Institut für Numerische und Angewandte Mathematik
FB Mathematik und Informatik der Universität Münster
Prof. Dr. Christian Engwer, MSc Marcel Koch

16.10.2019

Übung zur Vorlesung

Wissenschaftliches Rechnen

WS 2019/20 — Blatt 0

Abgabe: Keine Abgabe (Anwesenheitsaufgaben)

Achtung: Achten Sie darauf, Ihre Programme ordentlich zu formatieren und gut zu kommentieren. Die Form wird mit in die Bewertung eingehen.

Aufgabe 1 (Wertebereich von Basisdatentypen)

Schreiben Sie ein Programm, welches eine positive ganze Zahl einliest und in einem `short int` speichert. Addieren Sie anschließend 1 auf die Zahl und geben Sie das Ergebnis aus.

(a) Was passiert, wenn Sie 32.767 eingeben?

(b) Was passiert, wenn Sie 65.535 eingeben?

(c) Was passiert, wenn Sie statt eines `short int` einen `unsigned short int` verwenden und die Eingaben wiederholen?

(d) Was passiert, wenn Sie `(long) int` verwenden und die Eingaben wiederholen?

(e) Was passiert jeweils, wenn Sie stattdessen -1 addieren und Sie 0 eingeben?

Hinweis: Für das Einlesen der Zahl können Sie sich an Aufgabe 3 orientieren.

Aufgabe 2 (Fehlersuche)

```
#include <iostream>

// summiert alle Zahlen im Intervall [a,b]
int summieren (int a, int b)
{
  int summe;
  for (int i = a; i <= b; i++)
  {
    int summe = summe + i;
  }
  return 0;
}

int main ()
{
  std::cout << summieren(1,10) << std::endl;
  return 0;
}
```

Das obige Programm soll alle Zahlen von 1 bis 10 aufsummieren. Obwohl es syntaktisch korrekt ist, rechnet es falsch. Der Code enthält drei Fehler. Finden und reparieren Sie diese.

Aufgabe 3 (Fehlersuche 2)

```
#include <iostream>

int main ()
{
  double l; double h;
  std::cout << "Geben Sie die Laenge ein: ";
  std::cin >> l;
  std::cout << "Geben Sie die Hoehe ein: ";
  std::cin >> h;

  std::cout << "Flaeche: " << flaeche(l,h) << std::endl;
  return 0;
}

double flaeche (double laenge, double hoehe)
{
  double ergebnis;
  ergebnis = laenge * hoehe;
  return ergebnis;
}
```

Das obige Programm soll die Fläche eines Rechtecks berechnen, kann jedoch nicht fehlerfrei übersetzt werden. Finden und korrigieren Sie den Fehler.

Aufgabe 4 (Vernünftige Formatierung des Code)

Welcher Unterschied besteht bei der Ausführung folgender C++ Codefragmente?

(a) `if (x>0.0) y=sqrt(x); z=x*x;`

(b) `if (x>0.0) {y=sqrt(x); z=x*x;}`

(c) `if (x>0.0) {y=sqrt(x);} z=x*x;`

(d) `if (x>0.0) {y=sqrt(x)}; z=x*x;`

Schreiben Sie die Ausdrücke so auf, dass der Ablauf der Ausführung deutlich wird!

Aufgabe 5 (Fakultät)

Schreiben Sie ein Programm, welches die Fakultät $n! = 1 \cdot 2 \cdot \ldots \cdot n$ für alle ganzen Zahlen $0 < n \leq 10$ berechnet und ausgibt.

- Verlagern Sie die Fakultätsberechnung in eine Prozedur
  `unsigned int fakultaet (unsigned int n)`
- Das Hauptprogramm soll diese Prozedur für die gewünschten Werte von n aufrufen

Hinweis: Die Prozedur `fakultaet` kann sowohl iterativ als auch rekusiv implementiert werden.

Aufgabe 6 (Primzahlen)

Schreiben Sie ein Programm das alle Primzahlen bis n sucht und ausgibt, wobei n eine vom Benutzer festzulegende natürliche Zahl ist.

- Überlegen Sie sich einen einfachen Algorithmus um zu testen, ob eine Zahl prim ist
- Führen Sie diesen Test für alle Zahlen bis n durch

Aufgabe 7 (Zeiger 1)

Mit den Definitionen

```
   int x = 25;   int y = -12;
int* p1 = &x; int* p2 = &y;
```

erläutere man die Unterschiede folgender Anweisungen:

(a) `p1 = p2;` und `*p1 = *p2;`

(b) `if (p1 == p2) { ... }` und `if (*p1 == *p2) { ... }`

(c) `if (p1) { ... }` und `if (*p1) { ... }`

Aufgabe 8 (Zeiger 2)

Nach Definition der Variablen `test` und `zeiger` durch:

```
int test = 51; int* zeiger = &test;
```

stehen folgende Inkrementierungsanweisungen zur Auswahl:

(a) `*zeiger++`
(b) `(*zeiger)++`
(c) `*(zeiger++)`
(d) `*(++zeiger)`
(e) `*++zeiger`
(f) `++(*zeiger)`
(g) `++*zeiger`

Sind alle syntaktisch korrekt? Erläutern Sie die Wirkungsweisen.

