SCHETT MATTHIAS

SEN-ÜBUNG 09

Inhaltsverzeichnis

Aufgabe 1

Lösungsidee

Der generische Algorihtmus combine_if soll den Inhalt von zwei Container in einen dritten kopieren. Dazu soll, falls das Prädikat p zutrifft die Operation combOp ausgeführt werden. Der Algorithmus läuft mittels einer while Schleife von first1 bis end1 und ruft für jeden Iterator aus dem ersten und dem zweiten Container das Prädikat auf und führt anschließend die Operation aus. Das Ergebnis wird in den dritten Container kopiert. Da in der Schnittstelle nur Iteratoren übergeben werden, funktioniert der Algorithmus nur dann wenn alle drei Container gleich groß sind. Weiters funktioniert der Algorithmus nicht in Verbindung mit einem std::set da es hier nur konstante Iteratoren gibt, denen nicht direkt ein Wert zugewiesen werden kann.

Der Quellcode ist im Anhang unter Aufgabe 1 zu finden.

Testfälle

```
1. Original vector<int>
1, 1, 2, 3, 3,
2. Original vector<int>
3, 2, 1, 4, 1,
Result vector<int>
9, 4, 4, 144, 9,

1. Original list<float>
2, 2, 4, 5, 5,
2. Original list<float>
3, 4, 1, 3, 5,
Result list<float>
36, 64, 16, 225, 625,

1. Original vector<float>
9, 2, 2, 7, 8,
2. Original vector<float>
8, 5, 7, 3, 8,
Result vector<float>
5184, 100, 196, 441, 4096,
```

```
21
22 1, Original list<float>
23 6, 2, 7, 9, 4,
24 2. Original list<float>
25 9, 6, 7, 9, 2,
26 Result list<float>
27 2916, 144, 2401, 6561, 64,
```

Aufgabe 2

Lösungsidee

Die Templateklasse BinarySearchTree beinhaltet eine verschachtelte
TNode Struktur die alle Blätter des Baumes darstellt.
Folgende Methoden wurde als private deklariert

1. TNode* MakeNode(TValue const &val);
2. bool InsertSorted(TNode* &pLeaf, TValue const &value);
3. template <typename TVisitor> void PrintInOrder (TNode * pLeaf, TVisitor visitor) const;
4. template <typename TVisitor> void PrintPreOrder (TNode *pLeaf, TVisitor visitor) const;
5. template <typename TVisitor> void PrintPostOrder (TNode *pLeaf, TVisitor visitor) const;
6. void Flush (TNode * & pRoot);
7. TNode* copyTree(TNode* other);
   Folgende Methoden sind als public zu finden:
8. BinarySearchTree();
9. BinarySearchTree(BinarySearchTree<TValue, TPred> const &tree);
10. BinarySearchTree<TValue, TPred> &operator=(BinarySearchTree<TValue, TPred> const &tree);
11. B̃inarySearchTree();
12. bool Insert(TValue const &value); // returns false if already contained
13. template <typename TVisitor> void VisitPreOrder(TVisitor visitor) const;
14. template <typename TVisitor> void VisitInOrder(TVisitor visitor) const;
15. template <typename TVisitor> void VisitPostOrder(TVisitor visitor) const;

ad 1: Mit dieser Methode wird ein neuer Knoten am Heap angelegt.

ad 2: Fügt dem Baum einen neuen Knoten hinzu, dieser wird auch gleich an der richtigen Stelle einsortiert. Ist der Wert bereits vorhanden wird nicht eingefügt und false zurückgegeben.

ad 3: Inorder gibt den Baum in der richtigen Reihenfolge, also vom kleinsten zum Größten Wert.

ad 4: Gibt die Wurzel vor den Teilbäumen aus.

ad 5: Gibt die Wurzel nach den Teilbäumen aus.

ad 6: Flush löscht den gesamten Baum und gibt sämtlichen reservierten Speicher wieder frei.

ad 7 Hilfsfunktion die beim kopieren des Baumes hilft.

ad 8 Erstellt einen neuen Binären Such Baum

ad 9 Erstellt einen neuen Baum mit tree als Vorlage

ad 10 Weist einem Baum einen anderen zu.

ad 11 Räumt den allokierten Speicher wieder auf.

ad 12 Benutzt InsertSorted als Helper und fügt neuen Knoten hinzu.

ad 13 Siehe 4

ad 14 Siehe 3

ad 15 Siehe 5

Der Quellcode ist im Anhang unter Aufgabe 2 zu finden.

