SEN-Übung 2.1

Schett Matthias

13. April 2013

1 Aufgabe 1

1.1 Lösungsidee

Es soll ein binärer Suchbaum implementiert werden.
Dieser soll folgende Funktionen aufweisen:

1. bool Contains (TTreeNode const * const pRoot, int const Data);
2. void Delete (TTreeNode * & pRoot, int const Data);
3. void Flush (TTreeNode * & pRoot);
4. int Height (TTreeNode const * const pRoot);
5. void InsertSorted (TTreeNode * & pRoot, int const Data);
6. void PrintInOrder (TTreeNode const * const pRoot);
7. void PrintPostOrder (TTreeNode const * const pRoot);
8. void PrintPreOrder (TTreeNode const * const pRoot);
9. void PrintTree (TTreeNode const * const pRoot, size_t const Indent);

ad 1: Die Contains Methode sucht rekursiv ob in dem gegebenen Tree der gesuchte Wert enthalten ist.
ad 2: Die Delete Methode löscht einen Knoten mit dem gegebenen Wert aus dem Baum und hängt die anderen Knoten wieder richtig ein.
ad 3: Flus löscht den gesamten Baum und gibt sämtlichen reservierten Speicher wieder frei.
ad 4: Gibt die Höhe des Baumes aus.
ad 5: Fügt dem Baum einen neuen Knoten hinzu, dieser wird auch gleich an der richtigen Stelle einsortiert.
ad 6: Inorder gibt den Baum in der richtigen Reihenfolge, also vom kleinsten zum Größten Wert.
ad 7: Gibt die Wurzel nach den Teilbäumen aus.
ad 8: Gibt die Wurzel vor den Teilbäumen aus.
ad 9 Gibt den Baum von oben nach unten aus, genauso wie er aufgezeichnet werden würde.
Weiters werden noch folgende Hilfsmethoden verwendet:

1. TTreeNode * MakeNode (int const Data);
2. void PrintNodeData(TTreeNode const * const pNode);

ad 1: Mit dieser Methode wird ein neuer Knoten am Heap angelegt.
ad 2 Diese Methode gibt das Data Element der TTreeNode Struktur auf der std::out aus.

1.2 Programmcode in C++

1.2.1 Modul Header

```
1  ////////////////////////////////////////
2  // Workfile    : BinaryTree.h
3  // Author      : Matthias Schett
4  // Date        : 11-03-2013
5  // Description : Binary Search Tree
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////
9
10 struct TTreeNode {
11     int Data;
12     TTreeNode * pLeft;
13     TTreeNode * pRight;
14 };
15
16 bool Contains (TTreeNode const * const pRoot, int const Data);
17 void Delete (TTreeNode * & pRoot, int const Data);
18 void Flush (TTreeNode * & pRoot);
19 int Height (TTreeNode const * const pRoot);
20 void InsertSorted (TTreeNode * & pRoot, int const Data);
21
22 void PrintInOrder (TTreeNode const * const pRoot);
23 void PrintPostOrder (TTreeNode const * const pRoot);
24 void PrintPreOrder (TTreeNode const * const pRoot);
25 void PrintTree (TTreeNode const * const pRoot, size_t const
     Indent);
```

