

Labs EmbSW2

Michael Schmid

July 24, 2020

Contents

| | | |
|-----------|--|-----------|
| 1 | Lab 1 Listen | 2 |
| 1.1 | Aufgabe 1: Realisierung einer einfach verketteten Liste | 2 |
| 1.2 | Aufgabe 2: Liste für beliebige Typen | 4 |
| 1.3 | Aufgabe 3: Realisierung einer doppelt verketteten Liste | 5 |
| 1.4 | Aufgabe 4: Gemeinsame Basisklasse | 6 |
| 1.5 | Aufgabe 5: Gemeinsame Basisklasse mit Templates | 9 |
| 2 | Lab 2 Listen, Algorithmen und Komplexitätstheorie | 13 |
| 2.1 | Aufgabe 1: Klasse für die Speicherung von Messwerten | 13 |
| 2.2 | Aufgabe 2: Komplexitätsbetrachtungen | 17 |
| 2.3 | Aufgabe 3: Implementation eines dynamischen Stacks mit Hilfe einer verketteten Liste | 17 |
| 3 | Lab 3 Sortialgorithmen und Komplexitätstheorie | 21 |
| 3.1 | Aufgabe 1: Zeitmessung bei Fibonacci-Implementationen | 21 |
| 3.2 | Aufgabe 2: Klasse für Stoppuhr | 22 |
| 3.3 | Aufgabe 3: Komplexität | 23 |
| 3.4 | Aufgabe 4: Sortialgorithmen | 25 |
| 4 | Lab 4 Zufallszahlengeneratoren | 28 |
| 4.1 | Aufgabe 1: Verteilung der Zufallswerte durch <code>rand()</code> | 28 |
| 4.2 | Aufgabe 2: Güte von Zufallszahlengeneratoren | 29 |
| 4.3 | Aufgabe 3: Nicht ideale Münze (biased coin) | 32 |
| 5 | Lab 5 Dinierende Philosophen | 34 |
| 5.1 | Aufgabe 1: Untersuchung von Deadlocks | 34 |
| 5.2 | Aufgabe 2: Implementation der dinierenden Philosophen | 35 |
| 6 | Lab 6 CRC - Berechnung und -Implementation in C++ | 40 |
| 6.1 | Aufgabe 1: CRC-8 Berechnungsbeispiel (Papierübung) | 40 |
| 6.2 | Aufgabe 2: Verifikation empfangener Daten mittels CRC | 40 |
| 6.3 | Aufgabe 3: C++ - Implementation | 41 |
| 6.4 | Aufgabe 4: C++ - Implementation mit einer Lookup Tabelle | 43 |
| 6.5 | Aufgabe 5: C++ - Implementation mit einer Lookup Tabelle | 44 |
| 7 | Lab 7 inline und Bitfelder | 47 |
| 7.1 | Aufgabe 1: inline-Methoden | 47 |
| 7.2 | Aufgabe 2: Bitfelder | 50 |
| 7.3 | Aufgabe 3: Bitmasken | 52 |
| 8 | Lab 8 Digitale Ableitung und Pulsdetektion auf dem EmbSW-Roboter | 53 |
| 8.1 | Aufgabe 1: Implementation des FIR-Filters | 54 |
| 8.2 | Aufgabe 2: Implementation der Pulsdetektion | 59 |
| 9 | Lab 9 Code Bloat bei Templates, Code Hoisting | 61 |
| 9.1 | Aufgabe 1: Untersuchung von Assemblercode bei Templates | 61 |
| 9.2 | Aufgabe 2: Verhindern von Code Bloat durch Code Hoisting | 64 |
| 10 | Lab 10 Dynamic Memory Management | 67 |
| 10.1 | Aufgabe 1: Fixed-size Pool | 67 |
| 10.2 | Aufgabe 2: Block Allocator | 70 |

| | |
|---|-----------|
| 11 Lab 11 C++ and ROMability, Hardware Abstraction Layer (HAL) | 73 |
| 11.1 Aufgabe 1: C++ and ROMability, Hardware Abstraction Layer (HAL) | 73 |
| 11.2 Aufgabe 2: Ohne Hardware Abstraction Layer | 78 |
| 11.3 Aufgabe 3: Hardware Abstraction Layer in C | 79 |
| 11.4 Aufgabe 4: Hardware Abstraction Layer in C++ | 84 |
| 11.5 Aufgabe 5: Hardware Abstraction Layer in C++ mit Memory-mapped IO (MMIO) | 86 |
| 12 Lab 12 Zufallszahlengeneratoren | 90 |
| 12.1 Aufgabe 1: Wahrscheinlichkeitsverteilungen | 90 |
| 13 Lab 13 Security | 93 |
| 13.1 Aufgabe 1: Hashing | 93 |
| 13.2 Aufgabe 1: Verschlüsselten Text entschlüsseln | 93 |

1 Lab 1 Listen

1.1 Aufgabe 1: Realisierung einer einfach verketteten Liste

Realisierung einer einfach verketteten Liste Implementieren Sie eine einfach verkettete Liste für double-Werte in C++ mit der Entwicklungsumgebung Eclipse. Die Listenklasse erhält den Namen SList. SList soll die nachfolgenden Methoden anbieten, wobei die geeigneten Parameter noch zu definieren sind. Achten Sie auf den korrekten Einsatz von const.

- Nehmen Sie als Ausgangspunkt das vorgegebene Eclipse-Projekt. Ergänzen Sie bei Bedarf auch das einfache Testprogramm.
- Konstruktor (Ctor) Initialisiert eine leere Liste.
- Destruktor (Dtor) Löscht die gesamte Liste und gibt den allozierten Speicher wieder frei.
- insertAt Fügt ein Element an (nach) der bezeichneten Position ein. Einfügen am Anfang der Liste kann mittels Position 0 bewerkstelligt werden.
- deleteAt Löscht das Element an der bezeichneten Position.
- search Sucht das erste Element in der Liste mit dem definierten Wert und gibt die Position zurück. Falls das Element nicht in der Liste vorhanden ist, wird als Position der Wert 0 zurückgegeben.
- isEmpty Gibt true zurück, falls die Liste leer ist.
- getNumber Gibt die Anzahl der Elemente der Liste zurück.
- getValue Gibt den Wert des Elementes an der bezeichneten Position zurück.
- setValue Setzt den Wert des Elementes an der bezeichneten Position.
- print Schreibt den Inhalt der Liste in die Console.

1.1.1 Lösung

```

1  /*
2  * SList.h
3  *
4  * Singly-linked list for doubles
5  *
6  * Created on: 12.02.2016
7  * Author: rbondere
8  */
9
10 #ifndef SLIST_H_
11 #define SLIST_H_
12
13 class SList
14 {
15 public:
16     SList();
17     ~SList();
18
19     void insertAt(int pos, double val);
20     // inserts element at (after) position pos (0: at head)
21
22     void deleteAt(int pos);
23     // deletes element at position pos (>0)
24
25     int search(double val) const;
26     // searches val in list and returns position of first
27     // ↳ match, starting
28     // at head
29     // returns 0 if value is not found
30
31     bool isEmpty() const;
32     // returns true if list is empty, else false
33
34     int getNumber() const;
35     // returns number of elements
36
37     double getValue(int pos) const;
38     // returns value at position pos
39
40     void setValue(int pos, double val);
41     // sets value val at position pos
42
43     void print() const;
44     // prints content of list to console
45
46 private:
47     struct Node
48     {
49         double value;
50         Node* next;
51     };
52     Node* pHead; // ptr to head of list
53     int nr; // number of Elements
54
55     Node* nodePtr(int pos) const;
56     // returns a pointer to the node given by position pos
57 };
58 #endif /* SLIST_H_ */

```

```

1  /*
2   * SList.cpp
3   *
4   * Created on: 17.02.2020
5   * Author: rbondere
6   */
7
8  #include <cassert>
9  #include <iostream>
10 #include "SList.h"
11 using namespace std;
12
13 SList::SList() :
14     pHead(nullptr), nr(0)
15 {
16 }
17
18 SList::~SList()
19 {
20     for (Node* p = pHead; p; p = pHead)
21     {
22         pHead = pHead->next;
23         delete p;
24     }
25 }
26
27 void SList::insertAt(int pos, double val)
28 {
29     assert(pos >= 0);
30     Node* pEl = new Node;
31     pEl->value = val;
32     if (pos != 0)
33     {
34         Node* p = nodePtr(pos);
35         assert(p != nullptr);
36         pEl->next = p->next;
37         p->next = pEl;
38     }
39     else // insert at head
40     {
41         pEl->next = pHead;
42         pHead = pEl;
43     }
44     ++nr;
45 }
46
47 void SList::deleteAt(int pos)
48 {
49     assert(pos > 0 && pos <= nr);
50     Node* p = pHead; // cursor
51     Node* pDel = p; // node to be deleted
52     if (pos == 1) // first element
53         pHead = pHead->next;
54     else
55     {
56         for (int i = 1; i < pos - 1; ++i)
57         {
58             p = p->next;
59         }
60         pDel = p->next;
61         p->next = pDel->next;
62     }
63     delete pDel;
64     --nr;
65 }
66
67 int SList::search(double val) const
68 {
69     Node* p = pHead; // p points to element at pos 1, if not
70                     // empty
71     for (int i = 1; p; ++i)
72     {
73         if (p->value == val)
74             return i;
75         p = p->next;
76     }
77     return 0; // not found
78 }
79
80 bool SList::isEmpty() const
81 {
82     return nr == 0;
83 }
84
85 int SList::getNumber() const
86 {
87     return nr;
88 }
89
90 double SList::getValue(int pos) const
91 {
92     assert(pos > 0 && pos <= nr);
93     return nodePtr(pos)->value;
94 }
95
96 void SList::setValue(int pos, double val)
97 {
98     assert(pos > 0 && pos <= nr);
99     nodePtr(pos)->value = val;
100 }
101
102 void SList::print() const
103 {
104     cout << "-----" << endl;
105     cout << "Number of elements: " << nr << endl;
106     cout << "Content of list:" << endl;
107     for (Node* p = pHead; p; p = p->next)
108         cout << p->value << " ";
109     cout << endl;
110 }
111
112 SList::Node* SList::nodePtr(int pos) const
113 {
114     assert(pos > 0 && pos <= nr);
115     Node* p = pHead; // p points to element at pos 1, if not
116                     // empty
117     for (int i = 1; p && i < pos; ++i)
118     {
119         p = p->next;
120     }
121     return p;
122 }

```

```

1  //=====
2  // Name      : Lists.cpp
3  // Author    : Reto Bonderer
4  // Version   :
5  // Copyright : (c) HSR R. Bonderer
6  // Description : List implementations
7  //=====
8
9  #include <iostream>
10 #include "SList.h"
11 #include "DList.h"
12 using namespace std;
13
14 int main()
15 {
16     SList s;
17     cout << "Singly-linked list" << endl;
18     s.print();
19
20     s.insertAt(0, 1.23);
21     s.insertAt(0, -34.6);
22     s.insertAt(0, 0.4);
23     s.print();
24
25     s.insertAt(3, 7.97);
26     s.insertAt(1, 88.9);
27
28     DList d;
29     cout << "Doubly-linked list" << endl;
30     d.print();
31
32     d.insertAt(0, 1.23);
33     d.insertAt(0, -34.6);
34     d.insertAt(0, 0.4);
35     d.print();
36
37     d.insertAt(3, 7.97);
38     d.insertAt(1, 88.9);
39     d.print();
40
41     d.deleteAt(5);
42     d.insertAt(4, 44.5);
43     d.print();
44
45     return 0;
46 }

```

1.2 Aufgabe 2: Liste für beliebige Typen

Erweitern Sie die Klasse SList so, dass beliebige Typen abgespeichert werden können. Achten Sie dabei auch auf die korrekte Verwendung von Referenzen und des const Qualifiers.

Zur Erinnerung: Sie müssen hier Templates verwenden.

1.2.1 Lösung

```
1  /*
2   * SList.h
3   *
4   * Singly-linked list for type T
5   *
6   * Created on: 17.02.2020
7   * Author: rbondere
8   */
9
10 #ifndef SLIST_H_
11 #define SLIST_H_
12
13 #include <cassert>
14 #include <iostream>
15
16 template<typename T>
17 class SList
18 {
19 public:
20     SList() :
21         pHead(nullptr), nr(0)
22     {
23     }
24
25     ~SList();
26
27     void insertAt(int pos, const T& val);
28     // inserts element at (after) position pos (0: at head)
29
30     void deleteAt(int pos);
31     // deletes element at position pos (>0)
32
33     int search(const T& val) const;
34     // searches val in list and returns position of first
35     //    ↪ match, starting
36     // at head
37     // returns 0 if value is not found
38
39     const T& getValue(int pos) const;
40     // returns value at position pos
41
42     void setValue(int pos, const T& val);
43     // sets value val at position pos
44
45     bool isEmpty() const
46     // returns true if list is empty, else false
47     {
48         return nr == 0;
49     }
50
51     int getNumber() const
52     // returns number of elements
53     {
54         return nr;
55     }
56
57     void print() const;
58     // prints content of list to console
59
60 private:
61     struct Node
62     {
63         T value;
64         Node* next;
65     };
66     Node* pHead; // ptr to head of list
67     int nr; // number of Elements
68
69     Node* nodePtr(int pos) const;
70     // returns a pointer to the node given by position pos
71 };
72
73 template<typename T>
74 SList<T>::~SList()
75 {
76     for (Node* p = pHead; p; p = pHead)
77     {
78         pHead = pHead->next;
79         delete p;
80     }
81 }
82
83 template<typename T>
84 void SList<T>::insertAt(int pos, const T& val)
85 {
86     assert(pos >= 0);
87     Node* pEl = new Node;
88     pEl->value = val;
89     if (pos != 0)
90     {
91         Node* p = nodePtr(pos);
92         assert(p != nullptr);
93         pEl->next = p->next;
94         p->next = pEl;
95     }
96     else // insert at head
97     {
98         pEl->next = pHead;
99         pHead = pEl;
100     }
101     ++nr;
102 }
103
104 template<typename T>
105 void SList<T>::deleteAt(int pos)
106 {
107     assert(pos > 0 && pos <= nr);
108     Node* p = pHead; // cursor
109     Node* pDel = p; // node to be deleted
110     if (pos == 1) // first element
111         pHead = pHead->next;
112     else
113     {
114         for (int i = 1; i < pos - 1; ++i)
115         {
116             p = p->next;
117         }
118         pDel = p->next;
119         p->next = pDel->next;
120     }
121     delete pDel;
122     --nr;
123 }
124
125 template<typename T>
126 int SList<T>::search(const T& val) const
127 {
128     Node* p = pHead; // p points to element at pos 1, if not
129     //    ↪ empty
130     for (int i = 1; p; ++i)
131     {
132         if (p->value == val)
133             return i;
134         p = p->next;
135     }
136     return 0; // not found
137 }
138
139 template<typename T>
140 const T& SList<T>::getValue(int pos) const
141 {
142     assert(pos > 0 && pos <= nr);
143     return nodePtr(pos)->value;
144 }
145
146 template<typename T>
147 void SList<T>::setValue(int pos, const T& val)
148 {
149     assert(pos > 0 && pos <= nr);
150     nodePtr(pos)->value = val;
151 }
152
153 template<typename T>
154 void SList<T>::print() const
155 {
156     std::cout << "-----" << std::endl;
157     std::cout << "Number of elements: " << nr << std::endl;
158     std::cout << "Content of list:" << std::endl;
159     for (Node* p = pHead; p; p = p->next)
```

```

159     std::cout << p->value << " ";
160     std::cout << std::endl;
161 }
162
163 template<typename T>
164 typename SList<T>::Node* SList<T>::nodePtr(int pos) const
165 {
166     assert(pos > 0 && pos <= nr);
167     Node* p = pHead; // p points to element at pos 1, if not

```

```

168     ↪ empty
169     for (int i = 1; p && i < pos; ++i)
170     {
171         p = p->next;
172     }
173     return p;
174 }
175 #endif /* SLIST_H_ */

```

```

1 //=====
2 // Name      : ListTest.cpp
3 // Author     : Reto Bonderer
4 // Version    :
5 // Copyright  : (c) HSR R. Bonderer
6 // Description: List implementations
7 //=====
8
9 #include <iostream>
10 #include "SList.h"
11 using namespace std;
12
13 int main()
14 {
15     SList<double> s;
16     cout << "Singly linked list" << endl;

```

```

17     s.print();
18
19     s.insertAt(0, 1.23);
20     s.insertAt(0, -34.6);
21     s.insertAt(0, 0.4);
22     s.print();
23
24     s.insertAt(3, 7.97);
25     s.insertAt(1, 88.9);
26     s.print();
27
28     s.deleteAt(2);
29     s.print();
30
31     return 0;
32 }

```

1.3 Aufgabe 3: Realisierung einer doppelt verketteten Liste

Implementieren Sie die Klasse `DList`, welche eine doppelt verkettete Liste für `double`-Werte in C++ definiert. `DList` soll dieselben Methoden anbieten wie bereits `SList` in Aufgabe 1. Bemerkung: hier müssen Sie keine Templates verwenden.

1.3.1 Lösung

```

1 /*
2  * DList.h
3  * Doubly linked list for doubles
4  *
5  * Created on: 12.02.2016
6  * Author: rbondere
7  */
8
9 #ifndef DLIST_H_
10 #define DLIST_H_
11
12 class DList
13 {
14 public:
15     DList();
16     ~DList();
17
18     void insertAt(int pos, double val);
19     // inserts element at (after) position pos (0: at head)
20
21     void deleteAt(int pos);
22     // deletes element at position pos (>0)
23
24     int search(double val) const;
25     // searches val in list and returns position of first
26     // ↪ match, starting
27     // at head
28     // returns 0 if value is not found
29
30     bool isEmpty() const;

```

```

30 // returns true if list is empty, else false
31
32 int getNumber() const;
33 // returns number of elements
34
35 double getValue(int pos) const;
36 // returns value at position pos
37
38 void setValue(int pos, double val);
39 // sets value val at position pos
40
41 void print() const;
42 // prints content of list to console
43
44 private:
45     struct Node
46     {
47         double value;
48         Node* next;
49         Node* prev;
50     };
51     Node* pHead; // ptr to head of list
52     int nr; // number of Elements
53
54     Node* nodePtr(int pos) const;
55     // returns a pointer to the node given by position pos
56 };
57
58 #endif /* DLIST_H_ */

```

```

1 /*
2  * DList.cpp
3  *
4  * Created on: 17.02.2020
5  * Author: rbondere
6  */
7
8 #include <cassert>
9 #include <iostream>
10 #include "DList.h"
11 using namespace std;
12
13 DList::DList() :
14     pHead(nullptr), nr(0)

```

```

15 {
16 }
17
18 DList::~DList()
19 {
20     for (Node* p = pHead; p; p = pHead)
21     {
22         pHead = pHead->next;
23         delete p;
24     }
25 }
26
27 void DList::insertAt(int pos, double val)
28 {

```

```

29  assert(pos >= 0);
30  Node* pEl = new Node;
31  pEl->value = val;
32  if (pos != 0)
33  {
34      Node* p = nodePtr(pos);
35      assert(p != nullptr);
36      pEl->next = p->next;
37      pEl->prev = p;
38      p->next = pEl;
39  }
40  else // insert at head
41  {
42      pEl->next = pHead;
43      pEl->prev = nullptr;
44      pHead = pEl;
45  }
46  if (pEl->next != nullptr) // not last element in list
47      pEl->next->prev = pEl;
48  ++nr;
49  }
50
51 void DList::deleteAt(int pos)
52 {
53     assert(pos > 0 && pos <= nr);
54     Node* pDel = nodePtr(pos); // node to be deleted
55     assert(pDel != nullptr);
56     if (pos == 1) // first element
57         pHead = pHead->next;
58     else
59     {
60         pDel->prev->next = pDel->next;
61     }
62     if (pDel->next != nullptr) // not last element in list
63         pDel->next->prev = pDel->prev;
64     delete pDel;
65     --nr;
66 }
67
68 int DList::search(double val) const
69 {
70     Node* p = pHead; // p points to element at pos 1, if not
71     ↪ empty
72     for (int i = 1; p; ++i)
73     {
74         if (p->value == val)
75             return i;
76         p = p->next;
77     }
78     return 0;
79 }

```

```

76     }
77     return 0; // not found
78 }
79
80 bool DList::isEmpty() const
81 {
82     return nr == 0;
83 }
84
85 int DList::getNumber() const
86 {
87     return nr;
88 }
89
90 double DList::getValue(int pos) const
91 {
92     assert(pos > 0 && pos <= nr);
93     return nodePtr(pos)->value;
94 }
95
96 void DList::setValue(int pos, double val)
97 {
98     assert(pos > 0 && pos <= nr);
99     nodePtr(pos)->value = val;
100 }
101
102 void DList::print() const
103 {
104     cout << "-----" << endl;
105     cout << "Number of elements: " << nr << endl;
106     cout << "Content of list:" << endl;
107     for (Node* p = pHead; p; p = p->next)
108         cout << p->value << " ";
109     cout << endl;
110 }
111
112 DList::Node* DList::nodePtr(int pos) const
113 {
114     assert(pos > 0 && pos <= nr);
115     Node* p = pHead; // p points to element at pos 1, if not
116     ↪ empty
117     for (int i = 1; p && i < pos; ++i)
118     {
119         p = p->next;
120     }
121     return p;
122 }

```

1.4 Aufgabe 4: Gemeinsame Basisklasse

Die beiden Listen SList und DList haben verschiedene Gemeinsamkeiten, eine gemeinsame Basisklasse List bietet sich offensichtlich an. Implementieren Sie eine gemeinsame Basisklasse List. Achten Sie darauf, dass die Sichtbarkeiten (public, private, protected) so gewählt werden, dass die Unterklassen Zugriff auf die von ihnen benötigten Daten haben. Es ist jedoch keine Lösung, einfach alles als public zu deklarieren.

