FH-OÖ Hagenberg/HSD
Software Entwurf 2, SS 2013
Übung 9

HSD FH-HAGENBERG

Name: Abgabetermin: KW 23

Mat.Nr: Punkte:

Übungsgruppe: korrigiert:

Aufwand in h:

Beispiel 1 (12 Punkte) Generischer Algorithmus mit Prädikat: Implementieren Sie einen generischen Algorithmus `combine_if`, der folgender Schnittstelle entspricht:

```
template<typename InputItor1, typename InputItor2, typename OutputItor,
         typename CombineOp, typename Pred>
void combine_if(InputItor1 first1, InputItor1 end1, InputItor2 first2,
                OutputItor res, CombineOp combOp, Pred p)
```

1. Der Algorithmus soll auf jedes Element der ersten Sequenz `[first1,end1)` und das korrespondierende Element (gleiche Position) der zweiten Sequenz, deren Anfang die Iteratorposition `first2` ist, gemeinsam die Kombinierungsoperation `CombineOp` (binäre Operation) anwenden und das Ergebnis der Operation auf den Ausgabe-Iterator `res` kopieren. Die Bedingung für die Kombination zweier Elemente ist die Erfüllung (für beide Elemente) des angegebenen Prädikates `Pred`.

2. Definieren Sie eine Template-Funktion `Print`, deren erster Parameter ein beliebiger STL-Container sein kann und der zweite Parameter ein Defaultparameter ist, dessen Defaultwert die leere Zeichenkette ist und als Kopfzeile vor den Elementen ausgegeben wird. Die Print-Funktion soll zur Ausgabe den `std::copy`-Algorithmus aus der STL verwenden!

3. Testen Sie den Algorithmus mit verschiedenen Containern (Liste,Vektor und Set) für den Elementtyp `int` und `float` und geben Sie die Ergebnisse auf der Standardausgabe `std::cout` aus. Als Kombinierungsoperation definieren Sie eine allgemeine Funktion `ProductSquares` die das Produkt der Quadrate zweier Werte beliebigen Datentyps ermittelt und das Ergebnis zurück gibt. Zusätzlich sollen nur positive Werte miteinander kombiniert werden. Definieren Sie dazu ein allgemeines Prädikat `IsPos`, welches für `float` und `int` verwendet werden kann.

Beispiel 2 (12 Punkte) Template-Klasse für binäre Suchbäume: Implementieren Sie eine Template-Klasse für binäre Suchbäume entsprechend der folgenden Schnittstelle:

```
template <typename TValue, typename TPred = std::less<TValue> >
class BinarySearchTree {

  struct TNode {
    TValue value;
    TNode *pLeft;
    TNode *pRight;
  };

public:

  BinarySearchTree();
  BinarySearchTree(BinarySearchTree<TValue, TPred> const &tree);
  BinarySearchTree<TValue, TPred> &operator=(BinarySearchTree<TValue, TPred> const &tree);
  ~BinarySearchTree();

  bool Insert(TValue const &value);  // returns false if already contained

  template <typename TVisitor> void VisitPreOrder(TVisitor visitor) const;
  template <typename TVisitor> void VisitInOrder(TVisitor visitor) const;
  template <typename TVisitor> void VisitPostOrder(TVisitor visitor) const;

private:
  TNode *pRoot;

  // ... helper methods ...
};
```

Ein Baumdurchlauf mit den Visit-Methoden soll z.B. folgendermaßen funktionieren:

```
void Print(int const x) { ... }
...
BinarySearchTree<int> tree;
..
tree.VisitInOrder(Print);
```

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung!** Verwenden Sie immer **Module**, um den Testtreiber und die eigentliche Implementierung zu trennen! Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit **Kommentaren! Testen** Sie ihre Implementierungen ausführlich! Geben Sie **Lösungsideen** an!