1.2.2 Modul Cpp

```
1  ////////////////////////////////////////
2  // Workfile    : BinaryTree.cpp
3  // Author      : Matthias Schett
4  // Date        : 11-03-2013
5  // Description : Binary Search Tree
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////
9
10 #include "BinaryTree.h"
11 #include <iostream>
12 #include <string>
13
```

```
14 // ---------- PRIVATE SECTION --------
15
16 // Not in Header
17 TTreeNode * MakeNode (int const Data){
18     TTreeNode* const pNewNode = new TTreeNode;
19
20     pNewNode->Data = Data;
21     pNewNode->pLeft = 0;
22     pNewNode->pRight = 0;
23
24     return pNewNode;
25 }
26
27 void PrintNodeData(TTreeNode const * const pNode){
28     std::cout << "(" << pNode->Data << ") ";
29 }
30
31
32 // ---------- PRIVATE SECTION --------
33
34 bool Contains (TTreeNode const * const pRoot, int const Data){
35     if(pRoot != 0){
36
37         if( ( pRoot->Data == Data ) ){
38             return true;
39         } else {
40             if(Data < pRoot->Data){
41                 return Contains(pRoot->pLeft, Data);
42             } else {
43                 return Contains(pRoot->pRight, Data);
44             }
45         }
46     }
47
48     return false;
49 }
50
51 void Delete (TTreeNode * & pRoot, int const Data){
52     TTreeNode *pNodeToDelete = 0;
53     TTreeNode *pTemp = 0;
54     TTreeNode *pPrevious = 0;
55
56     if(pRoot != 0 && Contains(pRoot, Data)){
57         if(Data < pRoot->Data){
58             Delete(pRoot->pLeft, Data);
```

```
59          } else if(Data > pRoot->Data){
60              Delete(pRoot->pRight, Data);
61          } else {
62              pNodeToDelete = pRoot;
63              if(pNodeToDelete->pRight == 0){
64                  pRoot = pNodeToDelete->pLeft;
65              } else if(pNodeToDelete->pLeft == 0){
66                  pRoot = pNodeToDelete->pRight;
67              } else {
68                  pTemp = pNodeToDelete->pLeft;
69
70                  while(pTemp->pRight != 0){
71                      pPrevious = pTemp;
72                      pTemp = pTemp->pRight;
73                  }
74
75                  pNodeToDelete->Data = pTemp->Data;
76
77                  if(pPrevious != 0){
78                      pPrevious->pRight = pTemp->pLeft;
79                  } else {
80                      pNodeToDelete->pLeft = pTemp->pLeft;
81                  }
82
83                  pNodeToDelete = pTemp;
84              }
85
86              delete pNodeToDelete;
87              pNodeToDelete = 0;
88          }
89      }
90  }
91
92  void Flush (TTreeNode * & pRoot){
93      if (pRoot != 0)
94      {
95          Flush (pRoot->pLeft);
96          Flush (pRoot->pRight);
97          delete pRoot;
98          pRoot = 0;
99      }
100 }
101
102 int Height (TTreeNode const * const pRoot){
103     if(pRoot == 0){
```

```
104         return -1;
105     }
106 
107     int lefth = Height(pRoot->pLeft);
108     int righth = Height(pRoot->pRight);
109 
110     if(lefth > righth){
111         return lefth + 1;
112     } else {
113         return righth + 1;
114     }
115 }
116 
117 void InsertSorted (TTreeNode * & pRoot, int const Data){
118     if(pRoot != 0){
119         if(Data < pRoot->Data){
120             InsertSorted(pRoot->pLeft, Data);
121         } else {
122             InsertSorted(pRoot->pRight, Data);
123         }
124     } else {
125         pRoot = MakeNode(Data);
126     }
127 }
128 
129 void PrintInOrder (TTreeNode const * const pRoot){
130     if(pRoot != 0){
131         if(pRoot->pLeft != 0){
132             PrintInOrder(pRoot->pLeft);
133         }
134 
135         PrintNodeData(pRoot);
136 
137         if(pRoot->pRight != 0){
138             PrintInOrder(pRoot->pRight);
139         }
140     }
141 }
142 
143 void PrintPostOrder (TTreeNode const * const pRoot){
144     if(pRoot != 0){
145         if(pRoot->pLeft != 0){
146             PrintPostOrder(pRoot->pLeft);
147         }
148 
```

```
149         if(pRoot->pRight != 0){
150             PrintPostOrder(pRoot->pRight);
151         }
152 
153         PrintNodeData(pRoot);
154     }
155 }
156 
157 void PrintPreOrder (TTreeNode const * const pRoot){
158     if(pRoot != 0){
159         PrintNodeData(pRoot);
160 
161         if(pRoot->pLeft != 0){
162             PrintPreOrder(pRoot->pLeft);
163         }
164 
165         if (pRoot->pRight != 0) {
166             PrintPreOrder(pRoot->pRight);
167         }
168     }
169 }
170 
171 void PrintTree (TTreeNode const * const pRoot, size_t const
    Indent){
172     if(pRoot != 0){
173         std::string s(Indent, ' ');
174         PrintTree(pRoot->pRight, Indent + 3);
175         std::cout << s;
176         PrintNodeData(pRoot);
177         std::cout << std::endl;
178         PrintTree(pRoot->pLeft, Indent + 3);
179     }
180 
181 }
```

