Aufgabenblatt 5 Templates

Aufgabe 1

Entwickeln Sie eine Template-Klasse Vielleicht. Objekte können einen Wert enthalten müssen aber nicht. Der Wert wird im Attribut wert gespeichert. Der Typ des Attributs wert ist parametrisiert. Entwickeln Sie die Funktionalität Schritt für Schritt. Nach jedem Aufgabenteil testen Sie ihr Template, indem Sie das Template für verschiedene Typen instanziieren.

Entwickeln Sie die folgende Funktionalität für die Template Klasse Vielleicht:

a) Konstruktor mit einem Argument: Weist dem Attribut wert einen Wert zu.
b) Konstruktor ohne Argument: Speichert keinen Wert. Erzeugt ein leeres Objekt (Objekt mit Zustand „leer“).
c) bool leer() const: Liefert true zurück, falls Objekt leer ist, d.h. nichts gespeichert ist, sonst false.
d) const T& derWert() const: Liefert Referenz auf den gespeicherten Wert vom Typ T zurück. Falls die Methode an einem leeren Objekt aufgerufen wird soll ein Fehler gemeldet werden und das Programm beendet werden. Hier können Sie auch eine Ausnahme werfen und behandeln, wenn Sie wissen wie es geht.
e) std::string text() const: Liefert eine Textdarstellung des Inhaltes als String zurück. (Wie in Java die toString Methode), also z.B. „leer“ oder für den Typ int die Ganzzahl formatiert als Text. Es gibt elegantere und weniger elegante Möglichkeiten der Implementierung. Eine weniger elegante Variante wäre eine Spezialisierung für jeden Typ. Eine andere Möglichkeit ist die Verwendung von std::ostringstream (wie cout, schreibt aber in einen String). Damit haben Sie alle Typen behandelt, die std::ostringstream kennt. Für weitere Typen können Sie operator<< überladen oder text() spezialisieren. Suchen Sie eine Code sparende, elegante Formulierung.

   Hinweis: in dem Attribut vom Typ T des Templates Vielleicht können Sie den Zustand leer nicht speichern. Sie brauchen also ein zweites Attribut.

f) Erweitern Sie die Funktionalität Ihre Klasse
   I. Schreiben Sie die operator+ Operation. Resultat ist ein Objekt, welches den Wert speichert, der dem Ergebnis der Addition der Attributwerte entspricht. Hinweis: Das funktioniert nur, wenn die + Operation für den instanziierten Typ T existiert. Operationen mit einem leeren Objekt als Operand liefern als Resultat ein leeres Objekt. Implementieren Sie die Operation als globale Funktion (friend).
   II. Schreiben Sie die operator/ Operation. Das Resultat ist ein Objekt, welches den Wert speichert, der dem Ergebnis der Division der Attributwerte entspricht. Operationen mit einem leeren Objekt als Operand liefern als Resultat ein leeres Objekt. Implementieren Sie die Operation als globale Funktion (friend).
g) Instanziieren Sie zuerst für den Typ int, d.h. testen Sie diese Instanz. Wenn Sie Probleme mit Templates haben, können Sie auch einen Schritt zurückgehen und zuerst die Klasse Vielleicht ohne Template für den Typ int implementieren und testen und dann eine Template Klasse daraus machen.
h) Instanziieren Sie für den Typ double. Demonstrieren Sie die Funktionalität.

i) Betrachten Sie die Instanziierung für den Typ std::string. Die Implementierung von operator/ für Vielleicht<T> verlangt eine definierte Division für T. Die Division ist für string jedoch nicht definiert. Die generische Implementierung von operator/ kann also für Vielleicht<std::string> nicht verwendet werden. Eine Lösungsmöglichkeit ist operator/ zu spezialisieren für Vielleicht<std::string> als Argumenttyp. Implementieren Sie die Spezialisierung von operator/ für Vielleicht<std::string> so, dass die Funktionen ein Vielleicht<int> Objekt zurück gibt mit der Differenz der Länge der Strings als Wert für das Attribut vom Typ int. Demonstrieren Sie die Funktionalität.
j) Instanziieren Sie Vielleicht<T> für einen Typ, den Sie selbst definiert haben und für den Sie die Operationen + und / implementiert haben. Hinweis: sinnvolle Möglichkeiten sind z.B. Komplex oder Bruch. Für komplexe Zahlen und Brüche sind die Operationen + und / definiert. Sie können einfach das Standardverhalten implementieren.

Aufgabe 2

Implementieren Sie eine Prozedur add(), die eine beliebige Anzahl von Argumenten haben darf. add() soll für einen beliebigen Datentyp funktionieren, für den die Addition implementiert ist (operator+)
Beispiele sind : add(int,int) oder add(string, string, string)
Verwenden Sie das Variadic Template aus C++11 für die Implementierung.
Demonstrieren Sie Ihr Template für:

a) int (Typ der Sprache C++)
b) string (Typ aus der Klassenbibliothek)
c) einen selbstdefinierten Typ für den Sie die Addition definiert haben (Bruch, Komplex …).

Hinweis: Implementieren Sie ein Template für eine globale Funktion! Implementieren Sie kein Klassen Template.

Die Programme sind in Felix abzugeben. Dokumentierten Sie die C++-Texte, z.B mit Kommentaren. Der Abgabetermin ist im Felix Kurs hinterlegt.