SWO 3x Übung zu Softwareentwicklung mit klassischen Sprachen u. Bibliotheken 3 **WS 2014/15, Übung 06**

Abgabetermin: Sa in der KW 03

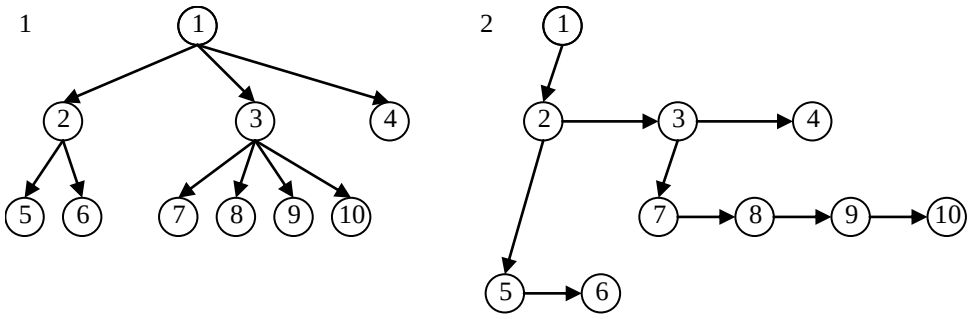
☐ **Gr. 1,** DI Franz Gruber-Leitner **Name** Roman Lumetsberger **Aufwand in h** 8

☐ **Gr. 2,** Dr. Erik Pitzer

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

1. Allgemeine Bäume (4 + 2 + 4 + 2 + 4 Punkte)

Im ersten Semester haben wir uns intensiv mit *Binärbäumen* beschäftigt, bei denen jeder Knoten maximal zwei Kinder hat. Eine Spezialform davon sind *binäre Suchbäume*, bei denen die Knoten "geordnet" sind (linkes Kind < Knoten <= rechtes Kind). Im dritten Semester haben wir uns bisher mit *allgemeinen Graphen* und deren Repräsentation (Adjazenzmatrix und Adjazenzliste) sowie Algorithmen darauf beschäftigt. *Allgemeine Bäume* fehlen noch in der Sammlung.
Entwickeln Sie einen abstrakten Datentyp zur Verwaltung von *allgemeinen Bäumen*, siehe z.B. Abbildung 1. Eine einfache Repräsentation solcher Bäume besteht darin, diese auf den Spezialfall der Binärbäume zurückzuführen, indem jeder Knoten einen Zeiger auf das erste Kind (in der Komponente `firstChild`) und einen Zeiger auf den Anfang der „Liste“ seiner Geschwister (in der Komponente `nextSibling`) hat. Jeder Knoten kommt hier mit zwei Zeigern aus, unabhängig davon, wie viele Kinder er hat. Man nennt diese Darstellung *kanonische Form*, siehe Abbildung 2.



Solch ein Datentyp lässt sich elegant mit den Mitteln der objektorientierten Programmierung realisieren: Ein möglicher Ansatz besteht darin, die Knoten des Baums und den Baum selbst durch zwei Klassen zu modellieren. Jeder Baum hat einen Zeiger auf den Wurzelknoten. Hier ist der Ansatz zur möglichen Klassendeklarationen für die abstrakte Klasse `Node` und die Klasse `Tree`:

```
class Node {
  private:
    Node *firstChild, *nextSibling;
    ...
  public:
    explicit Node(Node *firstChild = nullptr, Node *nextSibling = nullptr);
    virtual ~Node();
    virtual Node* getFirstChild() const;
    virtual Node* getNextSibling() const;
    virtual void setFirstChild(Node *n);
    virtual void setNextSibling(Node *n);
    virtual void print(std::ostream &os) const = 0;
    ...
};
```

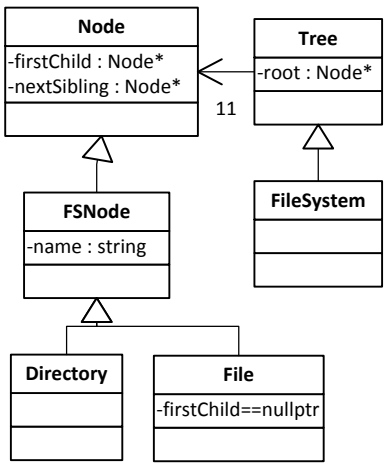
```
class Tree {
  protected:
    Node *root;
    ...
  public:
    Tree();
    virtual ~Tree();
    virtual Node* getRoot() const;
    virtual void insertChild(Node *parent, Node *child);
    virtual void deleteSubtree(Node *node);
    virtual int getSize() const;
    virtual void Clear();
    virtual void DeleteElements();
    virtual void print(std::ostream &os) const;
    ...
};
```

1. Erstellen Sie für beide Klassen je eine h- und eine cpp-Datei und implementieren Sie alle oben deklarierten Methoden in den beiden Klassen.
2. Erstellen Sie einen ersten konkreten Knotentyp `IntNode` der von `Node` abgeleitet ist und eine Datenkomponente `value` besitzt.
3. Fügen Sie in die Klasse `Tree` jene Methoden zum Einfügen neuer Knoten ein, um damit den Baum aus obigem Beispiel aufbauen zu können und stellen Sie eine Methode `getSize()` zur Verfügung, welche die Anzahl der Knoten im Baum liefert.
4. Ausgabeoperatoren `operator<<()` für beide Klassen dürfen natürlich auch nicht fehlen, damit Sie besser testen können.
5. Implementieren Sie einen geeigneten Zuweisungsoperator und Kopierkonstruktor für `Tree` und erläutern Sie anhand von Darstellungen deren Funktion.