1.4.1 Lösung

```

1  /*
2   * DList.h
3   * Double linked list for doubles
4   *
5   * Created on: 17.02.2020
6   * Author: rbondere
7   */
8
9  #ifndef DLIST_H_
10 #define DLIST_H_
11 #include "List.h"
12
13 class DList: public List
14 {
15 public:
16     DList();
17     virtual ~DList();
18
19     virtual void insertAt(int pos, double val);
20     // inserts element at (after) position pos (0: at head)
21
22     virtual void deleteAt(int pos);
23     // deletes element at position pos (>0)
24
25     virtual int search(double val) const;
26     // searches val in list and returns position of first
27     ↪ match, starting
28     // at head
29     // returns 0 if value is not found
30 }

```

```

29
30 int getPrev() const;
31 // returns position of previous element (0: no previous, i
32 ↪ .e. at head)
33
34 virtual double getValue(int pos) const;
35 // returns value at position pos
36
37 virtual void setValue(int pos, double val);
38 // sets value val at position pos
39
40 virtual void print() const;
41 // prints content of list to console
42
43 private:
44     struct Node
45     {
46         double value;
47         Node* next;
48         Node* prev;
49     };
50     Node* pHead; // ptr to head of list
51
52     Node* nodePtr(int pos) const;
53     // returns a pointer to the node given by position pos
54 };
55 #endif /* DLIST_H_ */

```

```

1  /*
2  * DList.cpp
3  *
4  * Created on: 17.02.2020
5  * Author: rbondere
6  */
7
8  #include <cassert>
9  #include <iostream>
10 #include "DList.h"
11 using namespace std;
12
13 DList::DList() :
14     List(), pHead(0)
15 {
16 }
17
18 DList::~DList()
19 {
20     for (Node* p = pHead; p; p = pHead)
21     {
22         pHead = pHead->next;
23         delete p;
24     }
25 }
26
27 void DList::insertAt(int pos, double val)
28 {
29     assert(pos >= 0);
30     Node* pEl = new Node;
31     pEl->value = val;
32     if (pos != 0)
33     {
34         Node* p = nodePtr(pos);
35         assert(p != nullptr);
36         pEl->next = p->next;
37         pEl->prev = p;
38         p->next = pEl;
39     }
40     else // insert at head
41     {
42         pEl->next = pHead;
43         pEl->prev = nullptr;
44         pHead = pEl;
45     }
46     if (pEl->next != nullptr) // not last element in list
47         pEl->next->prev = pEl;
48     setNumber(getNumber() + 1);
49 }
50
51 void DList::deleteAt(int pos)
52 {
53     assert(pos > 0 && pos <= getNumber());
54     Node* pDel = nodePtr(pos); // node to be deleted
55     assert(pDel != nullptr);
56     if (pos == 1) // first element
57         pHead = pHead->next;

```

```

58     else
59     {
60         pDel->prev->next = pDel->next;
61     }
62     if (pDel->next != nullptr) // not last element in list
63         pDel->next->prev = pDel->prev;
64     delete pDel;
65     setNumber(getNumber() - 1);
66 }
67
68 int DList::search(double val) const
69 {
70     Node* p = pHead; // p points to element at pos 1, if not
71     ↪ empty
72     for (int i = 1; p; ++i)
73     {
74         if (p->value == val)
75             return i;
76         p = p->next;
77     }
78     return 0; // not found
79 }
80
81 double DList::getValue(int pos) const
82 {
83     assert(pos > 0 && pos <= getNumber());
84     return nodePtr(pos)->value;
85 }
86
87 void DList::setValue(int pos, double val)
88 {
89     assert(pos > 0 && pos <= getNumber());
90     nodePtr(pos)->value = val;
91 }
92
93 void DList::print() const
94 {
95     cout << "-----" << endl;
96     cout << "Number of elements: " << getNumber() << endl;
97     cout << "Content of list:" << endl;
98     for (Node* p = pHead; p; p = p->next)
99         cout << p->value << " ";
100     cout << endl;
101 }
102
103 DList::Node* DList::nodePtr(int pos) const
104 {
105     assert(pos > 0 && pos <= getNumber());
106     Node* p = pHead; // p points to element at pos 1, if not
107     ↪ empty
108     for (int i = 1; p && i < pos; ++i)
109     {
110         p = p->next;
111     }
112     return p;
113 }

```

```

1  /*
2  * SList.h
3  *
4  * Singly-linked list for doubles
5  *
6  * Created on: 16.02.2016
7  * Author: rbondere
8  */
9
10 #ifndef SLIST_H_
11 #define SLIST_H_
12 #include "List.h"
13
14 class SList: public List
15 {
16 public:
17     SList();
18     virtual ~SList();
19
20     virtual void insertAt(int pos, double val);
21     // inserts element at (after) position pos (0: at head)
22
23     virtual void deleteAt(int pos);
24     // deletes element at position pos (>0)
25
26     virtual int search(double val) const;
27     // searches val in list and returns position of first

```

```

28     ↪ match, starting
29     // at head
30     // returns 0 if value is not found
31
32     virtual double getValue(int pos) const;
33     // returns value at position pos
34
35     virtual void setValue(int pos, double val);
36     // sets value val at position pos
37
38     virtual void print() const;
39     // prints content of list to console
40
41 private:
42     struct Node
43     {
44         double value;
45         Node* next;
46     };
47     Node* pHead; // ptr to head of list
48
49     Node* nodePtr(int pos) const;
50     // returns a pointer to the node given by position pos
51 };
52 #endif /* SLIST_H_ */

```

```

1  /*
2  * SList.cpp
3  *
4  * Created on: 17.02.2020
5  * Author: rbondere
6  */
7
8  #include <cassert>
9  #include <iostream>
10 #include "SList.h"
11 using namespace std;
12
13 SList::SList() :
14     List(), pHead(nullptr)
15 {
16 }
17
18 SList::~SList()
19 {
20     for (Node* p = pHead; p; p = pHead)
21     {
22         pHead = pHead->next;
23         delete p;
24     }
25 }
26
27 void SList::insertAt(int pos, double val)
28 {
29     assert(pos >= 0);
30     Node* pEl = new Node;
31     pEl->value = val;
32     if (pos != 0)
33     {
34         Node* p = nodePtr(pos);
35         assert(p != nullptr);
36         pEl->next = p->next;
37         p->next = pEl;
38     }
39     else // insert at head
40     {
41         pEl->next = pHead;
42         pHead = pEl;
43     }
44     setNumber(getNumber() + 1);
45 }
46
47 void SList::deleteAt(int pos)
48 {
49     assert(pos > 0 && pos <= getNumber());
50     Node* p = pHead; // cursor
51     Node* pDel = p; // node to be deleted
52     if (pos == 1) // first element
53         pHead = pHead->next;
54     else
55     {
56         for (int i = 1; i < pos - 1; ++i)

```

```

57     {
58         p = p->next;
59     }
60     pDel = p->next;
61     p->next = pDel->next;
62 }
63 delete pDel;
64 setNumber(getNumber() - 1);
65 }
66
67 int SList::search(double val) const
68 {
69     Node* p = pHead; // p points to element at pos 1, if not
70     // empty
71     for (int i = 1; p; ++i)
72     {
73         if (p->value == val)
74             return i;
75         p = p->next;
76     }
77     return 0; // not found
78 }
79
80 double SList::getValue(int pos) const
81 {
82     assert(pos > 0 && pos <= getNumber());
83     return nodePtr(pos)->value;
84 }
85
86 void SList::setValue(int pos, double val)
87 {
88     assert(pos > 0 && pos <= getNumber());
89     nodePtr(pos)->value = val;
90 }
91
92 void SList::print() const
93 {
94     cout << "-----" << endl;
95     cout << "Number of elements: " << getNumber() << endl;
96     cout << "Content of list:" << endl;
97     for (Node* p = pHead; p; p = p->next)
98         cout << p->value << " ";
99     cout << endl;
100 }
101
102 SList::Node* SList::nodePtr(int pos) const
103 {
104     assert(pos > 0 && pos <= getNumber());
105     Node* p = pHead; // p points to element at pos 1, if not
106     // empty
107     for (int i = 1; p && i < pos; ++i)
108     {
109         p = p->next;
110     }
111     return p;

```

```

1  //=====
2  // Name      : Lists.cpp
3  // Author    : Reto Bonderer
4  // Version   :
5  // Copyright : (c) HSR R. Bonderer
6  // Description : List implementations
7  //=====
8
9  #include <iostream>
10 #include "SList.h"
11 #include "DList.h"
12 using namespace std;
13
14 int main()
15 {
16     SList s;
17     cout << "Singly linked list" << endl;
18     s.print();
19
20     s.insertAt(0, 1.23);
21     s.insertAt(0, -34.6);
22     s.insertAt(0, 0.4);
23     s.print();
24
25     s.insertAt(3, 7.97);
26     s.insertAt(1, 88.9);

```

```

27     s.print();
28
29     s.deleteAt(2);
30     s.print();
31     cout << "-----" << endl;
32
33     DList d;
34     cout << "Doubly linked list" << endl;
35     d.print();
36
37     d.insertAt(0, 1.23);
38     d.insertAt(0, -34.6);
39     d.insertAt(0, 0.4);
40     d.print();
41
42     d.insertAt(3, 7.97);
43     d.insertAt(1, 88.9);
44     d.print();
45
46     d.deleteAt(5);
47     d.insertAt(4, 44.5);
48     d.print();
49
50     return 0;
51 }

```



```

1  /*
2  * List.h
3  *
4  * Created on: 15.02.2016
5  * Author: rbondere
6  */
7
8  #ifndef LIST_H_
9  #define LIST_H_
10
11 class List
12 {
13 public:
14     List() :
15         nr(0)
16     {
17     }
18
19     virtual ~List()
20     {
21     }
22
23     virtual void insertAt(int pos, double val) = 0;
24     // inserts element at (after) position pos (0: at head)
25
26     virtual void deleteAt(int pos) = 0;
27     // deletes element at position pos (>0)
28
29     virtual int search(double val) const = 0;
30     // searches val in list and returns position of first
31     // ↳ match, starting
32     // at head
33     // returns 0 if value is not found

```

```

34     int getNumber() const
35     // returns number of elements
36     {
37         return nr;
38     }
39
40     bool isEmpty() const
41     // returns true if list is empty, else false
42     {
43         return nr == 0;
44     }
45
46     virtual double getValue(int pos) const = 0;
47     // returns value at position pos
48
49     virtual void setValue(int pos, double val) = 0;
50     // sets value val at position pos
51
52     virtual void print() const = 0;
53     // prints content of list to console
54
55 protected:
56     void setNumber(int nr)
57     // sets number of elements
58     {
59         this->nr = nr;
60     }
61
62 private:
63     int nr; // number of Elements
64 };
65
66 #endif /* LIST_H_ */

```

1.5 Aufgabe 5: Gemeinsame Basisklasse mit Templates

Die komfortabelste Lösung ist offensichtlich, die beiden Klassen SList und DList als Templateklassen zu implementieren mit einer gemeinsamen Basisklasse.

1.5.1 Lösung

```

1  /*
2  * DList.h
3  * Double linked list for type T
4  *
5  * Created on: 17.02.2020
6  * Author: rbondere
7  */
8
9  #ifndef DLIST_H_
10 #define DLIST_H_
11
12 #include <cassert>
13 #include <iostream>
14
15 #include "List.h"
16
17 template<typename T>
18 class DList: public List<T>
19 {
20 public:
21     DList() :
22         List<T>(), pHead(nullptr)
23     {
24     }
25
26     virtual ~DList();
27
28     virtual void insertAt(int pos, const T& val);
29     // inserts element at (after) position pos (0: at head)
30
31     virtual void deleteAt(int pos);
32     // deletes element at position pos (>0)
33
34     virtual int search(const T& val) const;
35     // searches val in list and returns position of first
36     // ↳ match, starting
37     // at head
38     // returns 0 if value is not found
39
40     virtual const T& getValue(int pos) const;
41     // returns value at position pos

```

```

42     virtual void setValue(int pos, const T& val);
43     // sets value val at position pos
44
45     virtual void print() const;
46     // prints content of list to console
47
48 private:
49     struct Node
50     {
51         T value;
52         Node* next;
53         Node* prev;
54     };
55     Node* pHead; // ptr to head of list
56     Node* nodePtr(int pos) const;
57     // returns a pointer to the node given by position pos
58 };
59
60 template<typename T>
61 DList<T>::~DList()
62 {
63     for (Node* p = pHead; p; p = pHead)
64     {
65         pHead = pHead->next;
66         delete p;
67     }
68 }
69
70 template<typename T>
71 void DList<T>::insertAt(int pos, const T& val)
72 {
73     assert(pos >= 0);
74     Node* pEl = new Node;
75     pEl->value = val;
76     if (pos != 0)
77     {
78         Node* p = nodePtr(pos);
79         assert(p != nullptr);
80         pEl->next = p->next;
81         pEl->prev = p;
82         p->next = pEl;
83     }

```

```

84     else // insert at head
85     {
86         pEl->next = pHead;
87         pEl->prev = nullptr;
88         pHead = pEl;
89     }
90     if (pEl->next != nullptr) // not last element in list
91         pEl->next->prev = pEl;
92     List<T>::setNumber(List<T>::getNumber() + 1);
93 }
94
95 template<typename T>
96 void DList<T>::deleteAt(int pos)
97 {
98     assert(pos > 0 && pos <= List<T>::getNumber());
99     Node* pDel = nodePtr(pos); // node to be deleted
100     assert(pDel != nullptr);
101     if (pos == 1) // first element
102         pHead = pHead->next;
103     else
104     {
105         pDel->prev->next = pDel->next;
106     }
107     if (pDel->next != nullptr) // not last element in list
108         pDel->next->prev = pDel->prev;
109     delete pDel;
110     List<T>::setNumber(List<T>::getNumber() - 1);
111 }
112
113 template<typename T>
114 int DList<T>::search(const T& val) const
115 {
116     Node* p = pHead; // p points to element at pos 1, if not
117     ↪ empty
118     for (int i = 1; p; ++i)
119     {
120         if (p->value == val)
121             return i;
122         p = p->next;
123     }
124     return 0; // not found
125 }

```

```

125
126 template<typename T>
127 const T& DList<T>::getValue(int pos) const
128 {
129     assert(pos > 0 && pos <= List<T>::getNumber());
130     return nodePtr(pos)->value;
131 }
132
133 template<typename T>
134 void DList<T>::setValue(int pos, const T& val)
135 {
136     assert(pos > 0 && pos <= List<T>::getNumber());
137     nodePtr(pos)->value = val;
138 }
139
140 template<typename T>
141 void DList<T>::print() const
142 {
143     std::cout << "-----" << std::endl;
144     std::cout << "Number of elements: " << List<T>::getNumber()
145     ↪ << std::endl;
146     std::cout << "Content of list:" << std::endl;
147     for (Node* p = pHead; p; p = p->next)
148         std::cout << p->value << " ";
149     std::cout << std::endl;
150 }
151
152 template <typename T>
153 typename DList<T>::Node* DList<T>::nodePtr(int pos) const
154 {
155     assert(pos > 0 && pos <= List<T>::getNumber());
156     Node* p = pHead; // p points to element at pos 1, if not
157     ↪ empty
158     for (int i = 1; p && i < pos; ++i)
159     {
160         p = p->next;
161     }
162     return p;
163 }
164
165 #endif /* DLIST_H_ */

```

```

1  /*
2  * SList.h
3  *
4  * Singly-linked list for type T
5  *
6  * Created on: 17.02.2020
7  * Author: rbondere
8  */
9
10 #ifndef SLIST_H_
11 #define SLIST_H_
12
13 #include <iostream>
14 #include <cassert>
15
16 #include "List.h"
17
18 template<typename T>
19 class SList: public List<T>
20 {
21 public:
22     SList() :
23         List<T>(), pHead(nullptr)
24     {
25     }
26
27     virtual ~SList();
28
29     virtual void insertAt(int pos, const T& val);
30     // inserts element at (after) position pos (0: at head)
31
32     virtual void deleteAt(int pos);
33     // deletes element at position pos (>0)
34
35     virtual int search(const T& val) const;
36     // searches val in list and returns position of first
37     ↪ match, starting
38     // at head
39     // returns 0 if value is not found
40
41     virtual const T& getValue(int pos) const;
42     // returns value at position pos
43
44     virtual void setValue(int pos, const T& val);

```

```

45     // sets value val at position pos
46
47     virtual void print() const;
48     // prints content of list to console
49
50 private:
51     struct Node
52     {
53         T value;
54         Node* next;
55     };
56     Node* pHead; // ptr to head of list
57     Node* nodePtr(int pos) const;
58     // returns a pointer to the node given by position pos
59 };
60
61 template<typename T>
62 SList<T>::~SList()
63 {
64     for (Node* p = pHead; p; p = pHead)
65     {
66         pHead = pHead->next;
67         delete p;
68     }
69 }
70
71 template<typename T>
72 void SList<T>::insertAt(int pos, const T& val)
73 {
74     assert(pos >= 0);
75     Node* pEl = new Node;
76     pEl->value = val;
77     if (pos != 0)
78     {
79         Node* p = nodePtr(pos);
80         assert(p != nullptr);
81         pEl->next = p->next;
82         p->next = pEl;
83     }
84     else // insert at head
85     {
86         pEl->next = pHead;
87         pHead = pEl;
88     }
89     List<T>::setNumber(List<T>::getNumber() + 1);

```

```

90 }
91
92 template<typename T>
93 void SList<T>::deleteAt(int pos)
94 {
95     assert(pos > 0 && pos <= List<T>::getNumber());
96     Node* p = pHead; // cursor
97     Node* pDel = p; // node to be deleted
98     if (pos == 1) // first element
99         pHead = pHead->next;
100     else
101     {
102         for (int i = 1; i < pos - 1; ++i)
103         {
104             p = p->next;
105         }
106         pDel = p->next;
107         p->next = pDel->next;
108     }
109     delete pDel;
110     List<T>::setNumber(List<T>::getNumber() - 1);
111 }
112
113 template<typename T>
114 int SList<T>::search(const T& val) const
115 {
116     Node* p = pHead; // p points to element at pos 1, if not
117     ↪ empty
118     for (int i = 1; p; ++i)
119     {
120         if (p->value == val)
121             return i;
122         p = p->next;
123     }
124     return 0; // not found
125 }
126
127 template<typename T>
128 const T& SList<T>::getValue(int pos) const

```

```

128 {
129     assert(pos > 0 && pos <= List<T>::getNumber());
130     return nodePtr(pos)->value;
131 }
132
133 template<typename T>
134 void SList<T>::setValue(int pos, const T& val)
135 {
136     assert(pos > 0 && pos <= List<T>::getNumber());
137     nodePtr(pos)->value = val;
138 }
139
140 template<typename T>
141 void SList<T>::print() const
142 {
143     std::cout << "-----" << std::endl;
144     std::cout << "Number of elements: " << List<T>::getNumber()
145     ↪ << std::endl;
146     std::cout << "Content of list:" << std::endl;
147     for (Node* p = pHead; p; p = p->next)
148         std::cout << p->value << " ";
149     std::cout << std::endl;
150 }
151
152 template <typename T>
153 typename SList<T>::Node* SList<T>::nodePtr(int pos) const
154 {
155     assert(pos > 0 && pos <= List<T>::getNumber());
156     Node* p = pHead; // p points to element at pos 1, if not
157     ↪ empty
158     for (int i = 1; p && i < pos; ++i)
159     {
160         p = p->next;
161     }
162     return p;
163 }
164
165 #endif /* SLIST_H_ */