Testfälle

```
Insert Operation: true
Insert Operation: true
Insert Operation: true
Insert Operation: true
Insert Operation: true
Insert Operation: true
Insert Operation: true
Insert Operation: false

Print In Order:
2
3
4
5
6
7
8


Print Post Order
2
3
4
```

```
8
7
6
5


Print Pre Order
5
4
3
2
6
7
8

Print In Order of copy ctor:
2
3
4
5
6
7
8

Print In Order of assigment:
2
3
4
5
6
7
8
```

Anhang A
Aufgabe 1

Listing A.1: Header

```
1  ////////////////////////////////////
2  // Workfile    : GenericCombine.h
3  // Author      : Matthias Schett
4  // Date        : 27-05-2013
5  // Description : Combine if template
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 //*********************************
11 // Method:    combine_if
12 // FullName:  combine_if
13 // Access:    public
14 // Returns:   void
15 // Qualifier:
16 // Parameter: InputItor1 first1
17 // Parameter: InputItor1 end1
18 // Parameter: InputItor2 first2
19 // Parameter: OutputItor res
20 // Parameter: CombineOp combOp
21 // Parameter: Pred p
22 // Combines to container, all three container have to be of the same size
23 //*********************************
24 template<typename InputItor1, typename InputItor2, typename OutputItor, typename CombineOp,
       typename Pred>
25 void combine_if(InputItor1 first1, InputItor1 end1, InputItor2 first2, OutputItor res,
      CombineOp combOp, Pred p);
26
27 //*********************************
28 // Method:    Print
29 // FullName:  Print
30 // Access:    public
31 // Returns:   void
32 // Qualifier:
33 // Parameter: Container const & coll
```

```
34 // Parameter: std::string header
35 // Parameter: std::ostream & os
36 // Prints a container with the specified header to the specified ostream
37 //***********************************
38 template <typename Container>
39 void Print(Container const& coll, std::string header = "", std::ostream &os = std::cout );
40
41 #include "GenericCombine.impl"
```

Listing A.2: Implementierung

```
1  ////////////////////////////////////
2  // Workfile    : GenericCombine.impl
3  // Author      : Matthias Schett
4  // Date        : 27-05-2013
5  // Description : Combine if template
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9  #include <algorithm>
10 #include <string>
11 #include <ostream>
12 #include <iostream>
13 #include <iterator>
14
15 template<typename InputItor1, typename InputItor2, typename OutputItor, typename CombineOp,
       typename Pred>
16 void combine_if(InputItor1 first1, InputItor1 end1, InputItor2 first2, OutputItor res,
       CombineOp combOp, Pred p){
17     while(first1 != end1){
18         if(p(*first1) && p(*first2)){
19             *res = combOp(*first1,*first2++);
20         }
21         ++res; ++first1;
22     }
23 }
24
25 template <typename Container>
26 void Print(Container const& coll, std::string header, std::ostream &os){
27     typename Container::const_iterator begin(coll.begin());
28     typename Container::const_iterator end(coll.end());
29
30     os << header << endl;
31
32     ostream_iterator<Container::const_iterator::value_type> out_it (os, ", ");
33
34     copy(begin, end, out_it);
35
36 }
```

