Bachelorstudiengang „Scientific Programming“ / MATSE Ausbildung

FH Aachen Ausbildungsstandorte Jülich, Köln, Forschungszentrum Jülich
Rechen- und Kommunikationszentrum der RWTH Aachen

“C++”

Prüfung 02.07.2014

Prof. Dr. Alexander Voß

Name	
Vorname	
Matrikelnr.	
Sticknr./Login	
Unterschrift	

	Punkte maximal	**Punkte erreicht**	**Korrigiert durch**
Aufgabe 1	**12**		
Aufgabe 2	**32**		
Aufgabe 3	**56**		
Gesamtpunkte	**100**		
Note			

Allgemeine Randbedingungen und Anforderungen

- **Es gibt drei elektronische Aufgaben A1, A2, A3.**
- **Stick**
  - Sie erhalten auf Ihrem Stick eine cpp-Datei sowie ein makefile.
  - Die cpp-Datei enthält ein nicht funktionsfähiges, aber compilierbares und lauffähiges Grundgerüst, in dem Sie je nach Aufgabe Code implementieren und einkommentieren.
  - Das makefile dient wie üblich der Erstellung.
- **Abgabe**
  - **Sie geben nur die cpp-Datei ab, indem Sie diese Datei auf den Stick kopieren.** Optional laden Sie sie in den entsprechenden Ordner im Ilias hoch.
    (Ilias: A.Voß, C++, Klausurabgabe 2.7.)
- **Start**
  - **Füllen Sie zuerst das Deckblatt aus und vergessen Sie nicht zu unterschreiben und die Sticknummer anzugeben.**
  - **Die oben genannten Dateien kopieren Sie vom Stick in ein extra eingerichtetes Arbeitsverzeichnis.**
  - **Benennen Sie die cpp-Datei mit dem Namen cpp_<MATRNR>_<NAME> um und ersetzen Sie "<MATRNR>" durch Ihre Matrikelnummer und "<NAME>" durch Ihren Namen (ohne Umlaute).**
  - **In der cpp-Datei geben Sie bitte Ihren Namen und die Matrikelnummer im Kopf der Datei an (Konstanten `matse_name, matse_matrnr`).**
- **Prüfung**
  - **Beachten Sie: Jegliche Kommunikation während der Prüfung ist nicht erlaubt!**
  - Bei jeder Aufgabe gibt es Abzüge, wenn das Programm nicht compiliert, wenn der Code grob ineffizient, kryptisch oder umständlich bzw. unverständlich ist oder wenn Sie einen erfolgreichen Tipp bekommen haben.

Viel Erfolg!

Aufgabe A1

In dieser Aufgabe geht es darum, eine numerische Näherung eines Integrals einer Funktion zu bestimmen (Numerische Quadratur). Das Integral soll nach folgender Tangententrapezformel Q(f) für eine Funktion f auf einem Intervall [a,b] berechnet werden:

$$Q(f) = (b-a)\ f\left(\frac{a+b}{2}\right)$$

Implementieren Sie folgende Anforderungen:

a) Die Funktion `Quad(a,b)` berechnet für die vorgegebene Funktion `f` mit f(x)=x+1 eine Näherung nach obiger Tangententrapezformel auf dem Intervall [a,b] und gibt diesen Näherungswert zurück.

b) Die um die Angabe einer Funktion erweiterte Funktion `Quad(a,b,f)` berechnet analog zu a) eine Näherung des Integrals für eine beliebige Funktion f.
Geben Sie eine geeignete Signatur und Implementierung für `Quad(a,b,f)` an, so dass die Aufrufe `Quad(a,b,f)` in `Aufgabe1` möglich sind (Aufruf einkommentieren).

c) Die Funktion `Quad(a,b,f)` aus b) kann auch mit Lambda-Ausdrücken aufgerufen werden.
Definieren Sie in `Aufgabe1` die Funktion f(x)=x+1 als Lambda-Ausdruck und speichern ihn in der Variablen g, so dass die Aufrufe `Quad(a,b,g)` in `Aufgabe1` möglich sind (Aufruf einkommentieren).