```

```

1  /*
2  * List.h
3  *
4  * Created on: 15.02.2016
5  * Author: rbondere
6  */
7
8 #ifndef LIST_H_
9 #define LIST_H_
10
11 template<typename Item>
12 class List
13 {
14 public:
15     List() :
16         nr(0)
17     {
18     }
19     virtual ~List()
20     {
21     }
22
23     virtual void insertAt(int pos, const Item& val) = 0;
24     // inserts element at (after) position pos (0: at head)
25
26     virtual void deleteAt(int pos) = 0;
27     // deletes element at position pos (>0)
28
29     virtual int search(const Item& val) const = 0;
30     // searches val in list and returns position of first
31     ↪ match, starting
32     // at head
33     // returns 0 if value is not found

```

```

34
35     bool isEmpty() const
36     // returns true if list is empty, else false
37     {
38         return nr == 0;
39     }
40
41     int getNumber() const
42     // returns number of elements
43     {
44         return nr;
45     }
46
47     virtual const Item& getValue(int pos) const = 0;
48     // returns value at position pos
49
50     virtual void setValue(int pos, const Item& val) = 0;
51     // sets value val at position pos
52
53     virtual void print() const = 0;
54     // prints content of list to console
55
56 protected:
57     void setNumber(int n)
58     // sets number of elements
59     {
60         nr = n;
61     }
62
63 private:
64     int nr; // number of Elements
65 };
66
67 #endif /* LIST_H_ */

```

```

1  //=====
2  // Name      : ListTest.cpp
3  // Author    : Reto Bonderer
4  // Version   :
5  // Copyright : (c) HSR R. Bonderer
6  // Description : List implementations
7  //=====
8
9 #include <iostream>

```

```

10 #include "SList.h"
11 #include "DList.h"
12 using namespace std;
13
14 int main()
15 {
16     SList<double> s;
17     cout << "Singly linked list" << endl;
18     s.print();

```

```

19
20 s.insertAt(0, 1.23);
21 s.insertAt(0, -34.6);
22 s.insertAt(0, 0.4);
23 s.print();
24
25 s.insertAt(3, 7.97);
26 s.insertAt(1, 88.9);
27 s.print();
28
29 s.deleteAt(2);
30 s.print();
31 cout << "-----" << endl;
32
33 DList<int> d;
34 cout << "Doubly linked list" << endl;
35 d.print();

```

```

36
37 d.insertAt(0, 123);
38 d.insertAt(0, -346);
39 d.insertAt(0, 78);
40 d.print();
41
42 d.insertAt(3, 797);
43 d.insertAt(1, 88);
44 d.print();
45
46 d.deleteAt(5);
47 d.insertAt(4, 45);
48 d.print();
49
50 return 0;
51 }

```

2 Lab 2 Listen, Algorithmen und Komplexitätstheorie

2.1 Aufgabe 1: Klasse für die Speicherung von Messwerten

Als Vorgabe erhalten Sie eine Templateklasse für eine einfach verkettete Liste. Implementieren Sie darauf basierend eine Klasse für die Verwaltung von Messreihen. Die Klasse speichert eine beliebige Anzahl von Messwerten. Einer zu implementierenden Methode können Sie einen Toleranzwert in Prozent übergeben. Die Methode entfernt dann aus der Liste alle Messwerte, die mehr als dieser Toleranzwert vom Mittelwert der Messwerte abweichen. Überlegen Sie sich als erstes, ob die Beziehung zur Listenklasse eine Vererbung oder eine Aggregation ist. Begründen Sie Ihre Wahl.

2.1.1 Lösung

Die Messwertliste ist eine Liste, deshalb ist aus objektorientierter Sicht eine Vererbungsbeziehung vorzuziehen.

Vererbung:

```
1  /*
2  * MeasureList.h
3  *
4  * Created on: 26.02.2015
5  * Author: rbondere
6  * Inheritance version
7  */
8
9  #ifndef MEASURELIST_H_
10 #define MEASURELIST_H_
11
12 #include "SList.h"
13
14 class MeasureList: public SList<double>
15 {
16 public:
17     MeasureList(double tol = 20.0);
18     void insertValue(double val);
19     void setTolerance(double tol);
20     double getMean(); // returns mean value
21     void exclude(); // excludes values beyond tolerance band
22 private:
23     double tolerance; // in percent
24 };
25
26 #endif /* MEASURELIST_H_ */
```

```
1  /*
2  * MeasureList.cpp
3  *
4  * Created on: 26.02.2015
5  * Author: rbondere
6  */
7
8  #include <cmath>
9  #include "MeasureList.h"
10
11 MeasureList::MeasureList(double tol) :
12     tolerance(tol)
13 {
14 }
15
16 void MeasureList::setTolerance(double tol)
17 {
18     tolerance = tol;
19 }
20
21 void MeasureList::exclude()
22 {
23     double mean = getMean();
24     // caution: mean may be negative
25     double lower = mean - tolerance / 100.0 * fabs(mean);
26     double higher = mean + tolerance / 100.0 * fabs(mean);
27     for (int pos = 1; pos <= getNumber(); ++pos)
28     {
29         if (getValue(pos) < lower || getValue(pos) > higher)
30         {
31             deleteAt(pos);
32             --pos; // decrease position in this case
33         }
34     }
35 }
36
37 void MeasureList::insertValue(double val)
38 {
39     insertAt(0, val); // inserts value at head
40 }
41
42 double MeasureList::getMean()
43 {
44     double sum = 0.0;
45     int nr = getNumber();
46     for (int pos = 1; pos <= nr; ++pos)
47     {
48         sum += getValue(pos);
49     }
50
51     assert(nr);
52     return sum / nr;
53 }
```

```
1  /*
2  * SList.h
3  *
4  * Singly-linked list Template
5  * The following operators must be defined for every type T
6  * == << =
7  *
8  * Created on: 20.02.2020
9  * Author: rbondere
10 */
11
12 #ifndef SLIST_H_
13 #define SLIST_H_
14 #include <iostream>
15 #include <cassert>
16 template<typename T>
17 class SList
18 {
19 public:
20     SList() :
21         pHead(nullptr), currPos(0), nr(0)
22     {
23     };
24     ~SList();
25     void insertAt(int pos,
26                 T val);
27     // inserts element at (after) position pos (0: at head)
28     void deleteAt(int pos);
29     // deletes element at position pos (>0)
30     int search(T val) const;
31     // searches val in list and returns position of first
32     // ↳ match, starting
33     // at head
34     // returns 0 if value is not found
35     bool isEmpty() const
36     // returns true if list is empty, else false
37     {
38         return nr == 0;
39     }
40 }
```

```

38     }
39     int getNumber() const
40     // returns number of elements
41     {
42         return nr;
43     }
44     int getNext() const;
45     // returns position of next element (0: no next, i.e. at
46     // ↪ end)
47     int getPosition() const
48     // returns current position (0: list is empty)
49     {
50         return currPos;
51     }
52     void setPosition(int pos);
53     // sets current position to pos
54     T getValue(int pos) const;
55     // returns value at position pos
56     void setValue(int pos,
57                   T val);
58     // sets value val at position pos
59     void print() const;
60     // prints content of list to console
61 private:
62     struct Node
63     {
64         T value;
65         Node* next;
66     };
67     Node* pHead; // ptr to head of list
68     int currPos; // current position
69     int nr; // number of Elements
70
71     Node* nodePtr(int pos) const;
72     // returns a pointer to the node given by position pos
73 };
74 // template implementation
75
76 template<typename T>
77 SList<T>::~SList()
78 {
79     for (Node* p = pHead; p; p = pHead)
80     {
81         pHead = pHead->next;
82         delete p;
83     }
84 }
85
86 template<typename T>
87 void SList<T>::insertAt(int pos,
88                        T val)
89 {
90     assert(pos >= 0);
91     Node* pEl = new Node;
92     pEl->value = val;
93     if (pos != 0)
94     {
95         Node* p = nodePtr(pos);
96         assert(p != nullptr);
97         pEl->next = p->next;
98         p->next = pEl;
99     }
100     else // insert at head
101     {
102         pEl->next = pHead;
103         pHead = pEl;
104     }
105     ++nr;
106 }
107
108 template<typename T>
109 void SList<T>::deleteAt(int pos)
110 {
111     assert(pos > 0 && pos <= nr);
112     Node* p = pHead; // cursor
113     Node* pDel = p; // node to be deleted
114     if (pos == 1) // first element
115         pHead = pHead->next;
116     else
117     {

```

```

118         for (int i = 1; i < pos - 1; ++i)
119         {
120             p = p->next;
121         }
122         pDel = p->next;
123         p->next = pDel->next;
124     }
125     delete pDel;
126     --nr;
127 }
128
129 template<typename T>
130 int SList<T>::search(T val) const
131 {
132     Node* p = pHead; // p points to element at pos 1, if not
133     // ↪ empty
134     for (int i = 1; p; ++i)
135     {
136         if (p->value == val)
137             return i;
138         p = p->next;
139     }
140     return 0; // not found
141 }
142
143 template<typename T>
144 int SList<T>::getNext() const
145 {
146     if (currPos >= nr)
147         return 0;
148     else
149         return currPos + 1;
150 }
151
152 template<typename T>
153 void SList<T>::setPosition(int pos)
154 {
155     assert(pos >= 0 && pos <= nr);
156     currPos = pos;
157 }
158
159 template<typename T>
160 T SList<T>::getValue(int pos) const
161 {
162     assert(pos > 0 && pos <= nr);
163     return nodePtr(pos)->value;
164 }
165
166 template<typename T>
167 void SList<T>::setValue(int pos,
168                        T val)
169 {
170     assert(pos > 0 && pos <= nr);
171     nodePtr(pos)->value = val;
172 }
173
174 template<typename T>
175 void SList<T>::print() const
176 {
177     std::cout << "-----" << std::endl;
178     std::cout << "Number of elements: " << nr << std::endl;
179     std::cout << "Content of list:" << std::endl;
180     for (Node* p = pHead; p; p = p->next)
181         std::cout << p->value << " ";
182     std::cout << std::endl;
183 }
184
185 template<typename T>
186 typename SList<T>::Node* SList<T>::nodePtr(int pos) const
187 // returns a pointer to the node given by position pos
188 {
189     Node* p = pHead; // p points to element at pos 1, if not
190     // ↪ empty
191     for (int i = 1; p && i < pos; ++i)
192     {
193         p = p->next;
194     }
195     return p;
196 }
197 #endif /* SLIST_H_ */

```

```

1 //=====
2 // Name      : MListMain.cpp
3 // Author     : Reto Bonderer
4 // Version    :
5 // Copyright  : (c) HSR R. Bonderer

```

```

6 // Description : Measurement list main program
7 //=====
8
9 #include <iostream>
10 #include "MeasureList.h"

```

```

11 using namespace std;
12
13 int main()
14 {
15     MeasureList m;
16     m.setTolerance(5);
17     m.insertValue(9);
18     m.insertValue(10);
19     m.insertValue(11);
20     m.print();
21     cout << "Mean = " << m.getMean() << endl;
22
23     m.exclude();
24     m.print();
25     m.exclude();
26     m.print();
27     m.insertValue(10.44);
28     m.print();
29     cout << "Mean = " << m.getMean() << endl;
30     m.exclude();
31     m.print();
32     return 0;

```

Aggregation

```

1  /*
2  * MeasureList.h
3  *
4  * Created on: 24.02.2013
5  * Author: rbondere
6  * Aggregation version (no inheritance)
7  */
8
9 #ifndef MEASURELIST_H_
10 #define MEASURELIST_H_
11
12 #include "SList.h"
13
14 class MeasureList
15 {
16 public:
17     MeasureList(double tol = 20.0);
18     void insertValue(double val);
19     void setTolerance(double tol);
20     double getMean(); // returns mean value
21     void exclude(); // excludes values beyond tolerance band
22     void print() const; // prints content of list
23 private:
24     SList<double> list;
25     double tolerance; // in percent
26 };
27
28 #endif /* MEASURELIST_H_ */

```

```

1  /*
2  * MeasureList.cpp
3  *
4  * Created on: 24.02.2013
5  * Author: rbondere
6  */
7
8 #include <cmath>
9 #include "MeasureList.h"
10
11 MeasureList::MeasureList(double tol) :
12     tolerance(tol)
13 {
14 }
15
16 void MeasureList::setTolerance(double tol)
17 {
18     tolerance = tol;
19 }
20
21 void MeasureList::exclude()
22 {
23     double mean = getMean();
24     // caution: mean may be negative
25     double lower = mean - tolerance / 100.0 * fabs(mean);
26     double higher = mean + tolerance / 100.0 * fabs(mean);
27     for (int pos = 1; pos <= list.getNumber(); ++pos)
28     {
29         if (list.getValue(pos) < lower || list.getValue(pos) >
30             ↩ higher)
31         {
32             list.deleteAt(pos);
33             --pos;
34         }
35     }
36
37 void MeasureList::insertValue(double val)
38 {
39     list.insertAt(0, val); // inserts value at head
40 }
41
42 void MeasureList::print() const
43 {
44     list.print();
45 }
46
47 double MeasureList::getMean()
48 {
49     double sum = 0;
50     int nr = list.getNumber();
51     for (int pos = 1; pos <= nr; ++pos)
52     {
53         sum += list.getValue(pos);
54     }
55     assert(nr);
56     return sum / nr;
57 }

```

```

1  /*
2  * SList.h
3  *
4  * Singly-linked list Template
5  * The following operators must be defined for every type T
6  * == << =
7  *
8  * Created on: 24.02.2020
9  * Author: rbondere
10 */
11
12 #ifndef SLIST_H_
13 #define SLIST_H_
14 #include <iostream>
15 #include <cassert>
16 template<typename T>
17 class SList
18 {
19 public:
20     SList() :
21         pHead(nullptr), currPos(0), nr(0)
22
23 };
24 ~SList();
25 void insertAt(int pos,
26             T val);
27 // inserts element at (after) position pos (0: at head)
28 void deleteAt(int pos);
29 // deletes element at position pos (>0)
30 int search(T val) const;
31 // searches val in list and returns position of first
32     ↩ match, starting
33 // at head
34 // returns 0 if value is not found
35 bool isEmpty() const
36 // returns true if list is empty, else false
37 {
38     return nr == 0;
39 }
40 int getNumber() const
41 // returns number of elements
42 {

```

```

42     return nr;
43 }
44 int getNext() const;
45 // returns position of next element (0: no next, i.e. at
46 // ↪ end)
47 int getPosition() const
48 // returns current position (0: list is empty)
49 {
50     return currPos;
51 }
52 void setPosition(int pos);
53 // sets current position to pos
54 T getValue(int pos) const;
55 // returns value at position pos
56 void setValue(int pos,
57               T val);
58 // sets value val at position pos
59 void print() const;
60 // prints content of list to console
61 private:
62     struct Node
63     {
64         T value;
65         Node* next;
66     };
67     Node* pHead; // ptr to head of list
68     int currPos; // current position
69     int nr; // number of Elements
70
71     Node* nodePtr(int pos) const;
72     // returns a pointer to the node given by position pos
73 };
74
75 // template implementation
76
77 template<typename T>
78 SList<T>::~SList()
79 {
80     for (Node* p = pHead; p; p = pHead)
81     {
82         pHead = pHead->next;
83         delete p;
84     }
85 }
86
87 template<typename T>
88 void SList<T>::insertAt(int pos,
89                        T val)
90 {
91     assert(pos >= 0);
92     Node* pEl = new Node;
93     pEl->value = val;
94     if (pos != 0)
95     {
96         Node* p = nodePtr(pos);
97         assert(p != nullptr);
98         pEl->next = p->next;
99         p->next = pEl;
100     }
101     else // insert at head
102     {
103         pEl->next = pHead;
104         pHead = pEl;
105     }
106     ++nr;
107 }
108
109 template<typename T>
110 void SList<T>::deleteAt(int pos)
111 {
112     assert(pos > 0 && pos <= nr);
113     Node* p = pHead; // cursor
114     Node* pDel = p; // node to be deleted
115     if (pos == 1) // first element
116         pHead = pHead->next;
117     else
118     {
119         for (int i = 1; i < pos - 1; ++i)

```

```

120         p = p->next;
121     }
122     pDel = p->next;
123     p->next = pDel->next;
124 }
125 delete pDel;
126 --nr;
127 }
128
129 template<typename T>
130 int SList<T>::search(T val) const
131 {
132     Node* p = pHead; // p points to element at pos 1, if not
133     // ↪ empty
134     for (int i = 1; p; ++i)
135     {
136         if (p->value == val)
137             return i;
138         p = p->next;
139     }
140     return 0; // not found
141 }
142
143 template<typename T>
144 int SList<T>::getNext() const
145 {
146     if (currPos >= nr)
147         return 0;
148     else
149         return currPos + 1;
150 }
151
152 template<typename T>
153 void SList<T>::setPosition(int pos)
154 {
155     assert(pos >= 0 && pos <= nr);
156     currPos = pos;
157 }
158
159 template<typename T>
160 T SList<T>::getValue(int pos) const
161 {
162     assert(pos > 0 && pos <= nr);
163     return nodePtr(pos)->value;
164 }
165
166 template<typename T>
167 void SList<T>::setValue(int pos,
168                        T val)
169 {
170     assert(pos > 0 && pos <= nr);
171     nodePtr(pos)->value = val;
172 }
173
174 template<typename T>
175 void SList<T>::print() const
176 {
177     std::cout << "-----" << std::endl;
178     std::cout << "Number of elements: " << nr << std::endl;
179     std::cout << "Content of list:" << std::endl;
180     for (Node* p = pHead; p; p = p->next)
181         std::cout << p->value << " ";
182     std::cout << std::endl;
183 }
184
185 template<typename T>
186 typename SList<T>::Node* SList<T>::nodePtr(int pos) const
187 // returns a pointer to the node given by position pos
188 {
189     Node* p = pHead; // p points to element at pos 1, if not
190     // ↪ empty
191     for (int i = 1; p && i < pos; ++i)
192     {
193         p = p->next;
194     }
195     return p;
196 }
197 #endif /* SLIST_H_ */

```

```

1 //=====
2 // Name      : MListMain.cpp
3 // Author    : Reto Bonderer
4 // Version   :
5 // Copyright : (c) HSR R. Bonderer
6 // Description : Measurement list main program
7 //=====

```

```

8
9 #include <iostream>
10 #include "MeasureList.h"
11 using namespace std;
12
13 int main()
14 {

```



```

15 MeasureList m;
16 m.setTolerance(5);
17 m.insertValue(9);
18 m.insertValue(10);
19 m.insertValue(11);
20 m.print();
21 cout << "Mean = " << m.getMean() << endl;
22 m.exclude();

23 m.print();
24 m.exclude();
25 m.print();
26 m.insertValue(10.44);
27 m.exclude();
28 m.print();
29 return 0;
30 }

```

2.2 Aufgabe 2: Komplexitätsbetrachtungen

Bestimmen Sie den benötigten Aufwand der folgenden Algorithmen in O -Notation. Alle Algorithmen berechnen die Potenz $c = a^b$.

```

1 // berechnet 'a^^b', Voraussetzung: b>=0
2 double potenz1(double a, int b)
3 {
4     double c = 1;
5     while (b > 0)
6     {
7         c = c * a;
8         b = b - 1;
9     }
10    return c;
11 }
12 // berechnet 'a^^b', Voraussetzung: b>=0
13 double potenz2(double a, int b)
14 {
15     if (b == 0)
16         return 1;
17
18     else
19         return potenz2(a, b-1) * a;
20 }
21 // berechnet 'a^^b', Voraussetzung: b>=0
22 double potenz3(double a, int b)
23 {
24     double c = 1;
25     while (b > 0)
26     {
27         if (b % 2 == 1)
28             c = c * a;
29         a = a * a;
30         b = b / 2;
31     }
32    reeturn c;
33 }

```

2.2.1 Lösung

In der Funktion `potenz1()` wird die `while`-Schleife b mal durchlaufen, d.h. dieser Algorithmus ist $O(b)$. Die Funktion `potenz2()` wird b mal rekursiv aufgerufen, weitere Schleifen sind nicht vorhanden, d.h. dieser Algorithmus ist $O(b)$. In der Funktion `potenz3()` wird b bei jedem Schleifendurchlauf halbiert. Die Schleife wird ungefähr $\lg(b)$ mal durchlaufen (\lg = logarithmus dualis, Zweierlogarithmus). `potenz3()` ist demnach $O(\lg b)$.

2.3 Aufgabe 3: Implementation eines dynamischen Stacks mit Hilfe einer verketteten Liste

Ein Stack (Stapel, LIFO, Last-In-First-Out) ist eine Datenstruktur, bei der das Element, das zuerst auf den Stack gespeichert wird, als letztes wieder ausgelesen wird.

