Funktor und übergeben diesen an die Funktion. Der exakte Typ des Funktors ist dabei ein Templateparameter der Methode `trapezregel`. Testen Sie Ihre Funktion mit $f(x) = \sqrt{x}$ als Beispielfunktor und integrieren Sie diese numerisch über das Intervall $(0, \frac{3}{2})$ für verschiedene n. **[(3 Teilpunkte)]**

b) Nennen Sie Vor- und Nachteile der Verwendung von statischem Polymorphismus gegenüber dynamischem Polymorphismus. **[(2 Teilpunkte)]**

(5 Punkte)