Aufgabe 9 (Zeiger 3)

Nach den Definitionen:

```
int i = 5;
int* pi; int* pj;
char* pc; char* pd;
```

sollen die folgenden Zuweisungen durchgeführt werden:

(a) `pi = i;`
(b) `pi = &i;`
(c) `*pi = i;`
(d) `*pi = &i;`
(e) `pi = pj;`
(f) `pc = &pd;`
(g) `pi = pc;`
(h) `pd = *pi;`
(i) `*pi = i**pc;`
(j) `pi = 0;`

Sind alle syntaktisch korrekt? Erläutern Sie die Wirkungsweisen bzw. nennen Sie die Werte der Variablen auf der rechten und linken Seite.

Aufgabe 10 (Klassen und Objekte)

Quadratische Funktionen in einer Variablen lassen sich allgemein durch folgende Normalform beschreiben:

$$f(x) = ax^2 + bx + c \quad \text{mit} \quad a, b, c, x \in \mathbb{R}.$$

(a) Schreiben Sie eine Klasse `QuadF` zur Repräsentation solcher quadratischer Funktionen. Dem Konstruktor der Klasse sollen dabei die drei Koeffizienten a, b und c als Parameter übergeben werden können, welche dann in Attributen eines von Ihnen sinnvoll zu wählenden Datentyps gespeichert werden. Zusätzlich zu dem Konstruktor soll die Klasse die drei Methoden `getA`, `getB` und `getC` zur Verfügung stellen, mit denen die Koeffizienten einer Funktion ausgelesen werden können. Achten Sie darauf, die Attribute der Klasse `QuadF` vor Zugriffen aus fremden Klassen zu schützen.

(b) Ergänzen Sie die Klasse `QuadF` um eine Methode `double evaluate (double x)`, die den Wert der Funktion für den übergebenen Parameter x berechnet und das Ergebnis zurückgibt.

(c) Jede der oben beschriebenen Funktionen (mit $a \neq 0$) besitzt eine Stelle, an der die Funktion einen Extremwert (Minimum oder Maximum) annimmt. Bestimmen Sie allgemein die x-Koordinate der Extremstelle einer solchen Funktion und fügen Sie der Klasse `QuadF` eine Methode `double getExtremePos ()` hinzu, welche diese zurückgibt.

(d) Schreiben Sie ein Hauptprogramm `TestQuadF`, dessem Prozedur `main` zwei unterschiedliche Instanzen der Klasse `QuadF` angelegt und die Koeffizienten, die Extremstelle sowie den zugehörigen Extremwert der Funktionen auf der Konsole ausgibt.

Aufgabe 11 (Abstrakte Klassen, Vererbung, Virtuelle Methoden)

(a) Schreiben Sie ein eine abstrakte Klasse `Expression` zur Repräsentation arithmetischer Ausdrücke in C++. Die Klasse soll eine abstrakte Methode `evaluate ()` definieren, die zur Berechnung des Wertes eines arithmetischen Ausdrucks dient und als Ergebnis eine Zahl von Typ `double` zurückgibt. Außerdem soll die Klasse eine Methode `int compareTo (const Expression& exp2)` enthalten, die den Wert des Ausdrucks mit dem des übergebenen Ausdrucks vergleicht und -1, 0 oder 1 zurückgibt, wenn der Wert kleiner, gleich oder größer als der Wert von `exp2` ist.

(b) Implementieren Sie die konkreten Klassen `Constant`, `Sum` und `Product` als Unterklassen der Klasse Expression. Die Klasse `Constant` dient dabei der Darstellung konstanter Zahlen vom Typ `double`, die vom Konstruktor in ein Attribut `value` gespeichert werden. Die Konstruktoren der Klassen `Sum` und `Product` zur Realisierung von Summen bzw. Produkten sollen jeweils zwei Objekte vom Typ `const Expression&` entgegennehmen und in geeigneten, vor Zugriffen von außen gekapselten Attributen speichern. Natürlich ist die Methode `evaluate ()` in allen drei Klassen auf sinnvolle Weise zu implementieren.

(c) Schreiben Sie ein Testprogramm `ExpressionTest`, welches verschiedene arithmetische Ausdrücke erzeugt und sämtliche Methoden der Klassen auf sinnvolle Weise überprüft. Bei den Tests sollen mindestens die folgenden Ausdrücke verwendet werden:

$$((5.5 * 7.5) + 0.75), \quad (3 * (2 + 9)) \quad \text{und} \quad ((7 - 1) * (4.3 + 2.7))$$

(d) Jeder arithmetische Ausdruck soll mit Hilfe einer Methode

```
std::string toString ()
```

in einer lesbaren Weise dargestellt werden können. Überlegen Sie sich, wie solch eine Methode sinnvollerweise implementiert werden kann (welche Rolle spielt insbesondere die Klasse `Expression`), implementieren und testen Sie die Methode. Achten Sie auf eine korrekte Klammerung der Ausdrücke, die möglicherweise auch noch sinnvoller sein kann als jene in Aufgabenstellung (c).