Implementieren Sie eine Klasse `Stack`, die nicht auf Arrays, sondern mit der vorgegebenen einfach verketteten Liste arbeitet. Die `Stack`-Klasse soll als Template implementiert sein. Überlegen Sie sich auch hier als erstes, ob die Beziehung zur Listeklasse eine Vererbung oder eine Aggregation ist. Begründen Sie Ihre Wahl.

Als Methoden müssen Sie die bekannten Stackoperationen `push()`, `pop()`, `isEmpty()` und `peek()` realisieren. Reservieren Sie im Konstruktor die als Parameter übergebene Anzahl Listenelemente. Falls ein `push()` bei einem vollen Stack probiert wird, soll jetzt aber nicht ein Fehler ausgegeben werden, stattdessen sollen Sie den Stack dynamisch um zusätzliche Elemente vergrößern.

2.3.1 Lösung

Der Stack ist nicht eine Liste, er benutzt nur eine für die Speicherung der Daten. Eine Vererbung kommt deshalb nicht in Frage, Aggregation ist die richtige Variante. Da der Stack beliebige Daten speichern können soll, muss er unbedingt als Templateklasse implementiert werden.

```

1 /*
2  * SList.h
3  *
4  * Singly-linked list Template
5  * The following operators must be defined for every type T
6  * == << =
7  *
8  * Created on: 20.02.20
9  * Author: rbondere
10 */
11
12 #ifndef SLIST_H_
13 #define SLIST_H_
14
15 #include <cassert>
16 template<typename T>
17 class SList
18 {
19 public:
20     SList() :
21         pHead(nullptr), currPos(0), nr(0)
22     {
23     };
24     ~SList();
25     void insertAt(int pos,
26                 T val);
27     // inserts element at (after) position pos (0: at head)
28 };
29

```

```

27 void deleteAt(int pos);
28 // deletes element at position pos (>0)
29 int search(T val) const;
30 // searches val in list and returns position of first
    ↳ match, starting
31 // at head
32 // returns 0 if value is not found
33 bool isEmpty() const
34 // returns true if list is empty, else false
35 {
36     return nr == 0;
37 }
38 int getNumber() const
39 // returns number of elements
40 {
41     return nr;
42 }
43 int getNext() const;
44 // returns position of next element (0: no next, i.e. at
    ↳ end)
45 int getPosition() const
46 // returns current position (0: list is empty)
47 {
48     return currPos;
49 }
50 void setPosition(int pos);
51 // sets current position to pos
52 T getValue(int pos) const;
53 // returns value at position pos
54 void setValue(int pos,
55               T val);
56 // sets value val at position pos
57 void print() const;
58 // prints content of list to console
59 private:
60 struct Node
61 {
62     T value;
63     Node* next;
64 };
65 Node* pHead; // ptr to head of list
66 int currPos; // current position
67 int nr; // number of Elements
68
69 Node* nodePtr(int pos) const;
70 // returns a pointer to the node given by position pos
71 };
72
73 // template implementation
74
75 template<typename T>
76 SList<T>::~SList()
77 {
78     for (Node* p = pHead; p; p = pHead)
79     {
80         pHead = pHead->next;
81         delete p;
82     }
83 }
84
85 template<typename T>
86 void SList<T>::insertAt(int pos,
87                        T val)
88 {
89     assert(pos >= 0);
90     Node* pEl = new Node;
91     pEl->value = val;
92     if (pos != 0)
93     {
94         Node* p = nodePtr(pos);
95         assert(p != nullptr);
96         pEl->next = p->next;
97         p->next = pEl;
98     }
99     else // insert at head
100     {
101         pEl->next = pHead;
102         pHead = pEl;
103     }
104     ++nr;
105 }
106
107 template<typename T>
108 void SList<T>::deleteAt(int pos)
109 {
110     assert(pos > 0 && pos <= nr);
111
112     Node* p = pHead; // cursor
113     Node* pDel = p; // node to be deleted
114     if (pos == 1) // first element
115         pHead = pHead->next;
116     else
117     {
118         for (int i = 1; i < pos - 1; ++i)
119         {
120             p = p->next;
121         }
122         pDel = p->next;
123         p->next = pDel->next;
124     }
125     delete pDel;
126     --nr;
127 }
128
129 template<typename T>
130 int SList<T>::search(T val) const
131 {
132     Node* p = pHead; // p points to element at pos 1, if not
    ↳ empty
133     for (int i = 1; p; ++i)
134     {
135         if (p->value == val)
136             return i;
137         p = p->next;
138     }
139     return 0; // not found
140 }
141
142 template<typename T>
143 int SList<T>::getNext() const
144 {
145     if (currPos >= nr)
146         return 0;
147     else
148         return currPos + 1;
149 }
150
151 template<typename T>
152 void SList<T>::setPosition(int pos)
153 {
154     assert(pos >= 0 && pos <= nr);
155     currPos = pos;
156 }
157
158 template<typename T>
159 T SList<T>::getValue(int pos) const
160 {
161     assert(pos > 0 && pos <= nr);
162     return nodePtr(pos)->value;
163 }
164
165 template<typename T>
166 void SList<T>::setValue(int pos,
167                        T val)
168 {
169     assert(pos > 0 && pos <= nr);
170     nodePtr(pos)->value = val;
171 }
172
173 template<typename T>
174 void SList<T>::print() const
175 {
176     std::cout << "-----" << std::endl;
177     std::cout << "Number of elements: " << nr << std::endl;
178     std::cout << "Content of list:" << std::endl;
179     for (Node* p = pHead; p; p = p->next)
180         std::cout << p->value << " ";
181     std::cout << std::endl;
182 }
183
184 template<typename T>
185 typename SList<T>::Node* SList<T>::nodePtr(int pos) const
186 // returns a pointer to the node given by position pos
187 {
188     Node* p = pHead; // p points to element at pos 1, if not
    ↳ empty
189     for (int i = 1; p && i < pos; ++i)
190     {
191         p = p->next;
192     }
193     return p;
194 }
195 #endif /* SLIST_H_ */

```

```

1  /*
2  * stack.h
3  *
4  * Created on: 24.02.2017
5  * Author: rbondere
6  * Stack, der dynamisch erweitert wird
7  */
8
9  #ifndef STACK_H_
10 #define STACK_H_
11
12 #include "SList.h"
13
14 template<typename ElemType>
15 class Stack
16 {
17 public:
18     Stack() : error(false)
19         // Default-Konstruktor
20     {
21     }
22     void push(const ElemType& e);
23     // legt ein Element auf den Stack. Falls der Stack voll
24     // ist, wird er
25     // dynamisch erweitert
26     // wasError() gibt Auskunft, ob push() erfolgreich war
27
28     ElemType pop();
29     // nimmt ein Element vom Stack, falls der Stack nicht leer
30     // ist
31     // wasError() gibt Auskunft, ob pop() erfolgreich war
32
33     ElemType peek() const;
34     // liest das oberste Element vom Stack, falls der Stack
35     // nicht leer ist
36     // wasError() gibt Auskunft, ob peek() erfolgreich war
37
38     bool isEmpty() const;
39     // return: true: Stack ist leer
40     //         false: sonst
41
42     int getDepth() const;
43     // return: Stacktiefe: Anzahl gueltige Elemente im Stack
44
45     bool wasError() const
46     // return: true: Operation war fehlerhaft
47     //         false: sonst
48     {
49         return error;
50     }
51 private:
52     SList<ElemType> list; // Dynamischer Speicher fuer Stack
53     mutable bool error;  // true: Fehler passiert; false:
54     //                     sonst

```

```

51     // mutable: auch const-Methoden koennen dieses Attribut
52     //         ↪ setzen
53 };
54 // template implementation
55
56 template<typename ElemType>
57 void Stack<ElemType>::push(const ElemType& e)
58 {
59     error = false;
60     list.insertAt(0, e); // insert at head
61 }
62
63 template<typename ElemType>
64 ElemType Stack<ElemType>::pop()
65 {
66     error = list.isEmpty();
67     if (!error)
68     {
69         ElemType e = list.getValue(1); // pos == 1: element at
70         //         ↪ head
71         list.deleteAt(1);
72         return e;
73     }
74     else
75         return 0;
76 }
77
78 template<typename ElemType>
79 ElemType Stack<ElemType>::peek() const
80 {
81     error = list.isEmpty();
82     if (!error)
83     {
84         return list.getValue(1); // pos == 1: element at head
85     }
86     else
87         return 0;
88 }
89
90 template<typename ElemType>
91 bool Stack<ElemType>::isEmpty() const
92 {
93     return list.isEmpty();
94 }
95
96 template<typename ElemType>
97 int Stack<ElemType>::getDepth() const
98 {
99     return list.getNumber();
100 }
101 #endif // STACK_H_

```

```

1  /*
2  * StackTest.cpp
3  *
4  * Created on: 05.03.2020
5  * Author: rbondere
6  */
7
8  #include <iostream>
9  #include <string>
10 #include "StackUI.h"
11 using namespace std;
12
13 int main()
14 {
15     string theType = "";
16
17     cout << "\n\nElementType (int, double, char): ";
18     cin >> theType;
19     if (theType == "int")
20     {

```

```

21         StackUI<int> sUI;
22         sUI.dialog();
23     }
24     else if (theType == "double")
25     {
26         StackUI<double> sUI;
27         sUI.dialog();
28     }
29     else if (theType == "char")
30     {
31         StackUI<char> sUI;
32         sUI.dialog();
33     }
34     else
35         cout << "Illegal input" << endl;
36
37     cout << "Bye" << endl;
38
39     return 0;
40 }

```

```

1  /*
2  * StackUI.h
3  *
4  * User Interface for Stack application
5  * Created on: 24.02.2017
6  * Author: rbondere
7  */

```

```

8
9  #ifndef STACKUI_H_
10 #define STACKUI_H_
11 #include "Stack.h"
12
13 template<typename ElemType>
14 class StackUI

```

```

15 {
16     public:
17         void dialog();
18         // starts the user dialog
19     private:
20         Stack<ElemType> s;
21 };
22
23 // template implementation
24
25 // default Ctor of Stack is automatically called
26 #include <iostream>
27 using namespace std;
28
29 template<typename ElemType>
30 void StackUI<ElemType>::dialog()
31 {
32     char ch = 0;
33     ElemType e;
34     do
35     {
36         cout << "\n\nOperation (Quit, pUsh, pOp, peeK, isEmpty) ";
37         cin >> ch;
38         switch (ch)
39         {
40             case 'q':
41             case 'Q': // quit
42                 break;
43             case 'u':
44             case 'U': // push
45                 cout << "\nElement to push: ";
46                 cin >> e;
47                 s.push(e);
48                 if (s.wasError())
49                     cout << "\nError: Stack full.";
50
51                 break;
52             case 'o':
53             case 'O': // pop
54                 e = s.pop();
55                 if (s.wasError())
56                     cout << "\nError: Stack is empty (nothing to pop).";
57                 else
58                     cout << "\nPopped element " << e;
59                 break;
60             case 'k':
61             case 'K': // peek
62                 e = s.peek();
63                 if (s.wasError())
64                     cout << "\nError: Stack is empty (nothing to peek).";
65                 ↪ ;
66             else
67                 cout << "\nPeeked element " << e;
68                 break;
69             case 'e':
70             case 'E': // isEmpty
71                 if (s.isEmpty())
72                     cout << "\nStack is empty.";
73                 else
74                     cout << "\nStack contains " << s.getDepth() << "
75                     ↪ elements.";
76                 break;
77             default:
78                 cout << "\nInvalid operation.";
79                 break;
80         }
81     } while (ch != 'Q' && ch != 'q');
82 }
83 #endif /* STACKUI_H_ */

```

3 Lab 3 Sortieralgorithmen und Komplexitätstheorie

3.1 Aufgabe 1: Zeitmessung bei Fibonacci-Implementationen

Sie erhalten als Vorgabe die beiden Dateien `fiboit.c` und `fiborek.c`. Sie beinhalten eine iterative Implementation der Fibonacci-Zahlen, respektive eine rekursive Implementation, jeweils in der Programmiersprache C.

1. Implementieren Sie die beiden Programme je in C++ (sie müssen nicht objektorientiert sein).
2. Sie haben gesehen, dass die rekursive Implementation mit der Fakultät wächst. Verifizieren Sie die Theorie, indem Sie Zeitmessungen für die Berechnung vornehmen.

Hinweis zu den Zeitmessungen:

Mit einfachen Mitteln ist eine genaue absolute Zeitmessung nicht durchführbar. Für unsere Zwecke genügt hier eine relative Zeitmessung, die genügend genau ist. Wählen Sie eine der folgenden Varianten:

Variante 1: Messung der Standardzeit

Die Funktion `clock()` aus `<ctime>` liefert die abgelaufene CPU-Zeit in Clockticks seit Programmstart. Wenn diese Grösse durch `CLOCKS_PER_SEC` geteilt wird, erhält man eine Zeit in Sekunden.

```
1 #include <ctime>
2 clock_t start = clock();
3 // do something
4 clock_t end = clock();
5 cout << "Ticks: " << end-start << endl;
6 cout << "Time: " << static_cast<double>(end-start) / CLOCKS_PER_SEC << " sec" << endl;
```

Variante 2: Native Zeitmessung von Linux

```
1 #include <sys/resource.h>
2 #include <sys/types.h>
3 rusage tp;
4 double start; // Startzeit in Millisekunden
5 double end; // Endzeit
6 getrusage(RUSAGE_SELF, &tp);
7 start = static_cast<double>(tp.ru_utime.tv_sec) +
8         static_cast<double>(tp.ru_utime.tv_usec)/1E6;
9 // do something
10 getrusage(RUSAGE_SELF, &tp);
11 end = static_cast<double>(tp.ru_utime.tv_sec) +
12        static_cast<double>(tp.ru_utime.tv_usec)/1E6;
13 cout << "Dauer: " << end-start << " sec" << endl;
```

3.1.1 Lösung

Iterativ + Linux Zeitmessung

```
1 // Datei: fiboit.cpp
2 // berechnet die Fibonacci-Zahl iterativ
3 // R. Bonderer, 04.03.2013
4
5 #include <iostream>
6 #include <iomanip>
7 #include <sys/resource.h>
8 #include <sys/types.h>
9 using namespace std;
10
11 unsigned long fibonacci(unsigned int n);
12
13
14 int main(void)
15 {
16     unsigned int zahl;
17     unsigned long fibo;
18     rusage tp;
19     double start; // Startzeit in Sekunden, Auflösung in
20                   // ↳ Mikrosekunden
21     double end; // Endzeit
22
23     cout << "n eingeben: ";
24     cin >> zahl;
25
26     getrusage(RUSAGE_SELF, &tp);
27     start = static_cast<double>(tp.ru_utime.tv_sec) +
28             static_cast<double>(tp.ru_utime.tv_usec)/1E6;
```

```
29     fibo = fibonacci(zahl);
30
31     getrusage(RUSAGE_SELF, &tp);
32     end = static_cast<double>(tp.ru_utime.tv_sec) +
33           static_cast<double>(tp.ru_utime.tv_usec)/1E6;
34
35     cout << "Fibonacci(" << zahl << ")= " << fibo << endl;
36     cout << fixed << setprecision(3) << "Dauer: " << end-start
37           << " sec" << endl;
38     return 0;
39 }
40
41 unsigned long fibonacci(unsigned int n)
42 {
43     unsigned long fibMin1 = 1;
44     unsigned long fibMin2 = 1;
45     unsigned long fib = 1;
46
47     if (n == 1 || n == 2)
48         return fib;
49
50     for (unsigned int i = 3; i <= n; ++i)
51     {
52         fib = fibMin2 + fibMin1;
53         fibMin2 = fibMin1;
54         fibMin1 = fib;
55     }
56     return fib;
57 }
```

Rekursiv + clock() Zeitmessung

```

1 // Datei: fiborek.cpp
2 // berechnet die Fibonacci-Zahl rekursiv
3 // R. Bonderer, 03.03.2020
4
5 #include <iostream>
6 #include <ctime>
7
8 using namespace std;
9
10 // Funktionsprototypen
11 unsigned long fibonacci(unsigned int n);
12 // berechnet die Fibonacci-Zahl von n rekursiv
13 // return: Fibonacci-Zahl
14
15 int main()
16 {
17     unsigned int zahl;
18     unsigned long fibo;
19
20     cout << "n eingeben: ";
21     cin >> zahl;
22

```

```

23     clock_t start = clock();
24     fibo = fibonacci(zahl);
25     clock_t end = clock();
26
27     cout << "Fibonacci(" << zahl << ") = " << fibo << endl;
28     cout << "Start: " << start << endl;
29     cout << "End : " << end << endl;
30     cout << "Dauer: " << end-start << " Ticks" << endl;
31     cout << "Dauer: " << static_cast<double>(end-start) /
        ↪ CLOCKS_PER_SEC
        << " sec" << endl;
32     return 0;
33 }
34
35 unsigned long fibonacci(unsigned int n)
36 {
37     if (n == 1 || n == 2)
38         return 1;
39     else
40         return fibonacci(n - 1) + fibonacci(n - 2);
41 }
42

```

Rekursiv + Linux Zeitmessung

```

1 // Datei: fiborek.cpp
2 // berechnet die Fibonacci-Zahl rekursiv
3 // misst Zeit mit Linux CPU Zeitfunktionen
4 // R. Bonderer, 03.03.2020
5
6 #include <iostream>
7 #include <iomanip>
8 #include <sys/resource.h>
9 #include <sys/types.h>
10
11 using namespace std;
12
13 // Funktionsprototypen
14 unsigned long fibonacci(unsigned int n);
15 // berechnet die Fibonacci-Zahl von n rekursiv
16 // return: Fibonacci-Zahl
17
18 int main()
19 {
20     unsigned int zahl;
21     unsigned long fibo;
22     rusage tp;
23     double start; // Startzeit in Millisekunden
24     double end; // Endzeit
25
26     cout << "n eingeben: ";

```

```

27     cin >> zahl;
28
29     getrusage(RUSAGE_SELF, &tp);
30     start = static_cast<double>(tp.ru_utime.tv_sec) +
        static_cast<double>(tp.ru_utime.tv_usec)/1E6;
31
32     fibo = fibonacci(zahl);
33
34     getrusage(RUSAGE_SELF, &tp);
35     end = static_cast<double>(tp.ru_utime.tv_sec) +
        static_cast<double>(tp.ru_utime.tv_usec)/1E6;
36
37     cout << "Fibonacci(" << zahl << ") = " << fibo << endl;
38     cout << fixed << setprecision(3) << "Dauer: " << end-start
        ↪ << " sec" << endl;
39     return 0;
40 }
41
42 unsigned long fibonacci(unsigned int n)
43 {
44     if (n == 1 || n == 2)
45         return 1;
46     else
47         return fibonacci(n - 1) + fibonacci(n - 2);
48 }
49
50

```

3.2 Aufgabe 2: Klasse für Stoppuhr

Implementieren Sie eine Klasse `StopWatch`, die sich bei der Gründung eines Objekts die Zeit merkt. Bei jedem Aufruf der Methode `elapsed()` wird die bis dahin abgelaufene Zeit in Sekunden zurückgegeben. Beispiel:

```

1 StopWatch t; // starts stopwatch
2 double d = t.elapsed();

```

3.2.1 Lösung

```

1 /*
2  * Stopwatch.h
3  *
4  * Created on: 04.03.2013
5  * Author: rbondere
6  */
7
8 #ifndef STOPWATCH_H_
9 #define STOPWATCH_H_
10
11 class Stopwatch
12 {
13 public:
14     Stopwatch();
15     double elapsed() const; // returns elapsed time since
        ↪ start in seconds
16 private:
17     double startTime; // unit: seconds
18 };
19
20 #endif /* STOPWATCH_H_ */

```

```

1 /*
2  * Stopwatch.cpp
3  *
4  * Created on: 04.03.2013
5
6  * Author: rbondere
7  */
8 #include "StopWatch.h"

```

```

9  #include <sys/resource.h>
10 #include <sys/types.h>
11
12 Stopwatch::StopWatch()
13 {
14     rusage tp;
15     getrusage(RUSAGE_SELF, &tp);
16     startTime = static_cast<double>(tp.ru_utime.tv_sec) +
17                 static_cast<double>(tp.ru_utime.tv_usec)/1E6;
18 }

```

```

19
20 double Stopwatch::elapsed() const
21 {
22     rusage tp;
23     getrusage(RUSAGE_SELF, &tp);
24     return static_cast<double>(tp.ru_utime.tv_sec) +
25            static_cast<double>(tp.ru_utime.tv_usec)/1E6 -
26            ↪ startTime;
27 }

```

```

1  //=====
2  // Name      : StopwatchTest.cpp
3  // Author     : Reto Bonderer
4  // Version    :
5  // Copyright  : (c) HSR R. Bonderer
6  // Description : Test program for Stopwatch (Linux style)
7  //=====
8
9  #include <iostream>
10 #include "StopWatch.h"
11 using namespace std;
12

```

```

13 int main()
14 {
15     Stopwatch w;
16     for (int j=0; j<10; ++j)
17     {
18         for(long i=0; i<100000000; ++i)
19         {}
20         cout << w.elapsed() << " sec." << endl;
21     }
22
23     return 0;
24 }

```

3.3 Aufgabe 3: Komplexität

Es gilt die Annahme, dass ein gewisses Programm jeden Abend exakt eine Stunde Rechenzeit bekommt. Sie haben herausgefunden, dass das Programm $n = 1'000'000$ Datensätze verarbeiten kann. Nun wird ein neuer Rechner angeschafft, der 100 Mal schneller als der alte ist. Wie viele Datensätze kann Ihr Programm nun in einer Stunde verarbeiten, wenn wir die folgende Zeitkomplexität mit den Konstanten k_i annehmen?