Listing A.3: Testtreiber

```
1  ////////////////////////////////////
2  // Workfile    : Main.cpp
3  // Author      : Matthias Schett
4  // Date        : 27-05-2013
5  // Description : Combine if template
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include <iostream>
11 #include "GenericCombine.h"
12 #include <vector>
13 #include <random>
14 #include <algorithm>
15 #include <iterator>
16 #include "RandomGen.h"
17 #include <ostream>
18 #include <list>
19 #include <set>
20
21 using namespace std;
22
23 int RandNum () {
24     return rgen::GetRandVal(1, 5);
25 }
26
27 int RandNumDobule () {
28     return rgen::GetRandVal(1000, 5000) / 500;
29 }
30
31 template <typename T>
32 T ProductSquare(T const &a, T const &b){
33     return static_cast<T>( (pow(a, 2) * pow(b,2)) );
34 }
35
36 template <typename T>
37 bool isPos(T const &a){
38     return a > 0;
39 }
40
41 void testInt(ostream &os = cout){
42     vector<int> myVec(5);
43     vector<int> myVec2(5);
44     vector<int> newVec(5);
45     //vector<int> newVec;
46
47     ostream_iterator<int> out_it (os, ", ");
48
```

```
49      generate(myVec.begin(), myVec.end(), RandNum);
50
51      generate(myVec2.begin(), myVec2.end(), RandNum);
52
53      combine_if(myVec.begin(), myVec.end(), myVec2.begin(), newVec.begin(), ProductSquare<
            int>, isPos<int>);
54
55      Print(myVec, "1. Original vector<int>");
56      os << endl;
57      Print(myVec2, "2. Original vector<int>");
58      os << endl;
59      Print(newVec, "Result vector<int>");
60      os << endl << endl;
61  }
62
63  void testIntList(ostream &os = cout){
64      list<int> myVec(5);
65      list<int> myVec2(5);
66      list<int> newVec(5);
67
68      ostream_iterator<int> out_it (os, ", ");
69
70      generate(myVec.begin(), myVec.end(), RandNum);
71
72      generate(myVec2.begin(), myVec2.end(), RandNum);
73
74      combine_if(myVec.begin(), myVec.end(), myVec2.begin(), newVec.begin(), ProductSquare<
            int>, isPos<int>);
75
76      Print(myVec, "1. Original list<float>");
77      os << endl;
78      Print(myVec2, "2. Original list<float>");
79      os << endl;
80      Print(newVec, "Result list<float>");
81      os << endl << endl;
82  }
83
84  void testFloat(ostream &os = cout){
85
86      vector<float> myVec(5);
87      vector<float> myVec2(5);
88      vector<float> newVec(5);
89
90      ostream_iterator<float> out_it (os, ", ");
91
92      generate(myVec.begin(), myVec.end(), RandNumDobule);
93
94      generate(myVec2.begin(), myVec2.end(), RandNumDobule);
95
```

```
96      combine_if(myVec.begin(), myVec.end(), myVec2.begin(), newVec.begin(), ProductSquare<
            float>, isPos<float>);
97
98      Print(myVec, "1. Original vector<float>");
99      os << endl;
100     Print(myVec2, "2. Original vector<float>");
101     os << endl;
102     Print(newVec, "Result vector<float>");
103     os << endl << endl;
104 }
105
106 void testFloatList(ostream &os = cout){
107     list<float> myVec(5);
108     list<float> myVec2(5);
109     list<float> newVec(5);
110
111     ostream_iterator<float> out_it (os, ", ");
112
113     generate(myVec.begin(), myVec.end(), RandNumDobule);
114
115     generate(myVec2.begin(), myVec2.end(), RandNumDobule);
116
117     combine_if(myVec.begin(), myVec.end(), myVec2.begin(), newVec.begin(), ProductSquare<
            float>, isPos<float>);
118
119     Print(myVec, "1, Original list<float>");
120     os << endl;
121     Print(myVec2, "2. Original list<float>");
122     os << endl;
123     Print(newVec, "Result list<float>");
124     os << endl << endl;
125 }
126
127 int main(){
128     rgen::Init();
129
130     ostream &os = cout;
131
132     testInt(os);
133     testIntList(os);
134
135     testFloat(os);
136     testFloatList(os);
137
138     cin.get();
139     return 0;
140 }
```