Testen Sie Ihre Klassen ausführlich, überprüfen Sie, ob alle Objekte freigegeben werden.

2. Hierarchisches Dateisystem (8 Punkte)

Auf Basis der oben erstellten "Infrastruktur" für allgemeine Bäume, bietet es sich an, ein hierarchisches Dateisystem (*file system*) zu implementieren.



Erstellen Sie einen weiteren konkreten Knotentyp `FSNode` der eine neue Datenkomponente `name` vom Datentyp `string` hat, und von `FSNode` abgeleitet, zwei weitere Klassen: eine Klasse `Directory` und eine Klasse `File`. Die wesentliche Eigenschaft einer Datei ist, dass sie keine weiteren Dateien oder Verzeichnisse enthalten kann es muss also in einem `File`-Objekten immer `firstChild == nullptr` gelten.

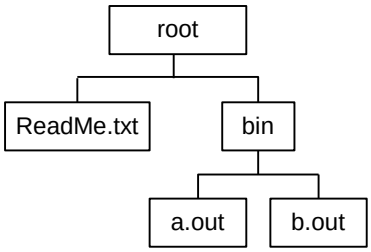
Zum Schluss leiten Sie noch eine Klasse `FileSystem` von `Tree` ab, die Funktionen zur Modifikation bietet.

Damit man mit Ihrem Dateisystem arbeiten kann, stellen Sie in der Klasse *FileSystem* folgende Methoden zur Verfügung, die, entsprechend der gleichnamigen Shell-Befehle, Methoden zum Erstellen und Löschen von Dateien und Verzeichnissen zur Verfügung stellen:

```
void touch(const string &path, const string &filename); // create new file
void mkdir(const string &path, const string &dirname); // create new directory
void rm(const string &path, const string &filename); // remove file
void rmdir(const string &path, const string &dirname); // remove directory
void ls() const; // list file system contents
```

Ihre Klassen sollten wie folgt verwendet werden können, um ein „Dateisystem“ gemäß der Abbildung rechts erstellen und verwalten zu können:

```
FileSystem *fs = new FileSystem();
fs->mkdir("", "root");
fs->touch("root", "ReadMe.txt");
fs->mkdir("root", "bin");
fs->touch("root/bin", "a.exe");
fs->touch("root/bin", "b.exe");
fs->ls();
fs->rm("root/bin", "a.exe");
fs->rmdir("", "root"); // -> ERROR: dir not empty
fs->ls();
fs->rm("root/bin", "b.exe");
fs->rmdir("", "root/bin");
fs->rm("root", "ReadMe.txt");
fs->rmdir("", "root");
fs->ls();
delete fs;
```



Hinweis:
Eine einfache Möglichkeit um Pfade mit Hilfe eines Separators zu trennen stellt die Funktion `strtok()` aus der C-Standardbibliothek dar, deren Funktion Sie in den Manual Pages nachlesen können.

1 Aufgabe 1 - Allgemeine Bäume

1.1 Anmerkungen

Diese Aufgabe wird mit der *minilib* umgesetzt und dadurch werden alle Methoden mit einem großen Anfangsbuchstaben benannt, da dies in der minilib Konvention ist.

1.2 Lösungsidee

Der Baum wird, wie in der Angabe vorgegeben, mit den 2 Klassen *Node* und *Tree* umgesetzt.

1.2.1 Node

Diese Klasse implementiert einen Knoten im Baum. Dabei hat ein solcher Node genau einen Zeiger auf einen Kindknoten und einen Zeiger auf den nächsten Geschwisterknoten. Mit diesen beiden Zeigern können allgemeine Bäume umgesetzt werden.

- Die Zugriffe auf die Datenkomponenten werden mit Zugriffsmethoden gewährleistet.
- Der *Destruktor* löscht sowohl den Kind- als auch den Geschwisterknoten und gibt den Speicher frei.
- Die Methode *Clone* kopiert sowohl den Geschwister- als auch den Kindknoten. Diese Methode wird verwendet um eine Kopie des Baumes anzulegen.

1.2.2 Tree

Die Klasse *Tree* implementiert die geforderten Methoden um einen allgemeinen Baum mit Hilfe der Klasse *Node* aufzubauen.

- *InsertChild* fügt einen Kindknoten unter den gegebenen Knoten ein. Sollte der *Parent* gleich nullptr sein, dann wird dieser Knoten als Root-Knoten verwendet. Dabei ist zu beachten, dass der *Parent* auch wirklich Teil des Baumes ist.
- *DeleteSubtree* löscht einen Teilbaum und gibt auch seinen Speicher frei.
- *Clear* setzt nur den Root-Knoten auf nullptr und **gibt keinen Speicher frei**.
- *GetSize* berechnet die akuelle Anzahl an Knoten im Baum.

Kopierkonstruktor
Der Kopierkonstruktor kopiert den gesamten Baum, indem er die Clone Methode des Root-Elements verwendet und dadurch auch alle Kinder und Geschwister kopiert werden.
Zuweisungsoperator
Auch dieser Operator kopiert den gesamten Baum, indem er alle Nodes kopiert. Hier ist zu beachten, dass der alte Baum freigegeben werden muss. Dabei werden alle Nodes freigegeben und können danach nicht mehr verwendet werden.