Die korrekten Werte für alle drei Teilaufgaben lauten wie folgt:

```
Int_[a1,b1] f(x) dx = 8
Int_[a2,b2] f(x) dx = -2
Int_[a3,b3] f(x) dx = 16
```

Bewertungsschema			
Aufgabe	**Max. Punkte**	**Erreichte Punkte**	**Kommentar**
a)	**4**		
b)	**4**		
c)	**4**		
Abzüge			
Gesamt	**12**		

Aufgabe A2

In dieser Aufgabe soll die Berechnung einer Näherung eines Integrals für eine abgeschlossene Menge, gegeben als Menge von Intervallen der Form [a,b], parallelisiert werden. Der Aufruf der entsprechenden Funktion `Quad` für eine Menge von Intervallen (gegeben in einem `vector`) findet sich in der Funktion `Aufgabe2`.
Da die Näherungsberechnung prinzipiell für Datentypen unterschiedlicher Genauigkeit gültig ist, könnte man alle Funktionen, Klassen und Strukturen generisch anlegen, also als Templates. Hier sind die Anforderungen allerdings *nicht* generisch zu lösen sondern fest für den Datentyp `double` umzusetzen. Erst in Teilaufgabe d) wird ein Template *beispielhaft* angegeben.

Implementieren Sie folgende Anforderungen:

a) Die Struktur `Intervall` enthält zwei Member a und b vom Typ `double`. Eine leere Struktur finden Sie bereits vordefiniert.

b) `Intervall` besitzt einen Konstruktor, der zwei double-Argumente a und b bekommt und die gleichnamigen Member damit initialisiert.
Damit ist auch die Definition des Vektors `IVs` und der Aufruf von `Quad` in `Aufgabe2` möglich (Definition und Aufruf einkommentieren).

c) Die Funktion `Quad(const vector<Intervall>& IVs)` startet für jedes Intervall in der Menge `IVs` einen eigenen Thread und summiert thread-safe auf dem jeweiligen Intervall eine Näherung entsprechend Aufgabe 1. Die Funktion `Quad(a,b)` aus Aufgabe 1 soll benutzt werden.
Hierbei ist zu beachten:

   i. Das Feld von Threads wird dynamisch angelegt.
   ii. Alle Threads sollen Ihr Ergebnis in der lokalen Variable `sum` aufsummieren. Der Zugriff auf `sum` ist zu schützen.
   iii. Parameter an die Worker-Funktion `Work` werden als Referenzen übergeben.

   Unter Punktabzügen sind folgende Alternativen bei Problemen möglich:

   - Das Anlegen der Threads entsprechend der Anzahl von Intervallen in `IVs` kann in einem Thread-array *fester* Länge (hier 3) erfolgen, wenn das dynamische Anlegen Probleme bereitet.
   - Die Übergabe per Referenz kann durch Zeigerübergaben ersetzt werden.
   - Die eigentliche Berechnung mithilfe der Funktion `Quad(a,b)` aus Aufgabe 1 kann im Fehlerfall durch eine Dummy-Funktion ersetzt werden.

   Die korrekte Ausgabe lautet dann wie folgt:

```
Sum_i Int_[a_i,b_i] f(x) dx = 22
```

d) Die Struktur `IntervallGeneric` ist ein Beispiel-Template analog zu `Intervall`, definiert einen Typ `value_type` für den Template-Parameter T und zwei Member a und b vom Typ `value_type`.
Dieses Template `IntervallGeneric` soll nicht verwendet sondern nur angegeben werden.