Anhang B
Aufgabe 2

Listing B.1: Header und Implementierung

```
1  ////////////////////////////////////
2  // Workfile    : BinarySearchTree.h
3  // Author      : Matthias Schett
4  // Date        : 04-06-2013
5  // Description : BinarySearch Tree
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 template <typename TValue, typename TPred = std::less<TValue> >
11 class BinarySearchTree {
12
13     struct TNode {
14         TValue value;
15         TNode *pLeft;
16         TNode *pRight;
17     };
18
19 private:
20     TNode *pRoot;
21
22     TNode* MakeNode(TValue const &val);
23     bool InsertSorted(TNode* &pLeaf, TValue const &value);
24     template <typename TVisitor> void PrintInOrder (TNode * pLeaf, TVisitor visitor) const;
25     template <typename TVisitor> void PrintPreOrder (TNode *pLeaf, TVisitor visitor) const;
26     template <typename TVisitor> void PrintPostOrder (TNode *pLeaf, TVisitor visitor) const
           ;
27     void Flush (TNode * & pRoot);
28     TNode* copyTree(TNode* other);
29
30 public:
31
32     BinarySearchTree();
33     BinarySearchTree(BinarySearchTree<TValue, TPred> const &tree);
```

```
34      BinarySearchTree<TValue, TPred> &operator=(BinarySearchTree<TValue, TPred> const &tree)
            ;
35      ~BinarySearchTree();
36
37      bool Insert(TValue const &value); // returns false if already contained
38
39      template <typename TVisitor> void VisitPreOrder(TVisitor visitor) const;
40      template <typename TVisitor> void VisitInOrder(TVisitor visitor) const;
41      template <typename TVisitor> void VisitPostOrder(TVisitor visitor) const;
42
43  };
44
45  template <typename TValue, typename TPred>
46  BinarySearchTree<TValue, TPred>::BinarySearchTree() : pRoot(NULL) {
47  }
48
49  template <typename TValue, typename TPred>
50  BinarySearchTree<TValue, TPred>::BinarySearchTree(BinarySearchTree<TValue, TPred> const &
        tree){
51      pRoot = copyTree(tree.pRoot);
52  }
53
54  template <typename TValue, typename TPred>
55  typename BinarySearchTree<TValue, TPred>::TNode* BinarySearchTree<TValue, TPred>::copyTree(
        typename BinarySearchTree<TValue, TPred>::TNode* other) {
56      //if node is empty (at bottom of binary tree)
57      /*
58      This creates a shallow copy which in turn causes a problem
59      with the deconstructor, could not work out how to create a
60      deep copy.
61      */
62      if (other == NULL) {
63          return NULL;
64      }
65
66      typename BinarySearchTree<TValue, TPred>::TNode* newNode = new BinarySearchTree<TValue,
            TPred>::TNode;
67
68      newNode->value = other->value;
69
70      newNode->pLeft = copyTree(other->pLeft);
71      newNode->pRight = copyTree(other->pRight);
72
73      return newNode;
74  }
75
76  template <typename TValue, typename TPred>
77  BinarySearchTree<TValue, TPred> &BinarySearchTree<TValue, TPred>::operator=(
        BinarySearchTree<TValue, TPred> const &tree){
```

```
78      if (this == &tree) {
79          return *this;
80      }
81      Flush(pRoot);
82      pRoot = copyTree(tree.pRoot);
83      return *this;
84  }
85
86  template <typename TValue, typename TPred>
87  BinarySearchTree<TValue, TPred>::~BinarySearchTree(){
88      Flush(pRoot);
89  }
90
91  template <typename TValue, typename TPred>
92  typename BinarySearchTree<TValue, TPred>::TNode *BinarySearchTree<TValue, TPred>::MakeNode(
        TValue const &val){
93      BinarySearchTree<TValue, TPred>::TNode *pNewNode = new BinarySearchTree<TValue, TPred
            >::TNode;
94
95      pNewNode->value = val;
96      pNewNode->pRight = 0;
97      pNewNode->pLeft = 0;
98
99      return pNewNode;
100 }
101
102 template <typename TValue, typename TPred>
103 void BinarySearchTree<TValue, TPred>::Flush (TNode * & pNode){
104     if (pNode != 0) {
105         Flush (pNode->pLeft);
106         Flush (pNode->pRight);
107         delete pNode;
108         pNode = 0;
109     }
110 }
111
112 template<typename TValue, typename TPred>
113 bool BinarySearchTree<TValue, TPred>::Insert(TValue const &value) {
114     if(pRoot != 0){
115         return InsertSorted(pRoot, value);
116     } else {
117         pRoot = MakeNode(value);
118         return true;
119     }
120
121 }
122
123 template <typename TValue, typename TPred>
```

```
124 bool BinarySearchTree<TValue, TPred>::InsertSorted(typename BinarySearchTree<TValue, TPred
        >::TNode* & pLeaf, TValue const &value){
125     if(pLeaf != 0){
126         if(pLeaf->value == value){
127             return false;
128         } else {
129             if(TPred()(value, pLeaf->value)) {
130                 return InsertSorted(pLeaf->pLeft, value);
131             } else {
132                 return InsertSorted(pLeaf->pRight, value);
133             }
134         }
135     } else {
136         pLeaf = MakeNode(value);
137         return true;
138     }
139
140 }
141
142 template <typename TValue, typename TPred>
143 template <typename TVisitor>
144 void BinarySearchTree<TValue, TPred>::PrintInOrder (TNode * pLeaf, TVisitor visitor) const
        {
145     if(pLeaf != 0){
146         if(pLeaf->pLeft != 0){
147             PrintInOrder(pLeaf->pLeft, visitor);
148         }
149
150         visitor(pLeaf->value);
151
152         if(pLeaf->pRight != 0){
153             PrintInOrder(pLeaf->pRight, visitor);
154         }
155     }
156 }
157
158 template <typename TValue, typename TPred>
159 template <typename TVisitor>
160 void BinarySearchTree<TValue, TPred>::VisitInOrder(TVisitor visitor) const {
161     if(pRoot != 0){
162         if(pRoot->pLeft != 0){
163             PrintInOrder(pRoot->pLeft, visitor);
164         }
165
166         visitor(pRoot->value);
167
168         if(pRoot->pRight != 0){
169             PrintInOrder(pRoot->pRight, visitor);
170         }
```

```
171     }
172 }
173
174 template <typename TValue, typename TPred>
175 template <typename TVisitor>
176 void BinarySearchTree<TValue, TPred>::PrintPreOrder (TNode *pLeaf, TVisitor visitor) const
        {
177     if(pLeaf != 0){
178         visitor(pLeaf->value);
179
180         if(pLeaf->pLeft != 0){
181             PrintPreOrder(pLeaf->pLeft, visitor);
182         }
183
184         if (pLeaf->pRight != 0) {
185             PrintPreOrder(pLeaf->pRight, visitor);
186         }
187     }
188 }
189
190 template <typename TValue, typename TPred>
191 template <typename TVisitor>
192 void BinarySearchTree<TValue, TPred>::VisitPreOrder(TVisitor visitor) const{
193     if(pRoot != 0){
194         visitor(pRoot->value);
195
196         if(pRoot->pLeft != 0){
197             PrintPreOrder(pRoot->pLeft, visitor);
198         }
199
200         if (pRoot->pRight != 0) {
201             PrintPreOrder(pRoot->pRight, visitor);
202         }
203     }
204 }
205
206 template <typename TValue, typename TPred>
207 template <typename TVisitor>
208 void BinarySearchTree<TValue, TPred>::PrintPostOrder(TNode *pLeaf, TVisitor visitor) const
        {
209     if(pLeaf != 0){
210         if(pLeaf->pLeft != 0){
211             PrintPostOrder(pLeaf->pLeft, visitor);
212         }
213
214         if(pLeaf->pRight != 0){
215             PrintPostOrder(pLeaf->pRight, visitor);
216         }
217
```

```
218         visitor(pLeaf->value);
219     }
220 }
221 
222 template <typename TValue, typename TPred>
223 template <typename TVisitor>
224 void BinarySearchTree<TValue, TPred>::VisitPostOrder(TVisitor visitor) const {
225     if(pRoot != 0){
226         if(pRoot->pLeft != 0){
227             PrintPostOrder(pRoot->pLeft, visitor);
228         }
229 
230         if(pRoot->pRight != 0){
231             PrintPostOrder(pRoot->pRight, visitor);
232         }
233 
234         visitor(pRoot->value);
235     }
236 }
```