2 Aufgabe 2 - Hierarchisches Dateisystem

2.1 Anmerkungen

Auch diese Aufgabe wird mit der *minilib* umgesetzt und dadurch werden alle Methoden mit einem großen Anfangsbuchstaben benannt, da dies in der minilib Konvention ist.

2.2 Lösungsidee

2.2.1 Node Klassen

Die benötigten Klassen werden, wie in der Angabe vorgegeben, abgeleitet und die Methoden entsprechend implementiert. Die Einschränkung der Klasse *File*, keine Kinder zu haben, kann durch das Überschreiben der Methode SetFirstChild sichergestellt werden.

2.2.2 Filesystem

Diese Klasse implementiert die geforderten Methoden. Dabei wird diese Klasse von *Tree* abgeleitet.
Zu den in der Basisklasse implementierten Methoden wird eine weitere benötigt, die es erlaubt einen Node im Tree anhand eines gegebenen Pfades zu finden. Diese muss also ausgehend vom Root alle Geschwister durchsuchen und bei jeden Seperator eine Ebene tiefer weitersuchen.
Hier wird die Funktion *strtok* der C-Standardbibliothek verwendet.

- **Touch**: Fügt einen neuen Knoten der Klasse *File* in den Baum ein.
- **Mkdir**: Fügt einen neuen Knoten der Klasse *Directory* in den Baum ein.
- **Rm**: Sucht und löscht den Knoten. Dabei muss beachtet werden, dass der gefundene Knoten vom Typ *File* ist.
- **Rmdir**: Verwendet *DeleteSubtree*, um das Verzeichnis zu löschen. Dabei muss geprüft werden, dass es keine Kindknoten gibt(also das Verzeichnis leer ist).
- **Ls**: Verwendet *Print*, um das Dateisystem auszugeben.

2.3 Sourcecode

Node.h

```
#ifndef NODE_H
#define NODE_H

#include <ostream>
#include <MLObject.h>


/* Node class used as base class for tree nodes
 *
 */
class Node : public ML::Object
{
  protected:
    Node *firstChild, *nextSibling;
  public:
    explicit Node (Node* firstChild = nullptr, Node* nextSibling = nullptr);
    // Deletes the firstChild and nextSibling and all containing sub elements
    virtual ~Node();

    virtual Node* GetFirstChild() const { return firstChild;}
    virtual Node* GetNextSibling() const { return nextSibling;}

    virtual void SetFirstChild(Node* node) { firstChild = node;}
    virtual void SetNextSibling(Node* node) { nextSibling = node;}

    //Prints the Node
    virtual void Print(std::ostream &os) const;

    // used for << operator in ML::Object
    virtual std::string AsString() const = 0;

    /* deeply clones the node */
    virtual Node* Clone() const = 0;

};

#endif // NODE_H
```

IntNode.h

```
#ifndef INTNODE_H
#define INTNODE_H

#include <sstream>
#include "MLObject.h"
#include "Node.h"

class IntNode : public Node
{
  private:
    int value;

  public:
```

```
14      IntNode(int value, Node* firstChild = nullptr, Node* nextSibling = nullptr);
15      virtual ~IntNode();
16
17      virtual std::string AsString() const override;
18
19      /* clones the node */
20      virtual IntNode *Clone() const override;
21
22    protected:
23
24  };
25
26  #endif // INTNODE_H
```

Tree.h

```
1   #ifndef TREE_H
2   #define TREE_H
3
4   #include <ostream>
5   #include <MLObject.h>
6   #include <Node.h>
7
8   class Tree : public ML::Object {
9
10   protected:
11    Node *root;
12    /* helper function used for checking if the given node is inside the tree */
13    virtual bool IsTreeNode(Node* startNode, Node* node) const;
14
15    /* finds the parent node of the given node */
16    virtual Node* GetParent(Node *startNode, Node *node) const;
17   public:
18    /* default constructor with null root */
19    Tree();
20    /* copy constructor */
21    Tree(const Tree &other);
22
23    /* destructor */
24    virtual ~Tree();
25
26    /*returns the root element */
27    virtual Node* GetRoot() const;
28
29    /* inserts a child at the given position */
30    virtual void InsertChild(Node *parent, Node *child);
31
32    /* Removes the subtree and deletes all elements */
33    virtual void DeleteSubtree(Node *node);
34
35    /* clears the tree without deleting the elements
36       caller has to take care to delete the elements
37    */
38    virtual void Clear();
39    /* removes and deletes all elements
40       held references are not valid anymore after calling this method */
41    virtual void DeleteElements();
42
```

```
43   /*prints the tree to the given stream */
44   virtual void Print(std::ostream &os) const;
45
46   /*minilib override used for << operator */
47   virtual std::string AsString() const override;
48
49   /* calculates the current size of the tree */
50   virtual int GetSize() const;
51
52   /*assignment operator */
53   virtual Tree& operator =(const Tree& other);
54 };
55
56 #endif // TREE_H
```

FSNode.h

```
1  #ifndef FSNODE_H
2  #define FSNODE_H
3
4  #include "Node.h"
5
6  class FSNode : public Node
7  {
8    protected:
9      std::string name;
10   public:
11     FSNode(std::string name, Node* firstChild = nullptr, Node* nextSibling = nullptr);
12     virtual ~FSNode();
13
14     virtual void SetName(std::string newName) {name = newName; }
15     virtual std::string GetName() {return name;}
16
17     /* used for << operator in ML::Object */
18     virtual std::string AsString() const override;
19
20     /* clones the node */
21     virtual FSNode* Clone() const override;
22
23 };
24
25 #endif // FSNODE_H
```