1.2.3 Testreiber

```
1  ////////////////////////////////////
2  // Workfile    : main.cpp
3  // Author      : Matthias Schett
4  // Date        : 11-03-2013
5  // Description : Binary Search Tree
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include "BinaryTree.h"
11 #include <iostream>
12
13 using namespace std;
14
15 int main(){
16
17
18     TTreeNode *pTree = 0;
19
20     InsertSorted(pTree, 5);
21     InsertSorted(pTree, 3);
22     InsertSorted(pTree, 4);
23     InsertSorted(pTree, 2);
24     InsertSorted(pTree, 7);
25     InsertSorted(pTree, 6);
26     InsertSorted(pTree, 8);
27
28     cout << "Tree contains a 3 " << (Contains(pTree, 3) ? "yes
           " : "no") << endl;
29
30     cout << "Tree contains a 10 " << (Contains(pTree, 10) ? "
           yes" : "no") << endl;
31
32     cout << "The height of the tree is " << Height(pTree) <<
           endl << "Pre Order" << endl;
33
34     PrintPreOrder(pTree);
35
36     cout << endl << endl << "In Order" << endl;
37
38     PrintInOrder(pTree);
39
40     cout << endl << endl << "Post Order" << endl;
```

```
41
42      PrintPostOrder(pTree);
43
44      cout << endl << endl << "Print Tree" << endl;
45
46      PrintTree(pTree, 3);
47
48      // TODO: Delete And Print
49
50      cout << endl << endl << "Delete and print Tree" << endl;
51
52      Delete(pTree, 3);
53
54      PrintInOrder(pTree);
55
56      cout << endl << endl << "Flush and print Tree" << endl;
57      Flush(pTree);
58
59      PrintInOrder(pTree);
60
61      cin.get();
62      return 0;
63  }
```

1.3 Testfälle

Ausgabe des Testtreibers:

```
Tree contains a 3 yes
Tree contains a 10 no
The height of the tree is 2
Pre Order
(5) (3) (2) (4) (7) (6) (8)

In Order
(2) (3) (4) (5) (6) (7) (8)

Post Order
(2) (4) (3) (6) (8) (7) (5)

Print Tree
          (8)
      (7)
          (6)
  (5)
          (4)
      (3)
          (2)


Delete and print Tree
(2) (4) (5) (6) (7) (8)

Flush and print Tree
```

2 Aufgabe 2

2.1 Lösungsidee

Es soll ein Parser für Arithmetische Ausdrücke gebaut werden. Dazu wird mittels der EBNF Notation eine Grammatik definiert. Als Input wird mithilfe der vorhanden Scanner Klasse eine Datei gelesen und aus dieser dann der Arithmetische Ausdruck geparst und das Ergebnis auf std::out angezeigt. Fehlermeldung werden ebenfalls über std::out angegben.

2.2 Programmcode in C++

2.2.1 Modul Header

```
1  ////////////////////////////////////////
2  // Workfile    : ArithExprModule.h
3  // Author      : Matthias Schett
4  // Date        : 14-03-2013
5  // Description : Binary Search Tree
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////
9  #include "scanner.h"
10
11 // Pruefung auf Terminalen Anfang fuer den AddOp
12 bool isTbAddOp(scanner &scan);
13
14 // Pruefung auf Terminalen Anfang fuer MulOp
15 bool isTbMulOp(scanner &scan);
16
17 // Pruefung auf Terminalen Anfang fuer Expression
18 bool isTbExpression(scanner &scan);
19
20 // Pruefung auf Terminalen Anfang fuer Term
21 bool isTbTerm(scanner &scan);
22
23 // Pruefung auf Terminalen Anfang fuer Factor
24 bool isTbFactor(scanner &scan);
25
26 // Scannt den Additionsoperator und liefert das Vorzeichen
27 int scanAddOp(scanner &scan);
28
29 // Erkenne einen Faktor
30 int scanFactor(scanner &scan);
31
32 //Erkennen einen Term
33 int scanTerm(scanner &scan);
34
35 // Erkenne einen Arithmetischen Ausdruck
36 int scanExpression(scanner &scan);
```