Listing B.2: Testtreiber

```
1  ////////////////////////////////////
2  // Workfile    : Main.cpp
3  // Author      : Matthias Schett
4  // Date        : 04-06-2013
5  // Description : BinarySearch Tree
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9  
10 
11 #include <iostream>
12 #include <vld.h>
13 #include "BinarySearchTree.h"
14 
15 
16 using namespace std;
17 
18 void Print(int const x){
19     cout << x << endl;
20 }
21 
22 int main(){
23 
24     BinarySearchTree<int> tree;
25 
26     cout << std::boolalpha << "Insert Operation: " << tree.Insert(5) << endl;
27     cout << "Insert Operation: " << tree.Insert(4) << endl;
28     cout << "Insert Operation: " << tree.Insert(3) << endl;
```

```
29      cout << "Insert Operation: " << tree.Insert(2) << endl;
30      cout << "Insert Operation: " << tree.Insert(6) << endl;
31      cout << "Insert Operation: " << tree.Insert(7) << endl;
32      cout << "Insert Operation: " << tree.Insert(8) << endl;
33      cout << "Insert Operation: " << tree.Insert(3) << endl;
34
35      cout << endl << "Print In Order: " << endl;
36      tree.VisitInOrder(Print);
37      cout << endl << endl << "Print Post Order" << endl;
38      tree.VisitPostOrder(Print);
39      cout << endl << endl << "Print Pre Order" << endl;
40      tree.VisitPreOrder(Print);
41
42
43      BinarySearchTree<int> tree2 (tree);
44      cout << endl << "Print In Order of copy ctor: " << endl;
45      tree.VisitInOrder(Print);
46
47      BinarySearchTree<int> tree3 = tree;
48      cout << endl << "Print In Order of assigment: " << endl;
49      tree.VisitInOrder(Print);
50
51
52      cin.get();
53      return 0;
54  }
```