Directory.h

```
1  #ifndef DIRECTORY_H
2  #define DIRECTORY_H
3
4  #include "FSNode.h"
5
6  class Directory : public FSNode
7  {
8    public:
9      Directory(std::string name);
10     virtual ~Directory();
11
```

```
12
13    protected:
14    private:
15  };
16
17  #endif // DIRECTORY_H
```

File.h

```
1   #ifndef FILE_H
2   #define FILE_H
3
4   #include "FSNode.h"
5
6   class File : public FSNode
7   {
8     public:
9       File(std::string name);
10      virtual ~File();
11
12      virtual void SetFirstChild(Node *node) override;
13    protected:
14    private:
15  };
16
17  #endif // FILE_H
```

FileSystem.h

```
1   #ifndef FILESYSTEM_H
2   #define FILESYSTEM_H
3
4   #include "Tree.h"
5   #include "Directory.h"
6
7   class FileSystem : public Tree
8   {
9     public:
10      FileSystem();
11      virtual ~FileSystem();
12
13      // create new file
14      virtual void Touch(const std::string &path, const std::string &filename);
15      // create new directory
16      virtual void Mkdir(const std::string &path, const std::string &dirname);
17      // remove file
18      virtual void Rm(const std::string &path, const std::string &filename);
19      // remove directory
20      virtual void Rmdir(const std::string &path, const std::string &dirname);
21      // list file system contents
22      virtual void Ls() const;
23    protected:
24      virtual Directory* FindNodeByPath(const std::string &path) const;
25      virtual FSNode* FindNodeInDirectory(const Directory* dir, const std::string &filename) const;
26    private:
27  };
```

```
28
29 #endif // FILESYSTEM_H
```

Node.cpp

```
1  #include <ostream>
2  #include <MLObject.h>
3  #include "Node.h"
4
5  using namespace std;
6  using namespace ML;
7
8  Node::Node(Node* firstChild, Node* nextSibling)
9    :firstChild(firstChild), nextSibling(nextSibling) {
10
11   //Register on minilib
12   Object::Register("Node","Object");
13 }
14
15 Node::~Node() {
16   if(firstChild != nullptr) delete firstChild;
17   if(nextSibling != nullptr) delete nextSibling;
18 }
19
20 void Node::Print(std::ostream& os) const {
21   os << AsString();
22 }
```

IntNode.cpp

```
1  #include <MLObject.h>
2  #include <sstream>
3  #include "IntNode.h"
4
5  using namespace ML;
6  using namespace std;
7
8  IntNode::IntNode(int value, Node* firstChild, Node* nextSibling)
9    :Node(firstChild, nextSibling), value(value) {
10
11   Object::Register("IntNode","Node");
12 }
13
14
15 IntNode::~IntNode() { /* nothing todo */}
16
17 std::string IntNode::AsString() const {
18   stringstream ss;
19   ss << "Int("<< value << ")";
20   return ss.str();
21
22 }
23
24 IntNode *IntNode::Clone() const {
25   Node *newFirstChild = nullptr, *newNextSibling = nullptr;
26   if(firstChild != nullptr)
27     newFirstChild = firstChild->Clone();
```

```
28
29    if(nextSibling != nullptr)
30      newNextSibling = nextSibling->Clone();
31
32    return new IntNode(value,newFirstChild, newNextSibling);
33  }
```