1. $k_1 \cdot n$
2. $k_2 \cdot n \cdot \log_{10} n$
3. $k_3 \cdot n^2$
4. $k_4 \cdot n^3$
5. $k_5 \cdot 10^n$

Hinweis: Verwenden Sie den Ansatz, dass der schnelle Rechner in einer Stunde gleich viel leistet, wie der langsame in 100 Stunden.

3.3.1 Lösung

Ansatz:

- Langsamer Rechner: $T(n) = 1\text{h}$
- Schnellerer Rechner : 100 Mal Schneller
 - Der schnellere Rechner leistet in 1h gleich viel wie der langsamere in 100 h.
 - 1. $T(n) = 1$ für $n = 10^6$ einsetzen \Rightarrow ergibt k ;
 - 2. $T(n) \stackrel{!}{=} 100$ setzen mit unter (1.) ermitteltem k ;
 - 3. nach n auflösen

a)

$$k_1 \cdot 10^6 = 1 \implies k_1 = \underline{10^{-6}}$$

$$k_1 \cdot n = 100 \implies n = \frac{1}{k} \cdot 100 = 10^6 \cdot 100 = \underline{\underline{10^8}}$$

b)

$$k_2 \cdot 10^6 \cdot \log_{10} 10^6 = 1$$

$$\Rightarrow k_2 \cdot 10^6 \cdot 6 = 1 \Rightarrow k_2 = \frac{1}{6} \cdot \underline{10^{-6}}$$

$$\Rightarrow k_2 \cdot n \cdot \log_{10} n = 100$$

$$\Rightarrow n \cdot \log_{10} n = 6 \cdot 10^8 =: c_2$$

$$\Rightarrow 10^{n \cdot \log_{10} n} = 10^{c_2}$$

$$\Rightarrow n^n = 10^{c_2} = 10^{6 \cdot 10^8}$$

Diese Gleichung lässt sich nicht geschlossen lösen.

solver: $n = 76127253$

c)

$$k_3 \cdot (10^6)^2 = 1$$

$$\Rightarrow k_3 \cdot 10^{12} = 1 \Rightarrow k_3 = \underline{10^{-12}}$$

$$k_3 \cdot n^2 = 100$$

$$\Rightarrow \underline{n} = \sqrt{10^{14}} = \underline{\underline{10^7}}$$

d)

$$k_4 \cdot (10^6)^3 = 1$$

$$\Rightarrow k_4 \cdot 10^{18} = 1 \Rightarrow k_4 = \underline{10^{-18}}$$

$$k_4 \cdot n^3 = 100 \Rightarrow n^3 = 10^{20}$$

$$\Rightarrow \underline{n} = 10^{\frac{20}{3}} = \underline{\underline{4641588}}$$

e)

$$k_5 \cdot 10^{10^6} = 1 \Rightarrow k_5 = \frac{1}{\underline{10^{10^6}}}$$

$$k_5 \cdot 10^n = 100$$

$$\Rightarrow 10^n = 100 \cdot 10^{10^6} = 10^{10^6+2}$$

$$\Rightarrow \underline{n} = 10^6 + 2 = \underline{\underline{1000002}}$$

d.h. mit einem 100 Mal schnelleren Rechner können Sie 2 (!) zusätzliche Datensätze berechnen.

3.4 Aufgabe 4: Sortieralgorithmen

Implementieren Sie einen rekursiven Quicksort-Algorithmus, der für Teildaten mit weniger als M Elementen zu Insertionsort (Direktes Einfügen) übergeht.

Bestimmen Sie empirisch den optimalen Wert von M . Nehmen Sie einen Array mit ungefähr fünf Millionen zufällig erzeugten Ganzzahlen an und messen Sie, mit welchem M die Sortierung am schnellsten abgearbeitet wird. Betrachten Sie dabei sowohl das Sortieren der unsortierten als auch der bereits sortierten Liste.

3.4.1 Lösung

Mit dem verwendeten Rechner wurde die Listengröße mit 5 Mio. Elementen des Typs `int` gewählt. Die Liste musste auf dem Heap angelegt werden, beim Einsatz einer Stackvariablen resultierte ein Stackoverflow.

In Abbildung 1 sind die Laufzeitmessungen dargestellt (Execution time in Sekunden). Wenn die Schwelle auf Null gesetzt wird, gibt es keinen Wechsel des Algorithmus, d.h. das ist der reine Quicksort-Algorithmus. Aus den Messungen ist ersichtlich, dass das Sortieren der unsortierten Liste am schnellsten ist, wenn unterhalb einer Listengröße von ca. 30-80 auf den Insertionsort umgestellt wird (obere Kurve 'Unsorted'). Wenn die Liste bereits sortiert ist, dann schneidet der reine Quicksort nicht schlechter ab (mittlere Kurve 'Sorted'). Auf-grund der Kombination dieser beider Kurven ist es sinnvoll, die Schwelle auf ca. 30 zu setzen. Die unterste Kurve stellt das Messrauschen, d.h. die Abweichung bei mehreren Messungen, dar bei der Sortierung von unsortierten Listen. Das Messrauschen ist doch recht hoch.

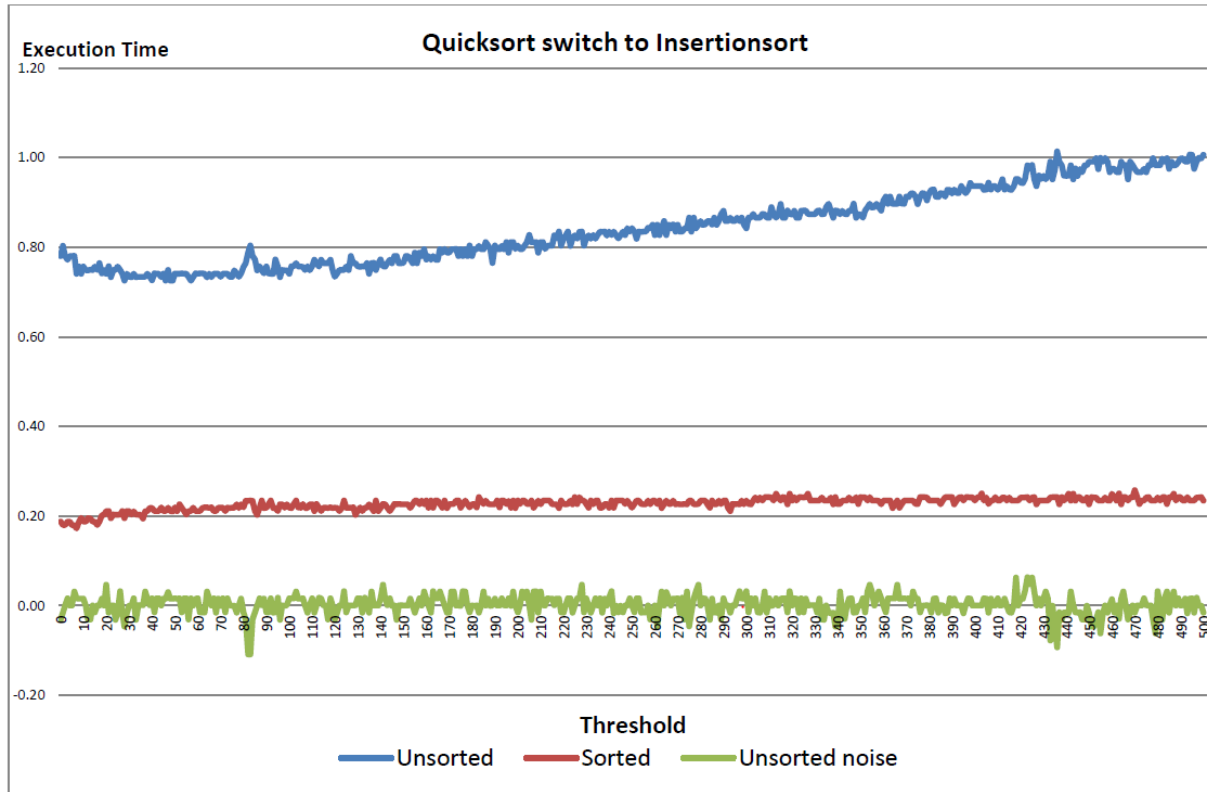


Abbildung 1: Laufzeitmessungen beim implementierten Algorithmus

```
1  /*
2   * List.h
3   *
4   * Created on: 01.03.2017
5   * Author: rbondere
6   */
7
8  #ifndef LIST_H_
9  #define LIST_H_
10
11 class List
12 {
13 public:
14     List(int theSize = 100, int theThreshold = 1, int rSeed =
15         ↪ 47);
16     virtual ~List();
```

```
16     void quickSort(int leftBound,
17                     int rightBound);
18     void insertionSort(int* d,          // start of list
19                       int listLength);
20     void randomFill(); // fill list with random values
21 private:
22     int size; // size of list
23     int threshold;
24     int* data; // dynamic array
25     int randomSeed;
26     void swap(int& a,
27               int& b);
28 };
29
30 #endif /* LIST_H_ */
```

```

1  /*
2  * List.cpp
3  *
4  * Created on: 01.03.2017
5  * Author: rbondere
6  */
7
8  #include <cstdlib>
9  #include <ctime>
10 #include "List.h"
11
12 List::List(int theSize, int theThreshold, int rSeed) :
13     size(theSize), threshold(theThreshold), data(new int[size]),
14     ↪ randomSeed(rSeed)
15 {
16 }
17
18 List::~List()
19 {
20     delete[] data;
21 }
22
23 void List::quickSort(int leftBound,
24                     int rightBound)
25 {
26     if (rightBound-leftBound < threshold)
27         insertionSort(&data[leftBound], rightBound-leftBound+1);
28
29     int left = leftBound; // index of left boundary
30     int right = rightBound; // index of right boundary
31     int pivot = data[(left + right) / 2]; // select middle
32     ↪ element in array
33
34     do
35     {
36         while (data[left] < pivot)
37             ++left;
38         while (data[right] > pivot)
39             --right;
40
41         if (left <= right)
42         {
43             if (left < right)
44                 swap(data[left], data[right]);
45             ++left;
46         }
47     }
48 }

```

```

44     --right;
45 }
46 } while (left <= right);
47
48 if (leftBound < right)
49     quickSort(leftBound, right);
50 if (rightBound > left)
51     quickSort(left, rightBound);
52
53 }
54
55 inline void List::insertionSort(int* d,
56                                 int listLength)
57 {
58     int j;
59     int buf; // Buffer
60
61     for (int i = 1; i < listLength; i++)
62     {
63         buf = d[i];
64         j = i;
65         while (j > 0 && buf < d[j-1])
66         {
67             d[j] = d[j-1];
68             j = j - 1;
69         }
70         d[j] = buf;
71     }
72 }
73
74 void List::randomFill()
75 {
76     srand(randomSeed);
77     for (int i = 0; i < size; ++i)
78         data[i] = rand();
79 }
80
81 void List::swap(int& a,
82                int& b)
83 {
84     int tmp = a;
85     a = b;
86     b = tmp;
87 }

```

```

1  //=====
2  // Name      : QuickSortFast.cpp
3  // Author    : Reto Bonderer
4  // Version   :
5  // Copyright : (c) HSR R. Bonderer
6  // Description : Fast Quicksort
7  //=====
8
9  #include <iostream>
10 #include "StopWatch.h"
11 #include "List.h"
12 using namespace std;
13
14 int main()
15 {
16     int nr = 5000000;
17     int threshold;
18     /*
19      cout << "Anzahl Listenelemente: ";
20      cin >> nr;
21      */
22     cout << "Schwelle fuer Umschaltung nach Insertionsort: ";
23     cin >> threshold;
24
25     List myList(nr, threshold);
26     myList.randomFill();
27
28     StopWatch t;

```

```

29     myList.quickSort(0, nr - 1);
30     cout << "Unsortierte Liste sortiert in " << t.elapsed() << "
31     ↪ sec" << endl;
32
33     StopWatch t1;
34     myList.quickSort(0, nr - 1);
35     cout << "Sortierte Liste sortiert in " << t1.elapsed() <<
36     ↪ " sec" << endl;
37
38     // for (threshold = 0; threshold <= 500; threshold++)
39     // {
40     //     List myList(nr, threshold);
41     //     myList.randomFill();
42     //     StopWatch t;
43     //     myList.quickSort(0, nr - 1);
44     //     cout << threshold << " " << t.elapsed();
45     //     StopWatch t1;
46     //     myList.quickSort(0, nr - 1); // already sorted
47     //     cout << " " << t1.elapsed() << endl;
48     // }
49
50
51     cout << "Fertig" << endl;
52     return 0;
53 }

```

```

1  /*
2  * StopWatch.h
3  *
4  * Created on: 04.03.2013

```

```

5  * Author: rbondere
6  */
7
8  #ifndef STOPWATCH_H_

```

```

9  #define STOPWATCH_H_
10
11 class Stopwatch
12 {
13     public:
14         Stopwatch();
15         double elapsed() const; // returns elapsed time since

```

```

16     private:
17         double startTime; // unit: seconds
18 };
19
20 #endif /* STOPWATCH_H_ */

```

```

1  /*
2  * Stopwatch.cpp
3  *
4  * Created on: 04.03.2013
5  * Author: rbondere
6  */
7
8  #include "StopWatch.h"
9  #include <sys/resource.h>
10 #include <sys/types.h>
11
12 Stopwatch::StopWatch()
13 {
14     rusage tp;

```

```

15     getrusage(RUSAGE_SELF, &tp);
16     startTime = static_cast<double>(tp.ru_utime.tv_sec) +
17                 static_cast<double>(tp.ru_utime.tv_usec)/1E6;
18 }
19
20 double Stopwatch::elapsed() const
21 {
22     rusage tp;
23     getrusage(RUSAGE_SELF, &tp);
24     return static_cast<double>(tp.ru_utime.tv_sec) +
25            static_cast<double>(tp.ru_utime.tv_usec)/1E6 -
26            ↪ startTime;

```

4 Lab 4 Zufallszahlengeneratoren

4.1 Aufgabe 1: Verteilung der Zufallswerte durch rand()

`rand()` liefert gleichverteilte Werte zwischen 0 und inklusive `RAND_MAX`, d.h. im Intervall $[0, \text{RAND_MAX}]$. Untersuchen Sie, wie gut diese Gleichverteilung eingehalten wird. Dazu soll das Testprogramm zufällige Zahlen im Bereich von $[1, 20]$ liefern. Speichern Sie die Anzahl in einem Array und stellen Sie das Resultat in einem GUI oder mit Hilfe von Excel dar. Hinweis: die man-Page für die Funktion `rand()` erhalten Sie mit Hilfe des Befehls `man 3 rand`

4.1.1 Lösung

Das Eclipse-Projekt ist in `./Loesung/RandDist` zu finden. Bei einer Umrechnung in einen bestimmten Bereich ist wichtig, dass alle Werte, insbesondere auch der grösste und der kleinste, gleichverteilt sind. Die einfachste Umrechnung kann mit Hilfe des Modulooperators `%` erreicht werden. Allerdings werden dann nur die niederwertigsten Bits der Zufallszahl berücksichtigt. Dies kann zu wenig zufälligen Folgen führen. Beachten Sie auch die Kommentare im Code.

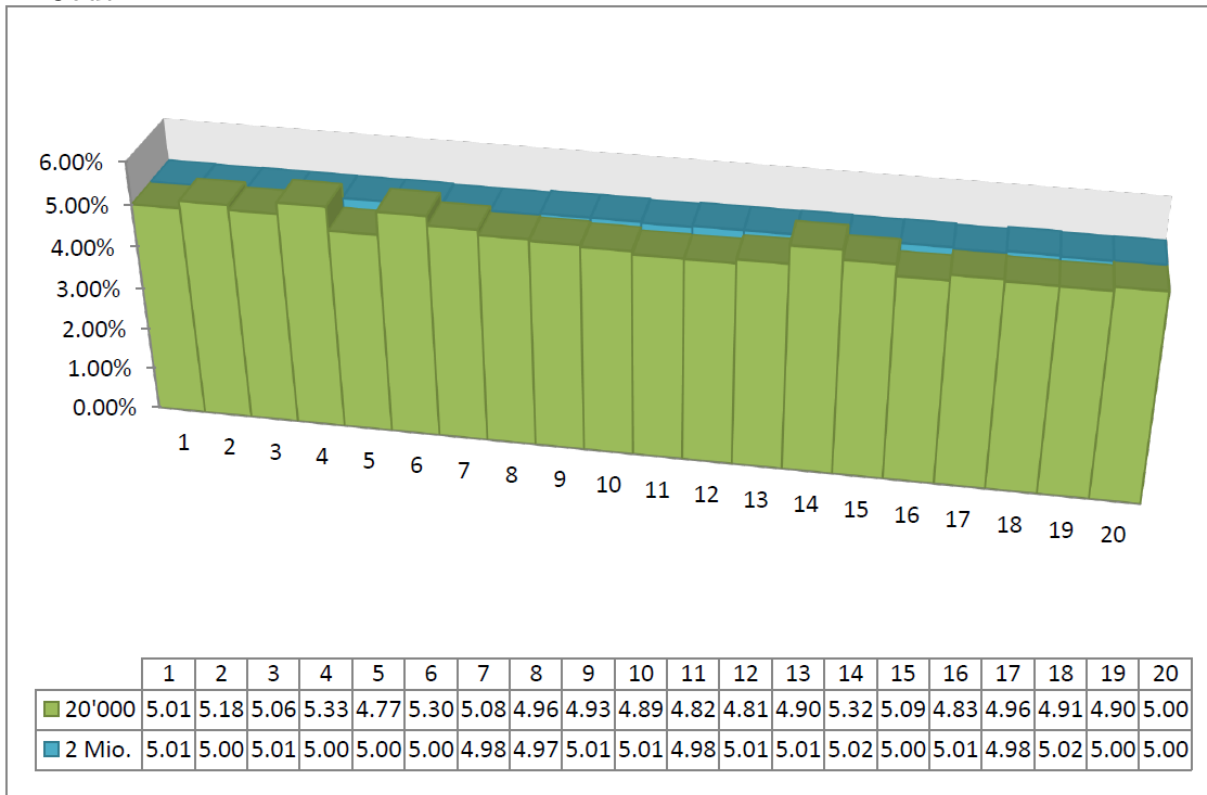


Abbildung 1: Verteilung der Zufallszahlen

In der folgenden Abbildung sind die Verteilungen bei 20'000, bzw. bei 2'000'000 Zufallszahlen dargestellt.

```
1 //=====
2 // Name      : RandDist.cpp
3 // Author    : Reto Bonderer
4 // Version   :
5 // Copyright : (c) HSR R. Bonderer
6 // Description : Uniform distribution
7 //=====
8
9 #include <iostream>
10 #include <iomanip>
11 #include <cstdlib>
12 using namespace std;
13
14 enum
15 {
16     minValue = 1, // minimal value for random numbers
17     maxValue = 20, // maximal value for random numbers
18     iterations = 2000000 // number of iterations
19 };
20
21 int main()
22 {
23     int histogram[maxValue - minValue + 1] = {0};
24     int r;
25     srand(time(0));
26     for (int i = 0; i < iterations; ++i)
27     {
28         // r = rand() % (maxValue - minValue + 1) + minValue;
29         // Modulo-Variante beruecksichtigt nur LSBs. Kann zu
30         //   schlechten (kaum
31         //   zufaelligen) Folgen fuehren
32
33         r = (int)(static_cast<double>(rand()) / (RAND_MAX + 1.0)
34             * (maxValue - minValue + 1) + minValue);
35         // '+ 1' ist notwendig, weil nach int konvertiert wird.
36         // Vor Konversion gibt es dadurch bei [1,20] Zahlen im
37         //   Bereich [0.0,20.0[
38         // Nach Konversion werden nur die ganzzahligen Anteile
39         //   verwendet
40         // Das hinterste "+ minValue" muss ausserhalb des
41         //   Typecasts sein,
42         //   andernfalls ergibt es ein Ueberhoehung bei 0, falls
43         //   minValue < 0
44         // cout << setw(2) << r << " | ";
45     }
```

```

39 ++histogram[r - minValue];
40 }
41 for (int i = 0; i < maxValue - minValue + 1; i++)
42     cout << histogram[i] << " ";
43     cout << endl;
44
45     cout << "Soll: je " << iterations / (maxValue - minValue +
46         ↳ 1) << endl;
47     return 0;

```

4.2 Aufgabe 2: Güte von Zufallszahlengeneratoren

Zufallszahlen werden oft als Zahlenreihen implementiert, die mit der folgenden Formel beschrieben werden können:

$$r_{i+1} = (A \cdot r_i) \% m \tag{1}$$

Die Formel besteht aus einem ganzzahligen Startwert r_0 und mit ganzzahligen Konstanten a (Multiplikator), c (Inkrement) und m (Modulus). Diese Konstanten sind sorgfältig zu wählen, damit die Zahlen zufällig werden. So erzeugte Zahlenfolgen haben immer eine Periode $\leq m$. Bei gegebenen Konstanten und Startwert r_0 ist die Folge eindeutig festgelegt.