2.2.2 Modul Cpp

```
1  ////////////////////////////////////////
2  // Workfile    : ArithExprModule.cpp
3  // Author      : Matthias Schett
```

```
4  // Date        : 14-03-2013
5  // Description : Binary Search Tree
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9  #include "ArithExprModule.h"
10 #include <iostream>
11
12 void doErrorHandling(std::string errorMessage){
13     std::cout << "Es wurde ein ungueltiges Zeichen in der
           Eingabe erkannt. Ergebnis wird trotz Fehlers
           ausgegeben: " << errorMessage << " ";
14 }
15
16 bool isTbAddOp(scanner &scan){
17     return scan.symbol_is_plus() || scan.symbol_is_minus();
18 }
19
20 bool isTbMulOp(scanner &scan){
21     return scan.symbol_is_multiply() || scan.
           symbol_is_division();
22 }
23
24 bool isTbExpression(scanner &scan){
25     return isTbAddOp(scan) || isTbTerm(scan);
26 }
27
28 bool isTbTerm(scanner &scan){
29     return isTbFactor(scan);
30 }
31
32 bool isTbFactor(scanner &scan){
33     return scan.symbol_is_integer() || scan.symbol_is_lpar();
34 }
35
36 int scanAddOp(scanner &scan){
37     int sign = 0;
38
39     if(scan.symbol_is_plus()){
40         sign = +1;
41     } else if(scan.symbol_is_minus()){
42         sign = -1;
43     } else {
44         doErrorHandling("Es wurde ein + oder ein - erwartet");
45     }
```

```
46
47      scan.next_symbol();
48
49      return sign;
50  }
51
52  int scanFactor(scanner &scan){
53      int val = 0;
54
55      if(scan.symbol_is_integer()){
56          val = scan.get_integer();
57      } else if(isTbExpression(scan)){
58          if(scan.symbol_is_lpar()){
59              scan.next_symbol();
60          }
61          val = scanExpression(scan);
62      } else {
63          doErrorHandling("Es wurde ein weiterer Ausdruck oder
                eine Zahl erwartet!");
64      }
65
66      scan.next_symbol();
67
68      return val;
69  }
70
71  int scanTerm(scanner &scan){
72
73      int val = scanFactor(scan);
74
75      while(isTbMulOp(scan)){
76          if(scan.symbol_is_multiply()){
77              scan.next_symbol();
78              val *= scanFactor(scan);
79          } else if(scan.symbol_is_division()){
80              scan.next_symbol();
81              int factor = scanFactor(scan);
82              if(factor != 0){
83                  val /= factor;
84              } else {
85                  doErrorHandling("Division durch 0 ist nicht
                        erlaubt");
86              }
87          } else {
```

```
88              doErrorHandling("Es wurde ein * oder / erwartet!")
                    ;
89          }
90
91      }
92
93      return val;
94  }
95
96  int scanExpression(scanner &scan){
97
98      int sign = 1;
99      int value = 0;
100
101     if(isTbAddOp(scan)){
102         sign = scanAddOp(scan);
103     }
104
105     value = sign * scanTerm(scan);
106
107     while(isTbAddOp(scan)){
108         sign = scanAddOp(scan);
109         value += sign * scanTerm(scan);
110     }
111
112     return value;
113 }
```

2.2.3 Testreiber

```
1  ////////////////////////////////////
2  // Workfile    : ArithExpr.cpp
3  // Author      : Matthias Schett
4  // Date        : 14-03-2013
5  // Description : Binary Search Tree
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9  #include <iostream>
10 #include <fstream>
11 #include "scanner.h"
12 #include "ArithExprModule.h"
13
14 int main (){
15
16     std::ifstream file("Input.txt");
17     scanner scan(file);
18
19     while(!scan.eof()){
20         std::cout << scanExpression(scan) << std::endl;
21     }
22     std::cin.get();
23     return 0;
24 }
```

2.3 Testfälle

Input:
8 * 2 + 4 * 2
(10 * 10) - 1
5 * 5 + (*1)
2 * 3 + 4 / 0

Output:
24
99
Es wurde ein ungueltiges Zeichen in der Eingabe erkannt. Ergebnis wird trotz Fehlers ausgegeben: Es wurde ein weiterer Ausdruck oder eine Zahl erwartet! 25
Es wurde ein ungueltiges Zeichen in der Eingabe erkannt. Ergebnis wird trotz Fehlers ausgegeben: Es wurde ein weiterer Ausdruck oder eine Zahl erwartet! 0
Es wurde ein ungueltiges Zeichen in der Eingabe erkannt. Ergebnis wird trotz Fehlers ausgegeben: Division durch 0 ist nicht erlaubt 10