Tree.cpp

```
1   #include "Tree.h"
2
3   #include <ostream>
4   #include <sstream>
5   #include <cassert>
6   #include <MLObject.h>
7
8   using namespace std;
9   using namespace ML;
10
11  Tree::Tree() : root(nullptr) {
12    Object::Register("Tree", "Object");
13  }
14
15  Tree::Tree(const Tree &tree):Tree() {
16    if(tree.root !=nullptr) {
17      root = tree.root->Clone();
18    }
19  }
20
21  Tree::~Tree() {
22    DeleteElements();
23  }
24
25  Tree &Tree::operator=(const Tree& other) {
26    if (this == &other) return *this;
27
28    DeleteElements();
29    if(other.root != nullptr ) {
30      root = other.GetRoot()->Clone();
31    }
32    return *this;
33  }
34
35  Node *Tree::GetRoot() const {
36    return root;
37  }
38
39  bool Tree::IsTreeNode(Node* startNode, Node* node) const {
40    bool result = false;
41    if(startNode == node) return true;
42    if(startNode->GetFirstChild() != nullptr)
43      result = IsTreeNode(startNode->GetFirstChild(), node);
44    if(!result && startNode->GetNextSibling() != nullptr)
45      result = IsTreeNode(startNode->GetNextSibling(), node);
46    return result;
47  }
48
49  Node *Tree::GetParent(Node *startNode, Node* node) const {
```

```
50   if(startNode ==nullptr) return nullptr;
51
52   Node* result = nullptr, *child ;
53   child = startNode->GetFirstChild();
54   while(child != nullptr) {
55     if(child == node) {
56       result = startNode;
57       break;
58     }
59     else {
60       child = child->GetNextSibling();
61     }
62   }
63   if (result == nullptr) {
64     result = GetParent(startNode->GetNextSibling(), node);
65   }
66   if( result == nullptr) {
67     result = GetParent(startNode->GetFirstChild(), node);
68   }
69   return result;
70
71 }
72
73
74 void Tree::InsertChild(Node* parent, Node* child) {
75   assert(child != nullptr);
76
77   //parent node nullptr means we are setting the root
78   if(parent == nullptr) {
79
80     if(root != nullptr) {
81       cerr << "Root cannot be replaced, use clear or deleteElements first" << endl;
82     }
83     else {
84       if(child->GetNextSibling() != nullptr) {
85         cerr << "Root node must not contain siblings" << endl;
86         return;
87       }
88       root = child;
89     }
90     return;
91   }
92
93   //check if the parent is inside the tree
94   if(!IsTreeNode(root, parent)) {
95     cerr << "Parent node " << (*parent) << " is not part of the tree" << endl;
96     return;
97   }
98
99   //Add the node as first child
100  if(parent->GetFirstChild() == nullptr) {
101    parent->SetFirstChild(child);
102  }
103  else { //add a new first child and move the current to the next sibling
104    child->SetNextSibling(parent->GetFirstChild());
105    parent->SetFirstChild(child);
106  }
107
```

```
108 }
109
110 void Tree::DeleteSubtree(Node* node) {
111   if(!IsTreeNode(root,node)) {
112     cerr << "Node " << (*node) << " is not part of the tree" << endl;
113     return;
114   }
115   Node *parent = GetParent(root, node);
116   //root node
117   if(parent == nullptr) {
118     delete node;
119     root = nullptr;
120   }
121   else {
122     Node* lastSibling = nullptr;
123     Node* currentSibling = parent->GetFirstChild();
124     while(currentSibling != nullptr && currentSibling != node) {
125       lastSibling = currentSibling;
126       currentSibling = currentSibling->GetNextSibling();
127     }
128     if(lastSibling == nullptr) {
129       parent->SetFirstChild(node->GetNextSibling());
130     }
131     else {
132       lastSibling->SetNextSibling(currentSibling->GetNextSibling());
133     }
134     node->SetNextSibling(nullptr);
135     delete node;
136   }
137
138 }
139
140
141 void Tree::Clear() {
142   root = nullptr;
143 }
144
145 void Tree::DeleteElements() {
146   if(root != nullptr) {
147     delete root;
148     root = nullptr;
149   }
150 }
151
152 void Tree::Print(std::ostream& os) const {
153   if(GetSize() == 0 ) {
154    os << "Tree is empty" << endl;
155   }
156   else {
157     int level = 0;
158
159     os << (*root) << endl;
160     Node *current = root->GetFirstChild();
161     level++;
162     while(current != nullptr) {
163       os << string(level * 2,' ');
164       os << (*current) << endl;
165       if(current->GetFirstChild() != nullptr) {
```

```
166         current = current->GetFirstChild();
167         level++;
168       }
169       else if(current->GetNextSibling() != nullptr) {
170         current = current->GetNextSibling();
171       }
172       else {
173         while(current != nullptr && current->GetNextSibling() == nullptr) {
174           current = GetParent(root, current);
175           level --;
176         }
177         if(current != nullptr) {
178           current = current->GetNextSibling();
179         }
180       }
181     }
182
183   }
184
185 }
186
187 std::string Tree::AsString() const {
188   stringstream ss;
189   Print(ss);
190   return ss.str();
191
192 }
193
194 /* Helper function for counting nodes */
195 void CountNodes (Node *start, int& count) {
196   count++;
197   if(start->GetFirstChild() != nullptr) CountNodes(start->GetFirstChild(), count);
198   if(start->GetNextSibling() != nullptr) CountNodes(start->GetNextSibling(), count);
199
200 }
201
202 int Tree::GetSize() const {
203   if(root == nullptr) return 0;
204
205   int count=0;
206   CountNodes(root, count);
207   return count;
208 }
```

FSNode.cpp

```
1  #include "FSNode.h"
2
3  using namespace std;
4
5  FSNode::FSNode(string name, Node* firstChild, Node* nextSibling)
6    :Node(firstChild, nextSibling), name(name) {
7
8    //Register on minilib
9    Object::Register("FSNode","Node");
10 }
11
12 FSNode::~FSNode(){/* nothing todo */}
```

```
13
14 std::string FSNode::AsString() const{
15   return name;
16 }
17
18 FSNode *FSNode::Clone() const {
19   Node *newFirstChild = nullptr, *newNextSibling = nullptr;
20   if(firstChild != nullptr)
21     newFirstChild = firstChild->Clone();
22
23   if(nextSibling != nullptr)
24     newNextSibling = nextSibling->Clone();
25
26   return new FSNode(name,newFirstChild, newNextSibling);
27 }
```

Directory.cpp

```
1 #include "Directory.h"
2
3 #include <MLObject.h>
4
5 using namespace ML;
6 using namespace std;
7
8 Directory::Directory(string name) :FSNode(name) {
9   //Register on minilib
10  Object::Register("Directory","FSNode");
11 }
12
13 Directory::~Directory(){/* nothing todo */}
```

File.cpp

```
1 #include "File.h"
2
3 #include <MLObject.h>
4
5 using namespace ML;
6 using namespace std;
7
8 File::File(string name) :FSNode(name) {
9   Object::Register("File","FSNode");
10 }
11
12 File::~File() {/* nothing todo */}
13
14 void File::SetFirstChild(Node* node) {
15   cerr << "File cannot have sub entries" << endl;
16 }
```