Zur Beurteilung der Güte von Zufallszahlen existieren in der Statistik verschiedene Verfahren. In dieser Aufgabe sollen Sie mit Hilfe des Computers einen graphischen Gütetest entwickeln und damit verschiedene Zufallszahlengeneratoren testen. Verwenden Sie dazu Qt. Sie finden ein Qt Creator Projekt als Vorgabe.

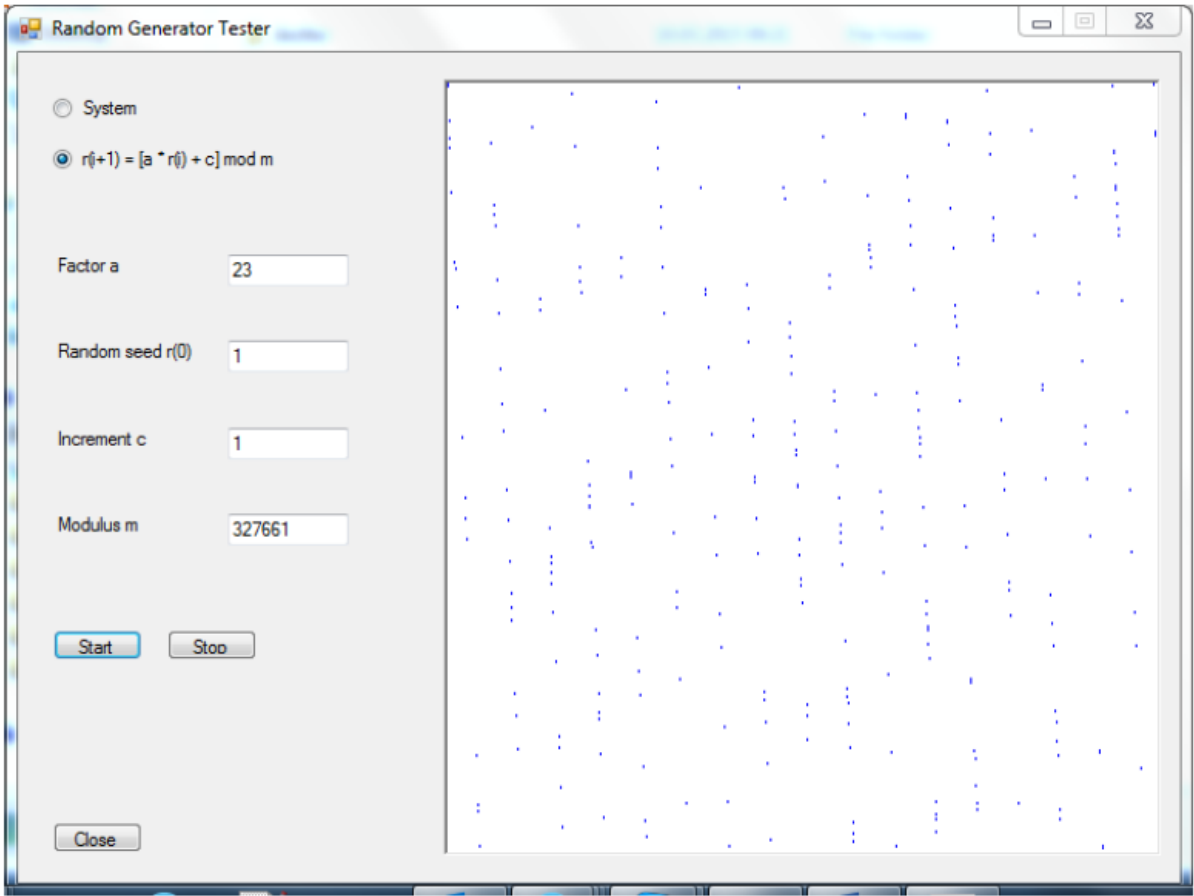


Abbildung 1: Beispielapplikation des Zufallszahlengenerator-Testers

1. Im graphischen Fenster soll ein Quadrat gezeichnet werden. Mittels zweier aufeinanderfolgender Zufallszahlen, die geeignet skaliert werden, wird ein Koordinatenpaar in diesem Quadrat festgelegt (1. Zufallszahl: x-Wert, 2. Zufallszahl: y-Wert) und an der entsprechenden Stelle ein Punkt gezeichnet. Im nächsten Schritt wird der y-Wert zum neuen x-Wert und für den neuen y-Wert muss eine neue Zufallszahl erzeugt werden. Dieser Vorgang wird beliebig wiederholt. Durch die Beobachtung der graphischen Anordnungen zeigen im allgemeinen ein allzu ausgeprägtes Muster. Damit das fortlaufende Zeichnen der Punkte verfolgt werden kann, wird ein Timer eingesetzt. Als Timerintervall wird 10 ms gewählt. Jedesmal wenn sich der Timer meldet, soll ein Punkt gezeichnet werden. In Ihrem Programm sollen die Werte r_0 , a , c und m eingegeben werden können. Auf dem Skriptserver finden Sie als Muster eine vollständige ausführbare Anwendung (siehe Abbildung 1).

2. Testen Sie z.B. die folgenden Zufallszahlengeneratoren:

| a | r_0 | c | m |
|------------|-------|-------|--------|
| 2 | 0 | 1 | 28 |
| 8 | 0 | 1 | 7 |
| 125 | 0 | 0 | 8192 |
| 21 | 1 | 3 | 64 |
| 25 | 1 | 1 | 64 |
| 3 | 1 | 1 | 32766 |
| 130 | 1 | 1 | 32766 |
| 1331 | 1 | 1 | 327661 |
| 1103515245 | 1 | 12345 | 32768 |

3. Testen Sie den Zufallszahlengenerator des Systems: `rand()`. Übrigens: mit Hilfe der C-Funktion `srand()` wird der Startwert r_0 , der random seed, festgelegt

4.2.1 Lösung

```
1 #include "randomtester.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
```

```
7     RandomTester w;
8     w.show();
9
10    return a.exec();
11 }
```

```
1 #ifndef RANDOMTESTER_H
2 #define RANDOMTESTER_H
3
4 #include <QMainWindow>
5 #include <QTimer>
6
7 #include <randomviewer.h>
8
9 namespace Ui {
10 class RandomTester;
11 }
12
13 class RandomTester : public QMainWindow
14 {
15     Q_OBJECT
16
17 public:
18     /**
19      * @brief RandomTester Ctor
20      * @param parent Parent
21      */
22     explicit RandomTester(QWidget *parent = 0);
23
24     /**
25      * @brief ~RandomTester Dtor
26      */
27     ~RandomTester();
28
29 public slots:
30     /**
31      * @brief onTimeout Draws a new random point
32      */
33     void onTimeout();
34
35     /**
36      * @brief onStartStopClicked Starts / stops the timer
37      */
38     void onStartStopClicked();
39
40     /**
```

```
41     * @brief onRadioClicked Select if the system random
42     *   ↳ generator or
43     * the equation  $r_{i+1} = (a * r_i + c) \bmod m$  is used to
44     *   ↳ generate the random values.
45     */
46     void onRadioClicked();
47
48 private:
49     bool validateParameters();
50     void invalidateParameters();
51
52     enum RandomSource
53     {
54         srcSystem,
55         srcModulo
56     };
57
58     Ui::RandomTester *ui;
59     RandomViewer* viewer; /// RandomViewer to draw the random
60     * points on the pixmap
61     QTimer* timer;
62
63     RandomSource randomSrc;
64
65     int xLim; /// x limit of the RandomViewer pixmap
66     int yLim; /// y limit of the RandomViewer pixmap
67
68     int x;
69     int y;
70
71     bool first;
72
73     ///  $r = (a * r + c) \bmod m$ 
74     long factor; /// a
75     long r; /// r
76     long increment; /// c
77     long modulus; /// m
78 };
79
80 #endif // RANDOMTESTER_H
```

```

1  #include <QTime>
2
3  #include "randomtester.h"
4  #include "ui_randomtester.h"
5
6  RandomTester::RandomTester(QWidget *parent) :
7      QMainWindow(parent),
8      ui(new Ui::RandomTester),
9      viewer(new RandomViewer),
10     randomSrc(srcModulo),
11     first(true)
12 {
13     ui->setUpUi(this);
14
15     switch (randomSrc)
16     {
17     case srcSystem:
18         ui->radioSystem->setChecked(true);
19         ui->radioModulo->setChecked(false);
20         break;
21     default:
22     case srcModulo:
23         ui->radioSystem->setChecked(false);
24         ui->radioModulo->setChecked(true);
25         break;
26     }
27     onRadioClicked();    // invoke function to update group box
28                          ↪ enabling
29
30     timer = new QTimer();
31     timer->setInterval(10);
32     connect(timer, SIGNAL(timeout()), this, SLOT(onTimeout()))
33     ↪ ;
34
35     // Inititalize graphics window
36     ui->centralWidget->layout()->addWidget(viewer);
37
38     xLim = viewer->size().width();
39     yLim = viewer->size().height();
40
41     // Initialize system random
42     qsrand(QDateTime::currentDateTime().toMsecsSinceEpoch());
43
44     connect(ui->btnStartStop, SIGNAL(clicked()), this, SLOT(
45     ↪ onStartStopClicked()));
46     connect(ui->radioSystem, SIGNAL(clicked()), this, SLOT(
47     ↪ onRadioClicked()));
48     connect(ui->radioModulo, SIGNAL(clicked()), this, SLOT(
49     ↪ onRadioClicked()));
50 }
51
52 RandomTester::~RandomTester()
53 {
54     if (timer->isActive())
55         timer->stop();
56
57     delete ui;
58     delete timer;
59     delete viewer;
60 }
61
62 void RandomTester::onTimeout()
63 {
64     x = y;
65     switch (randomSrc)
66     {
67     case srcSystem:
68         y = (1.0 * qrand() / RAND_MAX) * yLim;
69         break;
70     default:
71     case srcModulo:
72         r = (factor * r + increment) % modulus;
73         y = static_cast<int>(r) * yLim / modulus;
74         break;
75     }
76     if (!first)
77         viewer->addPoint(QPoint(x, y));
78     first = false;
79 }
80
81 void RandomTester::onStartStopClicked()
82 {
83     if (timer->isActive())
84
85         timer->stop();
86         ui->btnStartStop->setText("Start");
87         if (ui->radioModulo->isChecked())
88             ui->gbParameters->setEnabled(true);
89         ui->gbSource->setEnabled(true);
90     }
91     else
92     {
93         if (validateParameters())
94         {
95             viewer->clear();
96             first = true;
97             timer->start();
98             ui->btnStartStop->setText("Stop");
99             ui->gbSource->setEnabled(false);
100            ui->gbParameters->setEnabled(false);
101        }
102    }
103 }
104
105 void RandomTester::onRadioClicked()
106 {
107     if (ui->radioSystem->isChecked())
108     {
109         ui->gbParameters->setEnabled(false);
110         randomSrc = srcSystem;
111     }
112     else if (ui->radioModulo->isChecked())
113     {
114         ui->gbParameters->setEnabled(true);
115         randomSrc = srcModulo;
116     }
117 }
118
119 bool RandomTester::validateParameters()
120 {
121     bool res;
122     bool resAll = true;
123
124     invalidateParameters();
125
126     factor = ui->leFactor->text().toInt(&res);
127     if (!res)
128     {
129         ui->leFactor->setStyleSheet("background-color: red;");
130         resAll = false;
131     }
132
133     r = ui->leSeed->text().toInt(&res);
134     if (!res)
135     {
136         ui->leSeed->setStyleSheet("background-color: red;");
137         resAll = false;
138     }
139
140     increment = ui->leIncrement->text().toInt(&res);
141     if (!res)
142     {
143         ui->leIncrement->setStyleSheet("background-color: red;
144         ↪ ");
145         resAll = false;
146     }
147
148     modulus = ui->leModulus->text().toInt(&res);
149     if (!res)
150     {
151         ui->leModulus->setStyleSheet("background-color: red;")
152         ↪ ;
153         resAll = false;
154     }
155
156     return resAll;
157 }
158
159 void RandomTester::invalidateParameters()
160 {
161     ui->leFactor->setStyleSheet("");
162     ui->leSeed->setStyleSheet("");
163     ui->leIncrement->setStyleSheet("");
164     ui->leModulus->setStyleSheet("");
165 }

```

```

1  #ifndef RANDOMVIEWER_H
2  #define RANDOMVIEWER_H
3
4  #include <QFrame>
5  #include <QPixmap>
6  #include <QSize>
7  #include <QPen>
8
9  class RandomViewer : public QFrame
10 {
11     Q_OBJECT
12 public:
13     /**
14      * @brief RandomViewer Ctor
15      * @param s Size of the pixmap
16      * @param parent Parent
17      */
18     explicit RandomViewer(QSize s = QSize(600, 600), QWidget *
19         ↪ parent = 0);
20
21     /**
22      * @brief addPoint Draws the point in the pixmap
23
24      * @param p Point to draw
25      */
26     void addPoint(QPoint p);
27
28     /**
29      * @brief clear Clears the pixmap
30      */
31     void clear();
32
33 signals:
34
35 public slots:
36
37 protected:
38     void paintEvent(QPaintEvent *);
39
40 private:
41     QPen pen;
42     QPixmap pixmap;
43 };
44
45 #endif // RANDOMVIEWER_H

```

```

1  #include <QPainter>
2  #include <QPoint>
3
4  #include "randomviewer.h"
5
6  RandomViewer::RandomViewer(QSize s, QWidget *parent) :
7      QFrame(parent)
8  {
9      setFixedSize(s);
10     setFrameStyle(QFrame::Box);
11     pen.setColor(Qt::white);
12
13     pixmap = QPixmap(size());
14     pixmap.fill(Qt::black);
15 }
16
17 void RandomViewer::addPoint(QPoint p)
18 {
19     QPainter painter(&pixmap);
20     painter.setPen(pen);
21     painter.drawPoint(p);
22     update();
23 }
24
25 void RandomViewer::clear()
26 {
27     pixmap.fill(Qt::black);
28 }
29
30 void RandomViewer::paintEvent(QPaintEvent *event)
31 {
32     QPainter p(this);
33     p.drawPixmap(0, 0, pixmap);
34     QFrame::paintEvent(event);
35 }

```

4.3 Aufgabe 3: Nicht ideale Münze (biased coin)

Eine ideale (faire) Münze liefert je zur Hälfte Kopf und Zahl, d.h. $p(K) = p(Z) = 0.5$. Wir nehmen nun an, die Münze sei nicht ideal, z.B. $p(K) = 0.6, p(Z) = 0.4$.

1. In `./Vorgabe/Coin/biasedCoin.cpp` finden Sie die Funktion `biasedCoin()`, welche eine unfaire Münze implementiert. Studieren Sie den Code und verifizieren Sie, ob die Verteilung den Erwartungen entspricht.
2. Entwickeln Sie einen Algorithmus, der mit einer beliebig konstant unfairen Münze einen gleichverteilten Zufallsprozess erreichen kann. Implementieren Sie den Code, dabei müssen Sie die vorgegebene Funktion `biasedCoin()` nutzen und dürfen diese nicht abändern. Verifizieren Sie Ihren Algorithmus.

Hinweis: Sie müssen die Münze zweimal hintereinander werfen.

4.3.1 Lösung

1. Der Output in die Shell kann z.B. so aussehen:
0: 5998379 (59.9838 %)
1: 4001621 (40.0162 %)
2. Wenn mit den gegebenen Verteilungen $p(0) = 0.6, p(1) = 0.4$ zweimal hintereinander gewürfelt wird, dann resultiert die folgende Verteilung:
 $p(00) = 0.6 \cdot 0.6 = 0.36$
 $p(11) = 0.4 \cdot 0.4 = 0.16$
 $p(01) = 0.6 \cdot 0.4 = 0.24$
 $p(10) = 0.4 \cdot 0.6 = 0.24$

Der Algorithmus sieht deshalb wie folgt aus: Es muss zweimal hintereinander gewürfelt werden. Wenn die erste Zahl 0, die zweite 1 ist, dann wird 0 zurückgegeben. Wenn die erste Zahl 1, die zweite 0 ist, dann wird 1

zurückgegeben. Wenn die beiden Zahlen gleich sind, dann müssen zwei (!) weitere Zahlen gewürfelt werden. Die letzte Zahl darf nicht be-halten und nur eine zusätzliche Münze geworfen werden. Wieso?

Weil dann mit 60 % eine 0 kommen würde. Das würde das Resultat verfälschen.

Der Code ist unter ./Loesung/Coin/fairCoin.cpp zu finden.

Der Output kann nun so aussehen:

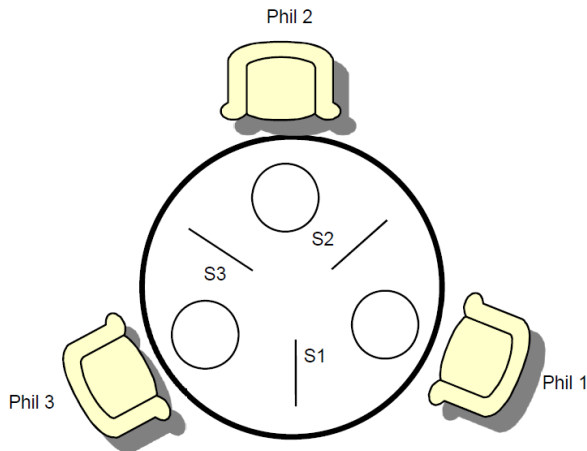
0: 5000155 (50.0016 %)

1: 4999845 (49.9984 %)

```
1 //=====
2 // Name      : biasedCoin.cpp
3 // Author    : Reto Bonderer
4 // Version   : 20190528
5 // Copyright : (c) HSR R. Bonderer
6 // Description : Get a uniform distribution from a biased coin
7 //=====
8
9 #include <iostream>
10 #include <iomanip>
11 #include <cstdlib>
12 using namespace std;
13
14 enum
15 {
16     iterations = 10000000 // number of iterations
17 };
18
19 // returns a biased coin value (0, 1)
20 unsigned int biasedCoin();
21
22 // returns a fair coin value (0, 1) using biasedCoin()
23 unsigned int fairCoin();
24
25 int main()
26 {
27     unsigned int histogram[2] = {0};
28     srand(time(0));
29
30     for (unsigned int i = 0; i < iterations; ++i)
31     {
32         ++histogram[biasedCoin()];
33     }
34
35     for (int i = 0; i < 2; ++i)
36     {
37         cout << i << ": " << setw(10) << histogram[i] << " ("
38             << histogram[i]*100.0/iterations << " %)" << endl;
39     }
40
41     return 0;
42 }
43
44 unsigned int biasedCoin()
45 {
46     enum
47     {
48         bias = 60 // p(0) = bias in percent, p(1) = 1-p(0)
49     };
50
51     if (rand() < RAND_MAX/100*bias)
52         return 0;
53     else
54         return 1;
55 }
56
57 unsigned int fairCoin()
58 {
59     while (1)
60     {
61         unsigned int first = biasedCoin();
62         unsigned int second = biasedCoin();
63         if (first == 0 && second == 1)
64             return 0;
65         else if (first == 1 && second == 0)
66             return 1;
67     }
68 }
```

5 Lab 5 Dinierende Philosophen

Das Problem der dinierenden Philosophen ist ein klassisches Synchronisationsproblem. In Abbildung 1 ist die Konfiguration mit drei Philosophen dargestellt. Das Problem lautet wie folgt: Einige Philosophen sitzen an einem runden Tisch und essen Nudeln. Zu ihrer Linken und Rechten befindet sich je ein Stäbchen, das sie mit ihrem Nachbarn teilen. Insgesamt befinden sich nur so viele Stäbchen wie Philosophen auf dem Tisch. Die Philosophen machen den ganzen Tag nichts anderes als zu denken, zu essen, wieder zu denken, etc. Wenn ein Philosoph essen will, muss er beide Stäbchen benutzen. Wenn einer seiner Nachbarn ein Stäbchen besitzt, muss er warten, bis beide Stäbchen frei sind. Obwohl alle Philosophen unterschiedlich schnell denken, kann der Fall eintreten, dass mehrere gleichzeitig essen wollen.