Aufgabe 12 (Templates)

Kopieren Sie Ihren Code aus Aufgabe 11 und modifizieren Sie die Kopie derart, dass die Bestandteile eines arithmetischen Ausdrucks auch durch andere Datentypen repräsentiert werden können als `double`.

(a) Realisieren Sie analog zu Aufgabe 11 (a), (b) und (d) ein abstraktes Klassentemplate `Expression` und davon abgeleitete Klassentemplates `Constant`, `Sum` und `Product` in C++.

(b) Implementieren Sie ein Aufgabe 11 (c) entsprechendes Testprogramm `ExpressionTest`. Verwenden Sie für die Bestandteile der arithmetischen Ausdrücke dabei nun passendere Datentypen.

Aufgabe 13 (Noch mehr Templates)

Schreiben Sie eine templatisierte Klasse, die ein Polynom mit festem maximalen Polynomgrad darstellt. Die Koeffizienten des Polynoms sollen als `std::array` gespeichert werden. Die Klasse soll folgenede Funktionalität enthalten:

- einen Konstruktor der Objekte vom Typen `std::array` akzeptiert
- die Operatoren $+/-$ und $+=/-=$,
- den Operator $\cdot$, der eine Multiplikation mit einem Skalar darstellt,
- die Funktion `evaluate` wie in Aufgabe 10.

Schreiben Sie ein Testprogramm, das die Funktionalität Ihrer Klasse überprüft.

Aufgabe 14 (Interfaces und Templates: Vorbereitung zu Blatt 1)

Sei $g_1 : \mathbb{R} \to \mathbb{R}$ integrierbar auf $[0,1]$ und $g_2 : \mathbb{R}^2 \to \mathbb{R}$ eine auf $[0,1]^2$ integrierbare Funktion. Aus der eindimensionalen Gauß-Quadraturformel mit zwei Stützstellen für das Intervall $[0,1]$

$$Q_2[g_1] := \frac{1}{2} g_1(s_1) + \frac{1}{2} g_1(s_2), \quad s_1 := \frac{1}{2} - \frac{\sqrt{3}}{6}, \quad s_2 := \frac{1}{2} + \frac{\sqrt{3}}{6},$$

ergibt sich folgende zweidimensionale Gauß-Quadraturformel für das Einheitsquadrat $[0,1]^2$:

$$Q_2^{(2)}[g_2] := \frac{1}{4} g_2(s_1, s_1) + \frac{1}{4} g_2(s_1, s_2) + \frac{1}{4} g_2(s_2, s_1) + \frac{1}{4} g_2(s_2, s_2)$$

Sie ist für alle Polynomfunktionen exakt, die aus Monomen $x_1^{k_1} x_2^{k_2}$ mit $0 \leq k_1, k_2 \leq 3$ bestehen.

Sei $f : \mathbb{R} \times \mathbb{R}^2 \to \mathbb{R}$, $(t, x) \mapsto f(t, x)$, eine Funktion mit $f(t, \cdot) : \mathbb{R}^2 \to \mathbb{R}$ ist für alle $t \in \mathbb{R}$ eine auf $[0, 1]^2$ integrierbare Funktion. Dann lässt sich das Integral

$$I[f](t) := \int_0^1 \int_0^1 f(t, x_1, x_2)\, dx_1\, dx_2 \qquad \text{mit} \quad (x_1, x_2) = x \tag{1}$$

zu jedem Zeitpunkt durch $I[f](t) \approx Q_2^{(2)}[f(t, \cdot)]$ approximieren.

(a) Definieren Sie den Typen von f als Template-Alias von `std::function`.

(b) Implementieren Sie ein Klassentemplate `UnitSquareIntegrator` zur approximativen Berechnung des Integrals (1) zu verschiedenen Zeitpunkten. Das Verfahren soll für verschiedene Datentypen anwendbar sein, mit denen sich Vektoren in $\mathbb{R}^2$ und das Ergebnis sowie Zeitpunkte in $\mathbb{R}$ repräsentieren lassen. Zu diesem Zweck soll `UnitSquareIntegrator` drei Template-Parameter `VectorType`, `ResultType` und `TimeType` besitzen. Setzen Sie voraus, dass `VectorType` eine Methode `operator[]` für den indexbasierten Zugriff auf die Elemente des Vektors und eine Methode `size` zu Verfügung stellt, wie es viele Container der C++-Standardbibliothek machen (siehe z.B. `std::array`).

Der Konstruktor des Klassentemplates soll die Funktion f erhalten.

(c) Die Methode `ResultType integrate (const TimeType& t) const` soll obiges Integral zu einem gegebenen Zeitpunkt `t` gemäß obiger Quadraturformel berechnen und den Wert zurückgeben.

(d) Testen Sie `UnitSquareIntegrator` anhand von Funktionen ihrer Wahl. Beachten Sie dabei die Aussage zur Exaktheit für bestimmte Polynomfunktionen.

Wählen Sie als Template-Parameter `VectorType` den Datentyp `std::array` und für die Komponenten des Vektors sowie als `ResultType` und `TimeType` den Datentyp `double`.