FileSystem.cpp

```
1 #include "FileSystem.h"
2
3 #include <MLObject.h>
```

```
4  #include <iostream>
5  #include <cstring>
6  #include "Directory.h"
7  #include "File.h"
8
9  using namespace std;
10 using namespace ML;
11
12 FileSystem::FileSystem() {
13   Object::Register("FileSystem","Tree");
14 }
15
16 FileSystem::~FileSystem() {
17   /* nothing todo */
18 }
19
20 Directory* FileSystem::FindNodeByPath(const std::string& path) const{
21   char *input = (char *) path.c_str();
22   const char *part;
23   Node *currentNode = root;
24
25   if(currentNode == nullptr) return nullptr; //filesystem empty
26
27   //no path, return root
28   if(path.empty())
29     return dynamic_cast<Directory*>(currentNode);
30
31   part = strtok (input,"/");
32   if(part == NULL) {
33     //no path splitter, only root
34     Directory *fsnode = dynamic_cast<Directory*>(currentNode);
35     if(fsnode == nullptr) {
36       cerr << "Invalid node type " << currentNode->Class() << " in file system" << endl;
37       return nullptr;
38     }
39
40     if(fsnode->GetName() == path) {
41       return fsnode; //path found
42     }
43
44     return nullptr; //path not found
45
46   }
47
48   Directory *fsnode = dynamic_cast<Directory*>(currentNode);
49   while (part != NULL)
50   {
51     //not a directory, error has to be printed on calling function
52     if(fsnode == nullptr) {
53       return nullptr;
54     }
55
56     while(fsnode != nullptr && fsnode->GetName() != string(part)) {
57       fsnode = dynamic_cast<Directory*>(fsnode->GetNextSibling());
58     }
59
60     if(fsnode == nullptr) { //path not found
61       return nullptr;
```

```
62      }
63
64      //check next path splitter
65      part = strtok (NULL, "/");
66      if(part != nullptr) {
67        fsnode = dynamic_cast<Directory*>(fsnode->GetFirstChild());
68      }
69    }
70    return fsnode;
71  }
72
73  FSNode *FileSystem::FindNodeInDirectory(const Directory* dir,const std::string& filename) const {
74    FSNode *file = dynamic_cast<FSNode*>(dir->GetFirstChild());
75    while(file !=nullptr && file->GetName() != filename) {
76      file = dynamic_cast<FSNode*>(file->GetNextSibling());
77    }
78    return file;
79  }
80
81
82  void FileSystem::Touch(const std::string& path, const std::string& filename) {
83    Directory *dir = FindNodeByPath(path);
84    if(dir == nullptr) {
85      cerr << "Directory " << path << " not found" << endl;
86      return;
87    }
88    //Verify that there is no file/dir with same name
89    FSNode *file = FindNodeInDirectory(dir, filename);
90    if(file != nullptr) {
91      cerr << "File/Directory " << filename << " already exists in directory " << path << endl;
92      return;
93    }
94    InsertChild(dir, new File(filename));
95  }
96
97  void FileSystem::Mkdir(const std::string& path, const std::string& dirname) {
98    if(path.empty()) {
99      if(root == nullptr) {
100       root=new Directory(dirname);
101     }
102     else {
103       cerr << "Directory " << (*root) << " cannot be replaced" << endl;
104     }
105   }
106   else {
107     Directory *dir = FindNodeByPath(path);
108     if(dir == nullptr) {
109       cerr << "Directory " << path << " not found" << endl;
110       return;
111     }
112
113     //Check if there is already a file/directory
114     FSNode *file = FindNodeInDirectory(dir, dirname);
115     if(file != nullptr) {
116       cerr << "File/Directory " << dirname << " already exists in directory " << path << endl;
117       return;
118     }
119
```

```
120     InsertChild(dir, new Directory(dirname));
121   }
122 }
123
124 void FileSystem::Rm(const std::string& path, const std::string& filename) {
125   Directory *dir = FindNodeByPath(path);
126   if(dir == nullptr) {
127     cerr << "Directory " << path << " not found" << endl;
128     return;
129   }
130
131   FSNode *file = FindNodeInDirectory(dir, filename);
132
133   if(file == nullptr) {
134     cerr << "File " << filename << " not found in directory " << path << endl;
135     return;
136   }
137
138   if(!file->IsA("File")) {
139     cerr << filename << " is no regular file" << endl;
140     return;
141   }
142
143   DeleteSubtree(file);
144 }
145
146 void FileSystem::Rmdir(const std::string& path, const std::string& dirname) {
147     Directory *dir = FindNodeByPath(path + "/" + dirname);
148     if(dir == nullptr) {
149       cerr << "Directory " << dirname << " not found in path " << path << endl;
150       return;
151     }
152     if(dir->GetFirstChild() != nullptr) {
153       cerr << "Directory " << (path + "/" + dirname) << " not empty and cannot be removed" << endl;
154       return;
155     }
156
157     DeleteSubtree(dir);
158 }
159
160 void FileSystem::Ls() const
161 {
162   //check if filesystem is empty to suppress
163   //tree is empty message
164   if( root == nullptr )
165     cout << "No files or directories" << endl;
166   else
167     Print(cout);
168 }
```