5.1 Aufgabe 1: Untersuchung von Deadlocks

Untersuchen Sie die folgenden Fälle auf Verklemmungen (Deadlocks). Formulieren Sie eine verklemmungsfreie Regel, bzw. welche der Varianten würden Sie implementieren? Die Regeln müssen so gestaltet werden, dass die Philosophen während der "Synchronisation" ohne Absprachen untereinander auskommen, d.h. bei Kenntnis dieser Regeln kann jeder Philosoph selbständig handeln.

1. Jeder hungrige Philosoph nimmt seine beiden Stäbchen gleichzeitig auf, sobald beide verfügbar sind. Falls nur eines verfügbar ist, lässt er dieses auf dem Tisch liegen.
2. Jeder hungrige Philosoph nimmt zuerst das Stäbchen links von seinem Teller auf. Das Stäbchen rechts von seinem Teller nimmt er nur dann auf, wenn er das linke schon hat.
3. Alle Stäbchen werden durchnummeriert. Jeder hungrige Philosoph nimmt zuerst das Stäbchen mit der kleineren Nummer bei seinem Teller. Das Stäbchen mit der grösseren Nummer nimmt er nur dann auf, wenn er das Stäbchen mit der kleineren Nummer schon besitzt.

5.1.1 Lösung

1. Diese Variante funktioniert. Entweder nimmt der Philosoph gar kein Stäbchen oder zwei aufs Mal. Er muss einfach noch beide aufs Mal fassen können. Dass könnte zu Problemen führen.
2. Diese Variante funktioniert überhaupt nicht. Wenn alle Philosophen Hunger haben, nimmt jeder das linke Stäbchen und alle warten auf das zweite, welches aber keiner freigeben wird. Ein typischer Deadlock
3. Diese Variante funktioniert, obwohl sie sehr ähnlich zu Variante b) aussieht. Die Regel gegen Deadlock lautet, dass alle die Ressourcen in der ganz genau gleichen Reihenfolge anfordern müssen. Zuerst links, dann rechts gemäss b) widerspricht dem. In b) will Phil 1 zuerst S1, dann S2, Phil 2 nimmt zuerst S2, dann S3, Phil 3 nimmt hingegen zuerst S3, dann S1. Hier liegt das Problem. In c) fordern Phil 1 und Phil 2 die Ressourcen genau gleich an wie in b). Phil 3 fordert aber zuerst S1 und dann S3 an, d.h. immer in derselben Reihenfolge.

5.2 Aufgabe 2: Implementation der dinierenden Philosophen

Im Verzeichnis `./Vorgabe/Philo` finden Sie eine Codevorgabe für fünf Philosophen. Das `main()` ist sehr komfortabel: es müssen nur Objekte der Klasse `Philosopher` definiert werden. Diese Klasse bietet nur die beiden Elementfunktionen `live()` und `join()` an. Alles weitere, insbesondere die Erzeugung des Threadkontexts, ist intern in der Klasse umgesetzt. Studieren Sie den Code in `Philosopher.h`.

```
1 {
2 public:
3     Philosopher(int pid, int thinkdelay, int eatDelay, Sticks& s);
4     ~Philosopher();
5     void live();           // the philosopher's life
6     void join();          // wait for philosopher to leave
7 private:
8     enum {nMeals = 3};
9     int id;               // this philosopher's id
10    int tDelay;            // how long does this philosopher think?
11    int eDelay;            // how long does this philosopher eat?
12    int left;              // left fork number
13    int right;             // right fork number
14    Sticks& stick;         // sticks used by all philosophers
15    pthread_attr_t attr;
16    pthread_t tid;         // thread id
17    void lifeThread();     // the (C++) - thread function
18    static void* staticWrapper(void* p) // C Wrapper for pthread_create()
19    {                       // p must be the this pointer
20        static_cast<Philosopher*>(p) -> lifeThread();
21        return 0;
22    }
23 };
```

1. Erläutern Sie die Elementfunktion `Philosopher::lifeThread()`. Wozu dient sie?
2. Erklären Sie die Funktion `Philosopher::staticWrapper()`. Wieso braucht es diese Funktion, was ist ihre Aufgabe und welche Beziehung hat diese Funktion zu `Philosopher::lifeThread()`?
3. Schreiben Sie ein Programm in C++, welches das Philosophenleben simuliert und für alle fünf Philosophen gleich fair ist. Ein Deadlock darf nie eintreten. Implementieren Sie jeden Philosophen als Thread in `Philosopher::lifeThread()`. Die Denkphase können Sie mit einer jeweils zufällig gewählten Schlafdauer umsetzen. Den Zugriff auf die Stäbchen müssen Sie in der Klasse `Sticks` als Monitor implementieren, d.h. alle Locks müssen in dieser Klasse mit `pthread_mutex_lock()`, bzw. `pthread_mutex_unlock()` gemacht werden, der Aufrufer muss sich nicht darum kümmern müssen. Eine Condition Variable braucht es hier ebenfalls.
4. In der Monitorklasse `Sticks` verwenden Sie bisher direkt die C-Funktionen `pthread_mutex_lock()`, bzw. `pthread_mutex_unlock()`. Setzen Sie jetzt RAII (Resource Acquisition Is Initialization) ein. Beachten Sie dazu auch die Klasse `ResourceLock` aus dem Praktikum 10 (Aufgabe 3) vom Modul Embedded Software Engineering 1.

5.2.1 Lösung

1. Die Elementfunktion `Philosopher::lifeThread()` ist die Threadfunktion, d.h. sie beinhaltet den Code, den ein Thread auszuführen hat. Sie hat den Zugriff auf sämtliche (auch privaten) Attribute und Funktionen der Klasse `Philosopher`. Sie zeigt, wie auch in C++ eine Elementfunktion als Threadfunktion genutzt werden kann.
2. Ein Thread muss mit der C-Funktion `pthread_create()` gestartet werden. Diese Funktion benötigt als Parameter einen Funktionspointer auf eine C-Funktion, welche die Threadfunktion beinhaltet. Die statische C++-Funktion `Philosopher::staticWrapper()` liegt ausserhalb des Klassenkontextes und kann der Funktion `pthread_create()` übergeben werden. Der Static Wrapper ruft im Objektkontext die Elementfunktion `Philosopher::lifeThread()` auf. Den Objektkontext erhält der Wrapper über den void-Pointer. Deshalb muss der Funktion `pthread_create()` der this-Pointer (das ist der Objektkontext) übergeben werden: `pthread_create(&tid, &attr, staticWrapper, this);`
3. siehe `./Loesung/Philo` Bei der Ausführung bemerken Sie allenfalls, dass `cout` nicht thread-safe ist. Einzelne Texte mögen auseinandergerissen (interleaved) sein.

```

1  /*
2  * Sticks.h
3  *
4  * Dining Philosophers: Sticks class (Monitor)
5  *
6  * Created on: 07.04.2016
7  * Author: Reto Bonderer
8  */
9  #ifndef STICKS_H_
10 #define STICKS_H_
11
12 class Sticks
13 {
14 public:
15     Sticks(int nr = 5);

```

```

16     ~Sticks();
17     int getNr() const {return nSticks;}
18     void put(int left, int right); // lay down left and
19         ↪ right stick
20     void get(int left, int right); // try to pick left and
21         ↪ right stick
22 private:
23     int nSticks;
24     volatile bool* stick; // ptr to dynamic array
25     pthread_mutex_t stickMutex;
26     pthread_cond_t stickCv;
27 };
28 #endif

```

```

1  /*
2  * Sticks.cpp
3  *
4  * Dining Philosophers: Sticks class (Monitor)
5  *
6  * Created on: 07.04.2016
7  * Author: Reto Bonderer
8  */
9
10 #include <pthread.h>
11 #include "Sticks.h"
12
13 Sticks::Sticks(int nr)
14 : nSticks(nr),
15   stick(new bool[nSticks])
16 {
17     for (int i = 0; i < nSticks; ++i)
18         stick[i] = false;
19
20     // Initialize mutex and condition variable objects
21     pthread_mutex_init(&stickMutex, 0);
22     pthread_cond_init (&stickCv, 0);
23 }
24
25 Sticks::~Sticks()
26 {

```

```

27     delete[] stick;
28     pthread_mutex_destroy(&stickMutex);
29     pthread_cond_destroy(&stickCv);
30 }
31
32 void Sticks::put(int left, int right)
33 {
34     pthread_mutex_lock(&stickMutex); // start of CS
35     stick[left] = false;
36     stick[right] = false;
37     pthread_cond_signal(&stickCv);
38     pthread_mutex_unlock(&stickMutex); // end of CS
39 }
40
41 void Sticks::get(int left, int right)
42 {
43     pthread_mutex_lock(&stickMutex); // start of CS
44     while (stick[left] || stick[right])
45     {
46         pthread_cond_wait(&stickCv, &stickMutex);
47     }
48     stick[left] = true;
49     stick[right] = true;
50     pthread_mutex_unlock(&stickMutex); // end of CS
51 }

```

```

1  /*
2  * Philosopher.h
3  *
4  * Dining Philosophers: Philosopher class
5  *
6  * Created on: 20.12.2016
7  * Author: Reto Bonderer
8  */
9  #ifndef PHILOSOPHER_H_
10 #define PHILOSOPHER_H_
11 #include <pthread.h>
12 #include "Sticks.h"
13
14 class Philosopher
15 {
16 public:
17     Philosopher(int pid, int thinkdelay, int eatDelay,
18         ↪ Sticks& s);
19     ~Philosopher();
20     void live(); // the philosopher's life
21     void join(); // wait for philosopher to leave
22 private:
23     enum {nMeals = 3};

```

```

23     int id; // this philosopher's id
24     int tDelay; // how long does this philosopher
25         ↪ think?
26     int eDelay; // how long does this philosopher
27         ↪ eat?
28     int left; // left fork number
29     int right; // right fork number
30     Sticks& stick; // sticks used by all philosophers
31     pthread_attr_t attr;
32     pthread_t tid; // thread id
33     void lifeThread(); // the (C++) - thread function
34     static void* staticWrapper(void* p) // C Wrapper for
35         ↪ pthread_create()
36     {
37         // p must be
38         ↪ the this pointer
39         static_cast<Philosopher*>(p)->lifeThread()
40         ↪ ;
41         return 0;
42     }
43 };
44 #endif

```

```

1  /*
2  * Philosopher.cpp
3  *
4  * Dining Philosophers: Philosopher class
5  *
6  * Created on: 24.03.2020
7  * Author: Reto Bonderer
8  */
9  #include <iostream>
10 #include <unistd.h>

```

```

11 #include "Philosopher.h"
12 #include "Sticks.h"
13 using namespace std;
14
15 Philosopher::Philosopher(int pid, int thinkDelay, int
16     ↪ eatDelay, Sticks& s)
17 : id(pid),
18   tDelay(thinkDelay),
19   eDelay(eatDelay),
20   stick(s)

```

```

20 {
21     left = id==1? stick.getNr() - 1: id-2; // follow the anti
        ↳ deadlock strategy
22     right = id-1;
23 }
24
25 Philosopher::~Philosopher()
26 {
27     pthread_attr_destroy(&attr);
28 }
29
30 void Philosopher::live()
31 {
32     // For portability, explicitly create threads in a
        ↳ joinable state
33     pthread_attr_init(&attr);
34     pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE
        ↳ );
35     pthread_create(&tid, &attr, staticWrapper, this); // call
        ↳ the static wrapper
36 }
37
38 void Philosopher::join()
39 {
40     pthread_join(tid, 0);
41 }
42
43 void Philosopher::lifeThread()
44 {
45     for (int i = 0 ; i < nMeals; ++i)
46     {
47         cout << "Philosopher " << id << " is thinking..." <<
            ↳ endl;
48         usleep(tDelay);
49         cout << "Philosopher " << id << " likes to eat..." <<
            ↳ endl;
50         stick.get(left, right);
51         cout << "Philosopher " << id << " is eating..." << endl;
52         usleep(eDelay);
53         cout << "Philosopher " << id << " puts sticks back..."
            ↳ << endl;
54         stick.put(left, right);
55     }
56     cout << "Philosopher " << id << " leaves." << endl;
57     pthread_exit(0);
58 }

```

```

1  /*
2  * PhiloTest.cpp
3  *
4  * Dining Philosophers: Philosopher test program
5  *
6  *
7  * Created on: 24.03.2020
8  * Author: Reto Bonderer
9  */
10 #include <pthread.h>
11 #include <unistd.h>
12 #include <iostream>
13 #include "Philosopher.h"
14 #include "Sticks.h"
15 using namespace std;
16
17 enum {numPhilos = 5};
18
19 int main()
20 {
21     Sticks s(numPhilos); // shared resource
22
23     Philosopher p[numPhilos] =
24     { // pid, thinkDelay, eatDelay, Sticks&
25         Philosopher(1, 100000, 500000, s),
26         Philosopher(2, 200000, 400000, s),
27         Philosopher(3, 300000, 300000, s),
28         Philosopher(4, 400000, 200000, s),
29         Philosopher(5, 500000, 100000, s)
30     };
31
32     for (int i = 0 ; i < numPhilos; ++i)
33     {
34         p[i].live();
35     }
36
37     // wait for all philosophers to leave
38     for (int i = 0 ; i < numPhilos; ++i)
39     {
40         p[i].join();
41     }
42     return 0;

```

4. siehe ./Loesung/PhiloRAII Sie benötigen die Klasse ResourceLock. Ein Objekt dieser Klasse müssen Sie in den beiden Elementfunktionen Sticks::get() und Sticks::put() einsetzen. Der Rest bleibt sich gleich.

```

1  /*
2  * Sticks.h
3  *
4  * Dining Philosophers: Sticks class (Monitor)
5  *
6  * Created on: 07.04.2016
7  * Author: Reto Bonderer
8  */
9  #ifndef STICKS_H_
10 #define STICKS_H_
11
12 class Sticks
13 {
14 public:
15     Sticks(int nr = 5);
16
17     ~Sticks();
18     int getNr() const {return nSticks;}
19     void put(int left, int right); // lay down left and right
        ↳ stick
20     void get(int left, int right); // try to pick left and
        ↳ right stick
21 private:
22     int nSticks;
23     volatile bool* stick; // ptr to dynamic array
24     pthread_mutex_t stickMutex;
25     pthread_cond_t stickCv;
26 };
27 #endif

```

```

1  /*
2  * Sticks.cpp
3  *
4  * Dining Philosophers: Sticks class (Monitor) using RAII
5  *
6  * Created on: 25.03.2020
7  * Author: Reto Bonderer
8  */
9
10 #include <pthread.h>
11 #include "Sticks.h"
12 #include "ResourceLock.h"
13
14 Sticks::Sticks(int nr)
15 : nSticks(nr),
16   stick(new bool[nSticks])
17 {
18     for (int i = 0; i < nSticks; ++i)
19         stick[i] = false;
20
21     // Initialize mutex and condition variable objects
22     pthread_mutex_init(&stickMutex, 0);
23     pthread_cond_init (&stickCv, 0);
24 }
25
26 Sticks::~Sticks()

```

```

27 {
28     delete[] stick;
29     pthread_mutex_destroy(&stickMutex);
30     pthread_cond_destroy(&stickCv);
31 }
32
33 void Sticks::put(int left, int right)
34 {
35     {
36         // start of CS
37         ResourceLock lock(stickMutex);
38         stick[left] = false;
39         stick[right] = false;
40         pthread_cond_signal(&stickCv);
41     }
42     // end of CS
43 }
44
45 void Sticks::get(int left, int right)
46 {
47     {
48         // start of CS
49         ResourceLock lock(stickMutex);
50         while (stick[left] || stick[right])
51         {
52             pthread_cond_wait(&stickCv, &stickMutex);
53         }
54         stick[left] = true;
55         stick[right] = true;
56     }
57     // end of CS
58 }

```

```

1  /*
2  * Philosopher.h
3  *
4  * Dining Philosophers: Philosopher class
5  *
6  * Created on: 20.12.2016
7  * Author: Reto Bonderer
8  */
9  #ifndef PHILOSOPHER_H_
10 #define PHILOSOPHER_H_
11 #include <pthread.h>
12 #include "Sticks.h"
13
14 class Philosopher
15 {
16 public:
17     Philosopher(int pid, int thinkdelay, int eatDelay, Sticks&
18         ↪ s);
19     ~Philosopher();
20     void live(); // the philosopher's life
21     void join(); // wait for philosopher to leave
22 private:
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
```

```

15 using namespace std;
16
17 enum {numPhilos = 5};
18
19 int main()
20 {
21     Sticks s(numPhilos); // shared resource
22     Philosopher p[numPhilos] =
23     { // pid, thinkDelay, eatDelay, Sticks&
24         Philosopher(1, 100000, 500000, s),
25         Philosopher(2, 200000, 400000, s),
26         Philosopher(3, 300000, 300000, s),
27         Philosopher(4, 400000, 200000, s),
28         Philosopher(5, 500000, 100000, s)

```

```

29     };
30
31     for (int i = 0 ; i < numPhilos; ++i)
32     {
33         p[i].live();
34     }
35
36     // wait for all philosophers to leave
37     for (int i = 0 ; i < numPhilos; ++i)
38     {
39         p[i].join();
40     }
41     return 0;
42 }

```

```

1  /*
2  * ResourceLock.h
3  * RAII implementation of lock/unlock mutex
4  *
5  * Created on: Apr 08, 2019
6  * Author: reto.bonderer
7  */
8
9 #ifndef RESOURCELOCK_H_
10 #define RESOURCELOCK_H_
11 #include <pthread.h>
12 class ResourceLock
13 {
14 public:

```

```

15     ResourceLock(pthread_mutex_t& m) :
16         mutex(m)
17     {
18         pthread_mutex_lock(&mutex);
19     }
20     ~ResourceLock()
21     {
22         pthread_mutex_unlock(&mutex);
23     }
24 private:
25     pthread_mutex_t& mutex; // mutex of shared resource
26 };
27
28 #endif /* RESOURCELOCK_H_ */

```

6 Lab 6 CRC - Berechnung und -Implementation in C++

6.1 Aufgabe 1: CRC-8 Berechnungsbeispiel (Papierübung)

Berechnen Sie die Checksumme für den unten abgebildeten Bytestream. Verwenden Sie dafür das Generatorpolynom $G = x^8 + x^2 + x + 1$ (entspricht CRC-8 CCITT).

Message : 00101101 = 0x2D

6.1.1 Lösung

Message: 0 0 1 0 1 1 0 1 = 0x2D
Polynomial: 1 0 0 0 0 0 1 1 1 = 0x107
CRC: **1 1 0 0 0 0 1 1 = 0xC3**

Berechnung:

| | |
|--|---------------------|
| 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 | = CRC |
| 1 0 0 0 0 0 1 1 1 | = XOR Polynomial |
| ----- | |
| 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 | = CRC |
| 1 0 0 0 0 0 1 1 1 | = XOR Polynomial |
| ----- | |
| 0 1 0 1 1 1 0 1 1 0 0 0 0 | = CRC |
| 1 0 0 0 0 0 1 1 1 | = XOR Polynomial |
| ----- | |
| 0 0 1 1 1 0 0 0 1 0 0 | = CRC |
| 1 0 0 0 0 0 1 1 1 | = XOR Polynomial |
| ----- | |
| 0 1 1 0 0 0 0 1 1 | = CRC = 0xC3 |
| ===== | |

6.2 Aufgabe 2: Verifikation empfangener Daten mittels CRC

Nach einer seriellen Übertragung wurden die folgenden drei Messages empfangen. Dabei entspricht das erste Byte den **Daten** und das zweite der **CRC-Prüfsumme** (MSB first). Das verwendete Generatorpolynom ist dasselbe wie in Aufgabe 1 (CRC-8 CCITT).

Überprüfen Sie, welche der drei Messages fehlerfrei übertragen wurden und welche nicht.

6.2.1 Lösung

Message 1: **1 1 0 1 1 0 0 1** 0 0 0 0 0 0 0 1 wurde korrekt empfangen

```
1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 = CRC = 0x01 = message1[1]
```

Bei der noch einfacheren Möglichkeit zur Verifikation werden die gesamten empfangenen Daten inklusive der Prüfbits durch das Generatorpolynom gelassen. Wenn alles korrekt ist, so muss am Schluss der Wert 0 als CRC übrigbleiben

```
1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 = CRC (alle empfangenen Datenbits)
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 = CRC
```

Message 2: **0 1 0 1 0 0 0 1** 0 1 1 1 0 1 1 1 wurde falsch empfangen

```
0 1 0 1 0 0 0 1 0 1 1 1 0 1 1 1 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 = CRC
1 0 0 0 0 0 0 1 1 1 = XOR Polynomial
-----
0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 = CRC != 0
```

Die zweite Message wurde falsch empfangen. Entweder ist das Daten- oder das Prüfsummenbyte korrupt. Der Empfänger muss in einem solchen Fall ein NAK zurückgeben, um die Message noch einmal anzufordern.