C++ 02.07.2014
Prof. Dr. Alexander Voß

Bewertungsschema			
Aufgabe	**Max. Punkte**	**Erreichte Punkte**	**Kommentar**
a)	**4**		
b)	**4**		
c)	**20**		
d)	**4**		
Abzüge			
Gesamt	**32**		

Aufgabe A3

In dieser Aufgabe geht es darum, eine Klasse `AbgMenge` für eine abgeschlossene Menge zu schreiben, die eine Menge von Intervallen der Form [a,b] verwalten kann.

Diese Klasse besitzt eine Reihe von Eigenschaften, die Sie implementieren:

a) Die Klasse `AbgMenge` besitzt einen nicht-öffentlichen Member vom Typ `vector<Intervall>`, der die Intervalle speichert.

b) Die Klasse `AbgMenge` besitzt einen Default-Konstruktor.

c) Die Klasse `AbgMenge` besitzt einen Konstruktor, der ein Argument vom Typ `Intervall` aus Aufgabe 2 bekommt und dieses Intervall speichert.

d) Die Klasse `AbgMenge` besitzt einen Copy-Konstruktor, der die Menge der Intervalle kopiert.

e) Die Klasse `AbgMenge` besitzt einen Konstruktor, der mit einer `initializer_list` aufgerufen werden kann. Die Intervalle sind immer so einsortiert, dass die Intervallanfangspunkte aufsteigend sortiert sind (siehe auch h) und i)).

Sind die Teilaufgaben a)..e) gelöst, können Sie die Definition der Variablen `I1, I2, I3` und `M1, M2, M3` in `Aufgabe3` einkommentieren.

f) Die Struktur `Intervall` und die Klasse `AbgMenge` besitzen jeweils einen Ausgabeoperator, der folgende Ausgabe erzeugt (Ausgabe in `Aufgabe3` einkommentieren):

```
I1=[2,4], I2=[-3,-1], I3=[6,8]
M1={ } size=0
M2={ [2,4] } size=1
M3={ [-3,-1] [6,8] } size=2
```

g) Es existiert ein Index-Operator, der eine konstante Referenz auf das n'te Intervall zurückgibt. Ist der Parameter des Index-Operators unzulässig, wird eine Ausnahme geworfen. Ausgabe:

```
M3[1]=[6,8]
Indexfehler
```

h) Es existiert eine Funktion `add`, die ein Intervall übergeben bekommt und es in den Vektor der Intervalle so einsortiert, dass die Intervallanfänge aufsteigend sortiert sind.
Achtung: Sie brauchen keine Überlappungen zu berechnen oder Intervalle zu ersetzen, falls sie sich einschliessen etc. Ebenso sind hier doppelte Intervalle möglich.
Hinweis: In der Klasse `vector` existiert eine Funktion, die Sie bei Bedarf benutzen können: `insert(iterator,element)`.

i) Es existiert ein +-Operator, der zwei abgeschlossene Mengen zusammenfügen kann und eine neue Menge, bestehend aus allen Einzelmengen der Operatoren, zurück liefert. Die Intervallanfangspunkte sind aufsteigend sortiert.
Genau wie in Teilaufgabe h) müssen keine Überlappungen berechnet oder doppelte Intervalle eliminiert werden.
Wenn der Operator verfügbar ist, kann die Summe in `Aufgabe3` berechnet werden.

j) Über die Intervall eines Objekts der Klasse `AbgMenge` kann, wie in `Aufgabe3` zu sehen ist, iteriert werden (Ausgabe in `Ausgabe3` einkommentieren).

Ausgabe:

```
Ergebnis op+: M1={[-3,-1] [2,4] [6,8] }
```

Mit dieser Funktionalität an der Hand könnte jetzt eine parallele Berechnung von Integralnäherungen auf abgeschlossenen Mengen stattfinden.

Bewertungsschema			
Aufgabe	**Max. Punkte**	**Erreichte Punkte**	**Kommentar**
a)	**4**		
b)	**4**		
c)	**4**		
d)	**4**		
e)	**8**		
f)	**4**		
g)	**8**		
h)	**8**		
i)	**4**		
j)	**8**		
Abzüge			
Gesamt	**56**		