main.cpp

```
1 #include <iostream>
2 #include "Tree.h"
3 #include "IntNode.h"
4 #include <MLObject.h>
5 #include "FileSystem.h"
6
```

```
7  using namespace std;
8
9  int main(int argc, char** argv)
10 {
11     if(argc != 2) {
12       cerr << "Wrong parameter count" << endl;
13       cerr << "Usage: " << argv[0] << " testcase";
14       return 0;
15     }
16
17     IntNode *root = new IntNode(1);
18     Tree *tree = new Tree();
19     tree->InsertChild(nullptr, root);
20     tree->InsertChild(root, new IntNode(2));
21     tree->InsertChild(root, new IntNode(3));
22     tree->InsertChild(root, new IntNode(4));
23
24     tree->InsertChild(root->GetFirstChild()->GetNextSibling(), new IntNode(7));
25     tree->InsertChild(root->GetFirstChild()->GetNextSibling(), new IntNode(8));
26     tree->InsertChild(root->GetFirstChild()->GetNextSibling(), new IntNode(9));
27     tree->InsertChild(root->GetFirstChild()->GetNextSibling(), new IntNode(10));
28
29     tree->InsertChild(root->GetFirstChild()->GetNextSibling()->GetNextSibling(), new IntNode(5));
30     tree->InsertChild(root->GetFirstChild()->GetNextSibling()->GetNextSibling(), new IntNode(6));
31
32     int testcase = atoi(argv[1]);
33     switch(testcase) {
34       case 1:
35         cout << "Example Tree and GetSize:" << endl;
36         cout << "Size (NodeCount): " << tree->GetSize() << endl;
37         cout << (*tree) << endl;
38         break;
39
40       case 2:
41         cout << "Delete subtree (3):" << endl;
42         cout << "Size before: " << tree->GetSize() << endl;
43         tree->DeleteSubtree(root->GetFirstChild()->GetNextSibling());
44         cout << "Size after: " << tree->GetSize() << endl;
45         cout << (*tree) << endl;
46         break;
47
48       case 3:
49         cout << "DeleteElements:" << endl;
50         cout << "Size before: " << tree->GetSize() << endl;
51         tree->DeleteElements();
52         cout << "Size after: " << tree->GetSize() << endl;
53         cout << "Nodes already deleted" << endl;
54         WriteMetaInfo(cout);
55         break;
56
57       case 4:
58         cout << "Clear:" << endl;
59         cout << "Size before: " << tree->GetSize() << endl;
60         tree->Clear();
61         cout << "Size after: " << tree->GetSize() << endl;
62         cout << "Nodes not deleted" << endl;
63         WriteMetaInfo(cout);
64         delete root;
```

```
65          break;
66
67        case 5:
68          cout << "Invalid operations:" << endl;
69          tree->InsertChild(nullptr, root);
70          //Use a block here to define the new node
71          {
72            IntNode *newNode = new IntNode(11);
73            tree->InsertChild(newNode, root);
74            tree->DeleteSubtree(newNode);
75            delete newNode;
76          }
77          break;
78
79        case 6:
80          cout << "Copy constructor and assignment operator:" << endl;
81          {
82            Tree newTree = *tree;
83            //delete old tree elements to check copy of the nodes
84            tree->DeleteElements();
85            cout << "Copy tree size: " << newTree.GetSize() << endl;
86            cout << newTree << endl;
87            Tree assigmentTree;
88            assigmentTree.InsertChild(nullptr, new IntNode(20));
89            assigmentTree = newTree;
90            //delete newTree to check assigment copy of the nodes
91            newTree.DeleteElements();
92
93            cout << "Assignment tree size: " << assigmentTree.GetSize() << endl;
94            cout << assigmentTree << endl;
95          }
96          break;
97
98        case 7:
99          cout << "Filesystem:" << endl;
100         {
101           FileSystem *fs = new FileSystem();
102           fs->Mkdir("", "root");
103           fs->Touch("root", "ReadMe.txt");
104           fs->Mkdir("root", "bin");
105           fs->Touch("root/bin", "a.exe");
106           fs->Touch("root/bin", "b.exe");
107           fs->Ls();
108           fs->Rm("root/bin", "a.exe");
109           fs->Rmdir("", "root");
110           cout << endl;
111           fs->Ls();
112           fs->Rm("root/bin", "b.exe");
113           fs->Rmdir("", "root/bin");
114           fs->Rm("root", "ReadMe.txt");
115           fs->Rmdir("", "root");
116           cout << endl;
117           fs->Ls();
118           delete fs;
119
120         }
121         break;
122
```

```
123       case 8:
124         cout << "Filesystem invalid operations:" << endl;
125         {
126           FileSystem *fs = new FileSystem();
127           fs->Mkdir("", "root");
128           fs->Mkdir("root", "Folder");
129           //file already exists
130           fs->Mkdir("root", "Folder");
131           //file already exists
132           fs->Touch("root", "Folder");
133 
134           fs->Touch("root", "ReadMe.txt");
135           //root cannot be replaced
136           fs->Mkdir("", "root1");
137           //not a directory
138           fs->Rmdir("root","ReadMe.txt");
139           //not a file
140           fs->Rm("root","Folder");
141           delete fs;
142 
143         }
144         break;
145     }
146 
147     //delete tree from above
148     delete tree;
149     WriteMetaInfo(cout);
150     return 0;
151 
152 
153 }
```

2.4 Testfälle

2.4.1 Testfall 1 - Beispielgraph und GetSize

```
romanlum@ubuntu: ~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$ ./Tree 1
Example Tree and GetSize:
Size (NodeCount): 10
Int(1)
  Int(4)
  Int(3)
    Int(10)
    Int(9)
    Int(8)
    Int(7)
  Int(2)
    Int(6)
    Int(5)



============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      10 |      10 |           0
   Tree                 |       1 |       1 |           0
------------------------+---------+---------+-------------
 Number of classes:  4  | Summary: all objects deleted
============================================================

romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$
```

2.4.2 Testfall 2 - Methode DeleteSubtree

```
romanlum@ubuntu: ~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$ ./Tree 2
Delete subtree (3):
Size before: 10
Size after: 5
Int(1)
  Int(4)
  Int(2)
    Int(6)
    Int(5)



============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      10 |      10 |           0
   Tree                 |       1 |       1 |           0
------------------------+---------+---------+-------------
 Number of classes:  4  | Summary: all objects deleted
============================================================
```

2.4.3 Testfall 3 - Methode DeleteElements

```
romanlum@ubuntu: ~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug

romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$ ./Tree 3
DeleteElements:
Size before: 10
Size after: 0
Nodes already deleted


============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      10 |      10 |           0
   Tree                 |       1 |       0 |           1
------------------------+---------+---------+-------------
 Number of classes:  4  | Summary: 1 object(s) still alive
============================================================



============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      10 |      10 |           0
   Tree                 |       1 |       1 |           0
------------------------+---------+---------+-------------
 Number of classes:  4  | Summary: all objects deleted
============================================================

romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$
```

2.4.4 Testfall 4 - Methode Clear

```
romanlum@ubuntu: ~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$ ./Tree 4
Clear:
Size before: 10
Size after: 0
Nodes not deleted


============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      10 |       0 |          10
   Tree                 |       1 |       0 |           1
------------------------+---------+---------+-------------
 Number of classes:  4  | Summary: 11 object(s) still alive
============================================================



============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      10 |      10 |           0
   Tree                 |       1 |       1 |           0
------------------------+---------+---------+-------------
 Number of classes:  4  | Summary: all objects deleted
============================================================

romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$
```

2.4.5 Testfall 5 - Tree - ungültige Operationen

```
romanlum@ubuntu: ~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$ ./Tree 5
Invalid operations:
Root cannot be replaced, use clear or deleteElements first
Parent node Int(11) is not part of the tree
Node Int(11) is not part of the tree


============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      11 |      11 |           0
   Tree                 |       1 |       1 |           0
------------------------+---------+---------+-------------
 Number of classes:  4  | Summary: all objects deleted
============================================================

romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$
```

2.4.6 Testfall 6 - Tree - Kopierkonstruktor und Zuweisungsoperator

```
romanlum@ubuntu: ~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug
Copy constructor and assignment operator:
Copy tree size: 10
Int(1)
  Int(4)
  Int(3)
    Int(10)
    Int(9)
    Int(8)
    Int(7)
  Int(2)
    Int(6)
    Int(5)

Assignment tree size: 10
Int(1)
  Int(4)
  Int(3)
    Int(10)
    Int(9)
    Int(8)
    Int(7)
  Int(2)
    Int(6)
    Int(5)



============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      31 |      31 |           0
   Tree                 |       1 |       1 |           0
------------------------+---------+---------+-------------
 Number of classes:  4  | Summary: all objects deleted
============================================================

romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$
```

2.4.7 Testfall 7 - Dateisystem

```
romanlum@ubuntu: ~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$ ./Tree 7
Filesystem:
root
  bin
    b.exe
    a.exe
  ReadMe.txt
Directory /root not empty and cannot be removed

root
  bin
    b.exe
  ReadMe.txt

No files or directories


============================================================
 Meta information for MiniLib application
-----------------------+------------------------------------
 Class hierarchy       | Number of dynamic objects
                       +---------+---------+-----------------
                       | created | deleted | still alive
-----------------------+---------+---------+-----------------
 Object                |       0 |       0 |           0
   Node                |       0 |       0 |           0
     IntNode           |      10 |      10 |           0
     FSNode            |       0 |       0 |           0
       Directory       |       2 |       2 |           0
       File            |       3 |       3 |           0
   Tree                |       1 |       1 |           0
     FileSystem        |       1 |       1 |           0
-----------------------+---------+---------+-----------------
 Number of classes:  8 | Summary: all objects deleted
============================================================

romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$
```

2.4.8 Testfall 8 - Dateisystem - ungültige Operationen

```
romanlum@ubuntu: ~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug
romanlum@ubuntu:~/swo3/UebungMoodle6/Beispiel/Tree/bin/Debug$ ./Tree 8
Filesystem invalid operations:
File/Directory Folder already exists in directory root
File/Directory Folder already exists in directory root
Directory root cannot be replaced
Directory ReadMe.txt not found in path root
Folder is no regular file


============================================================
 Meta information for MiniLib application
------------------------+-----------------------------------
 Class hierarchy        | Number of dynamic objects
                        +---------+---------+-------------
                        | created | deleted | still alive
------------------------+---------+---------+-------------
 Object                 |       0 |       0 |           0
   Node                 |       0 |       0 |           0
     IntNode            |      10 |      10 |           0
     FSNode             |       0 |       0 |           0
       Directory        |       2 |       2 |           0
       File             |       1 |       1 |           0
   Tree                 |       1 |       1 |           0
     FileSystem         |       1 |       1 |           0
------------------------+---------+---------+-------------
 Number of classes:  8  | Summary: all objects deleted
```