6.3 Aufgabe 3: C++ - Implementation

Implementieren Sie eine Klasse Crc, welche die Prüfsumme CRC-8 CCITT mit Schiebeoperationen auf Bit- Ebene berechnet. Um den Algorithmus zu überprüfen, soll ein Testprogramm geschrieben werden, welches entweder die Prüfsumme der Datenbytes aus Aufgabe 2 berechnet, oder die Prüfsumme der vorgegebenen Datei ausgibt. Der Dateiname soll der Applikation als Argument übergeben werden.

6.3.1 Lösung

Wird das Programm ohne Argument aufgerufen, berechnet es die Prüfsumme von 0xd9, 0x51, 0x61.

```
1 $ ./crcTest
2 CRC-8 CCITT Bit um Bit, einfach aber ineffizient
3
4 CRC-8 CCITT der einzelnen Datenbytes:
5 Byte 00: crc(0xd9) = 0x01
6 Byte 01: crc(0x51) = 0xb0
7 Byte 02: crc(0x61) = 0x20
8 CRC-8 CCITT von allen Datenbytes:
9 crc(0xd95161) = 0x2c
```

Mit einem gültigen Dateinamen als Argument wird diese Datei in einen Byte-Buffer gelesen und die Prüfsumme über den gesamten Buffer berechnet.

```
1 $ ./crcTest datafile.txt
2 CRC-8 CCITT Bit um Bit, einfach aber ineffizient
3 Datei: datafile.txt
4 Anzahl Datenbytes: 72125
5 CRC-8 CCITT: 0x10
```

```
1 /*
2  * Crc.h
3  *
4  * CRC 8 bit simple implementation
5  *
6  * Created on: 25.03.2019
7  * Author: Reto Bonderer
8  */
9 #ifndef CRC_H_
10 #define CRC_H_
11
12 #include <stdint.h>
13
14 typedef uint8_t CrcType;
15
16 class Crc
17 {
18 public:
19     Crc(CrcType thePoly = 0x07);
20     CrcType getCrc(const uint8_t message[], unsigned int
21         ↳ nBytes) const;
22 private:
23     CrcType poly;
24 };
25 #endif // CRC_H_
```

```
1 /*
2  * Crc.cpp
3  *
4  * Created on: 25.03.2019
5  * Author: Reto Bonderer
6  */
7
8 #include "Crc.h"
9
10 enum {crcWidth = 8 * sizeof(CrcType),
11     crcTopBit = 1 << (crcWidth - 1)};
12
13 Crc::Crc(CrcType thePoly) :
14     poly(thePoly)
15 {
16 }
17
18 CrcType Crc::getCrc(const uint8_t message[], unsigned int
19     ↳ nBytes) const
20 {
21     CrcType remainder = 0;
22
23     // modulo-2 division; byte by byte
24     for (unsigned int byte = 0; byte < nBytes; ++byte)
25     {
26         // XOR the next byte to the remainder
27         remainder ^= (message[byte] << (crcWidth - 8));
28
29         // modulo-2 division; bit by bit
30         for (int bit = 8; bit > 0; --bit)
31         {
32             if (remainder & crcTopBit)
33             {
34                 remainder = (remainder << 1) ^ poly;
35             }
36             else
37             {
38                 remainder = (remainder << 1);
39             }
40         }
41     }
42     return remainder;
43 }
```

```
1 /*
2  * crcTest.h
3  *
4  * Created on: 25.03.2019
5  * Author: Reto Bonderer
6  */
7
8 #include <iostream>
9 #include <fstream>
10 #include <iomanip>
11 #include "Crc.h"
12
13 using namespace std;
14
15 int main(int argc, char* argv[])
16 {
17     // Predefined data will be used if no file name has been
18     ↳ passed.
19     const uint8_t data[] = {0xd9, 0x51, 0x61};
20
21     Crc crc(0x07); // use CRC-8 CCITT
22     CrcType crcValue;
23
24     cout << "CRC-8 CCITT Bit um Bit, einfach aber ineffizient"
25         ↳ << endl << endl;
26
27     if (argc > 1)
28     {
29         // compute CRC of the passed file (argv[1])
30         ifstream f;
31         uint8_t* buf;
32         unsigned int len;
33
34         try
35         {
36             f.open(argv[1], ios::in | ios::binary);
37
38             // determine file size
```

```

37     f.seekg(0, ios::end);
38     len = f.tellg();
39     f.seekg(0, ios::beg);
40
41     // read file content into byte buffer
42     buf = new uint8_t[len];
43     f.read((char*)buf, len);
44     f.close();
45
46     cout << "Datei: " << argv[1] << endl;
47     cout << "Anzahl Datenbytes: " << len << endl;
48
49     // calculate CRC of the whole byte buffer
50     crcValue = crc.getCrc(buf, len);
51
52     cout << "CRC-8 CCITT: " << setfill('0') << hex <<
53         << showbase << internal
54         << setw(4) << (int)crcValue << endl;
55
56     // free allocated memory
57     delete[] buf;
58 }
59 catch (const exception& ex)
60 {
61     cerr << "Unable to read file '" << argv[1] << "'!" <<
62         << endl;
63 }
64 else
65 {
66     cout << "CRC-8 CCITT der einzelnen Datenbytes:" << endl;
67     for (unsigned int i = 0; i < sizeof(data)/sizeof(data[0]);
68         ++i)
69     {
70         // compute CRC of the individual data bytes
71         crcValue = crc.getCrc(&data[i], 1);
72
73         cout << "Byte " << dec << setfill('0') << setw(2) << i
74             << ": crc(" << setfill('0') << showbase << hex << internal << setw
75                 << (4) << (int)data[i]
76                 << ") = " << setw(4) << (int)crcValue << endl;
77     }
78
79     // compute CRC of all data bytes
80     crcValue = crc.getCrc(data, sizeof(data)/sizeof(data[0]));
81
82     cout << "CRC-8 CCITT von allen Datenbytes:" << endl;
83     cout << "crc(" << setfill('0') << showbase << hex <<
84         << internal << setw(4);
85     for (unsigned int i = 0; i < sizeof(data)/sizeof(data[0]);
86         ++i)
87     {
88         cout << (int)data[i] << noshowbase << setw(2);
89     }
90     cout << ") = " << showbase << setw(4) << (int)crcValue <<
91         << endl << endl;
92 }
93 return 0;

```

6.4 Aufgabe 4: C++ - Implementation mit einer Lookup Tabelle

Implementieren Sie eine Klasse Crc, welche die Prüfsumme CRC-8 CCITT mit Hilfe einer 8 Bit breiten Lookup Tabelle berechnet. Die Lookup Tabelle kann im Voraus berechnet werden oder sogar konstant im ROM abgelegt werden. Um den Algorithmus zu überprüfen soll ein Testprogramm geschrieben werden, welches entweder die Prüfsumme der Datenbytes aus Aufgabe 2 berechnet, oder die Prüfsumme der vorgegebenen Datei ausgibt. Der Dateiname soll der Applikation als Argument übergeben werden. Variieren Sie die Dateigröße und vergleichen Sie die Berechnungszeit mit der Variante aus der Aufgabe 3.

Hinweis File I/=:

```

1  #include <fstream>
2  ...
3  f.open(argv[1], ios::in | ios::binary);
4  // determine file size
5  f.seekg(0, ios::end);
6  len = f.tellg();
7  f.seekg(0, ios::beg);
8  // Allocate byte buffer
9  buf = new uint8_t[len];
10 // read file content into byte buffer
11 f.read((char*)buf, len);
12 f.close();
13 // ...
14 delete[] buf; // free allocated memory

```

6.4.1 Lösung

```

1  /*
2   * Crc.h
3   *
4   * CRC 8 bit lookup table implementation
5   *
6   * Created on: 25.03.2019
7   * Author: Reto Bonderer
8   */
9  #ifndef CRC_H_
10 #define CRC_H_
11
12 #include <stdint.h>
13
14 typedef uint8_t CrcType;
15
16 class Crc
17 {
18 public:
19     Crc(CrcType thePoly = 0x07);
20     CrcType getCrc(const uint8_t message[], unsigned int
21         << nBytes) const;
22 private:
23     CrcType poly;
24     CrcType crcTable[256];
25     void computeTable();
26 };
27 #endif // CRC_H_

```

```

1  /*
2  * crc.cpp
3  *
4  * Created on: 31.03.2016
5  * Author: Reto Bonderer
6  */
7
8  #include <iostream>
9  #include <iomanip>
10 #include "Crc.h"
11 using namespace std;
12
13 enum {crcWidth = 8 * sizeof(CrcType),
14       crcTopBit = 1 << (crcWidth - 1)};
15
16 Crc::Crc(CrcType thePoly) :
17     poly(thePoly)
18 {
19     computeTable();
20 }
21
22 void Crc::computeTable()
23 {
24     CrcType remainder = 0;
25
26     for (unsigned int i = 0; i < 256; ++i)
27     {
28         remainder = i << (crcWidth - 8);
29         for (unsigned int bit = 8; bit > 0; --bit)

```

```

30     {
31         if (remainder & crcTopBit)
32         {
33             remainder = (remainder << 1) ^ poly;
34         }
35         else
36         {
37             remainder <<= 1;
38         }
39     }
40     crcTable[i] = remainder;
41 }
42
43
44 CrcType Crc::getCrc(const uint8_t message[], unsigned int
45     ↳ nBytes) const
46 {
47     CrcType remainder = 0;
48     uint8_t data;
49
50     for (unsigned int byte = 0; byte < nBytes; ++byte)
51     {
52         data = message[byte] ^ (remainder >> (crcWidth - 8));
53         remainder = crcTable[data] ^ (remainder << 8);
54     }
55     return remainder;
56 }

```

```

1  /*
2  * crcTest.h
3  *
4  * Created on: 31.03.2016
5  * Author: Reto Bonderer
6  */
7
8  #include <iostream>
9  #include <fstream>
10 #include <iomanip>
11 #include "Crc.h"
12
13 using namespace std;
14
15 int main(int argc, char* argv[])
16 {
17     // Predefined data will be used if no file name has been
18     ↳ passed.
19     const uint8_t data[] = {0xd9, 0x51, 0x61};
20
21     Crc crc(0x07); // use CRC-8 CCITT
22     CrcType crcValue;
23
24     cout << "CRC-8 CCITT mit einer Byte-orientierten Lookup
25     ↳ Tabelle" << endl << endl;
26
27     if (argc > 1)
28     {
29         // compute CRC of the passed file (argv[1])
30         ifstream f;
31         uint8_t* buf;
32         unsigned int len;
33
34         try
35         {
36             f.open(argv[1], ios::in | ios::binary);
37
38             // determine file size
39             f.seekg(0, ios::end);
40             len = f.tellg();
41             f.seekg(0, ios::beg);
42
43             // read file content into byte buffer
44             buf = new uint8_t[len];
45             f.read((char*)buf, len);
46             f.close();
47
48             cout << "Datei: " << argv[1] << endl;
49             cout << "Anzahl Datenbytes: " << len << endl;

```

```

48
49     // calculate CRC of the whole byte buffer
50     crcValue = crc.getCrc(buf, len);
51
52     cout << "CRC-8 CCITT: " << setfill('0') << hex <<
53     ↳ showbase << internal
54     ↳ setw(4) << (int)crcValue << endl;
55
56     // free allocated memory
57     delete[] buf;
58 }
59 catch (const exception& ex)
60 {
61     cerr << "Unable to read file '" << argv[1] << "'!" <<
62     ↳ endl;
63 }
64
65 else
66 {
67     cout << "CRC-8 CCITT der einzelnen Datenbytes:" << endl;
68
69     for (unsigned int i = 0; i < sizeof(data)/sizeof(data[0]);
70     ↳ ++i)
71     {
72         // compute CRC of the individual data bytes
73         crcValue = crc.getCrc(&data[i], 1);
74
75         cout << "Byte " << dec << setfill('0') << setw(2) << i
76         ↳ << ": crc(" << showbase << hex << internal << setw
77         ↳ (4) << (int)data[i]
78         ↳ << ") = " << setw(4) << (int)crcValue << endl;
79     }
80
81     // compute CRC of all data bytes
82     crcValue = crc.getCrc(data, sizeof(data)/sizeof(data[0]));
83
84     cout << "CRC-8 CCITT von allen Datenbytes:" << endl;
85     cout << "crc(" << setfill('0') << showbase << hex <<
86     ↳ internal << setw(4);
87     for (unsigned int i = 0; i < sizeof(data)/sizeof(data[0]);
88     ↳ ++i)
89     {
90         cout << (int)data[i] << noshowbase << setw(2);
91     }
92     cout << ") = " << showbase << setw(4) << (int)crcValue <<
93     ↳ endl << endl;
94 }
95 return 0;
96 }

```

6.5 Aufgabe 5: C++ - Implementation mit einer Lookup Tabelle

Mit dem Tool Callgrind kann die Laufzeit eines Programms analysiert werden. In der Standardkonfiguration werden die Anzahl der ausgeführten Instruktionen aufgezeichnet. Die aufgezeichnete Anzahl der Instruktionen wird zudem

mit Sourcecode-Zeilen und Funktionsaufrufen in Beziehung gesetzt. Zusätzlich kann eine Cachesimulation aktiviert werden, die weitere Informationen über die Laufzeitperformance des Programms geben kann. Die Cachesimulation führt das gleiche aus wie das Tool Cachegrind. Callgrind schreibt das Ergebnis der Laufzeitanalyse in eine Datei. Die Datei kann dann mit dem Programm KCachegrind visualisiert werden.

<http://valgrind.org/docs/manual/cl-manual.html> <http://valgrind.org/docs/manual/cg-manual.html>

In dieser Aufgabe sollen Sie die CRC-8 CCITT Implementationen mittels Tabelle und mittels Schiebeoperationen auf Bit-Ebene mit Valgrind profilieren.

Beachten Sie die Vorgabe und binden Sie Ihre eigenen CRC-Implementationen ein.

Callgrind: Profilieren Sie die beiden CRC Implementationen mit dem Tool Callgrind. Visualisieren sie die Ausgabe mit KCachegrind und untersuchen sie die Performanceunterschiede.

1. In welchem Sourcecode-File liegt der Hotspot. Auf welchen Sourcecode-Zeilen liegt der Hotspot im entsprechenden File für die CRC-Berechnung? Wie unterscheiden sich die beiden Implementationen bezüglich Laufzeitperformance?
2. Wie verändert sich die Performance der beiden Programme, wenn mit Optimierungsstufe -O3 kompiliert wird? Verändert sich der Hotspot der beiden CRC-8 Implementationen gleichermassen mit eingeschalteter Optimierung?

Hinweis: Übergeben Sie für das Callgrind-Profilieren das datafile.txt dem CRC Testprogramm, damit der CRC über einige Bytes berechnet werden muss und dadurch der Overhead für Systemfunktionen bezogen auf die CRC Berechnung kleiner wird.

6.5.1 Lösung

Die verwendeten Profiling-Befehle finden Sie in den make-Files in ./Loesung/A5.

1. Bei beiden Implementationen benötigt die Funktion `Crc::getCrc()` aus `crc.cpp` am meisten Rechenzeit. In dieser Funktion liegt der Hotspot bei der for-Schleife. Der Unterschied der beiden Implementationen kann an der Anzahl Zyklen festgestellt werden. Bei der Tabellenimplementation werden 1'442'513 Zyklen und bei der Schiebeoperationsvariante 7'571'101 Zyklen verbraucht. Daraus geht hervor, dass die Tabellenvariante etwa 5.2-mal schneller ist als die Schiebeoperationsvariante.

```
1 #
2 # Makefile to profile the CRC program with valgrind
3 # File: makefile
4 # Gian Danuser, 06.04.2016
5 #
6 CC = g++
7 LINK = g++
8 # '-g3' to include debugging information
9 CFLAGS = -c -Wall -pedantic -g3
10 LFLAGS = -pedantic -Wall
11 OBJS = Crc.o crcTest.o
12 EXEC = crcTest
13
14 $(EXEC): $(OBJS)
```

```
15 $(LINK) $(LFLAGS) -o $(@) $(OBJS)
16
17 crcTest.o: crcTest.cpp Crc.h
18 $(CC) $(CFLAGS) crcTest.cpp
19
20 Crc.o: Crc.cpp Crc.h
21 $(CC) $(CFLAGS) Crc.cpp
22
23 clean:
24 rm -f $(EXEC) $(OBJS) callgrind.out.*
25
26 callgrind: $(EXEC)
27 valgrind --tool=callgrind ./$(EXEC) datafile.txt &&
    ↪ kcachegrind
```

```
1 #
2 # Makefile to profile the CRC program with valgrind
3 # File: makefile
4 # Gian Danuser, 06.04.2016
5 #
6 CC = g++
7 LINK = g++
8 # '-g3' to include debugging information
9 CFLAGS = -c -Wall -pedantic -g3 -O3
10 LFLAGS = -pedantic -Wall
11 OBJS = Crc.o crcTest.o
12 EXEC = crcTest
13
14 $(EXEC): $(OBJS)
```

```
15 $(LINK) $(LFLAGS) -o $(@) $(OBJS)
16
17 crcTest.o: crcTest.cpp Crc.h
18 $(CC) $(CFLAGS) crcTest.cpp
19
20 Crc.o: Crc.cpp Crc.h
21 $(CC) $(CFLAGS) Crc.cpp
22
23 clean:
24 rm -f $(EXEC) $(OBJS) callgrind.out.*
25
26 callgrind: $(EXEC)
27 valgrind --tool=callgrind ./$(EXEC) datafile.txt &&
    ↪ kcachegrind
```

2. Die Schiebeoperationsvariante reduziert die Anzahl der Zyklen um 70 % auf 2'235'591 Zyklen und die Tabellenvariante ebenfalls um 70 % auf 432'757 Zyklen.

```

1  #
2  # Makefile to profile the CRC program with valgrind
3  # File: makefile
4  # Gian Danuser, 06.04.2016
5  #
6  CC = g++
7  LINK = g++
8  # '-g3' to include debugging information
9  CFLAGS = -c -Wall -pedantic -g3
10 LFLAGS = -pedantic -Wall
11 OBJS = Crc.o crcTest.o
12 EXEC = crcTest
13
14 $(EXEC): $(OBJS)

```

```

15 $(LINK) $(LFLAGS) -o $(@) $(OBJS)
16
17 crcTest.o: crcTest.cpp Crc.h
18 $(CC) $(CFLAGS) crcTest.cpp
19
20 Crc.o: Crc.cpp Crc.h
21 $(CC) $(CFLAGS) Crc.cpp
22
23 clean:
24 rm -f $(EXEC) $(OBJS) callgrind.out.*
25
26 callgrind: $(EXEC)
27 valgrind --tool=callgrind ./$$(EXEC) datafile.txt &&
    ↪ kcachegrind

```

```

1  #
2  # Makefile to profile the CRC program with valgrind
3  # File: makefile
4  # Gian Danuser, 06.04.2016
5  #
6  CC = g++
7  LINK = g++
8  # '-g3' to include debugging information
9  CFLAGS = -c -Wall -pedantic -g3 -O3
10 LFLAGS = -pedantic -Wall
11 OBJS = Crc.o crcTest.o
12 EXEC = crcTest
13
14 $(EXEC): $(OBJS)

```

```

15 $(LINK) $(LFLAGS) -o $(@) $(OBJS)
16
17 crcTest.o: crcTest.cpp Crc.h
18 $(CC) $(CFLAGS) crcTest.cpp
19
20 Crc.o: Crc.cpp Crc.h
21 $(CC) $(CFLAGS) Crc.cpp
22
23 clean:
24 rm -f $(EXEC) $(OBJS) callgrind.out.*
25
26 callgrind: $(EXEC)
27 valgrind --tool=callgrind ./$$(EXEC) datafile.txt &&
    ↪ kcachegrind

```

7 Lab 7 inline und Bitfelder

Allgemeine Bemerkungen zu Effizienz- und Performancebetrachtungen Die Effizienz- und Performancebetrachtungen sind stark von der Qualität des Compilers abhängig. Die aktuelle Version des GNU-Compilers (Version 7.4.0) erzeugt sehr effizienten Code, da die Link-Time Optimization eingeführt wurde. Im Gegensatz zur Compile-Time Optimization wird der aus den einzelnen Objectfiles gelinkte Programmcode als Ganzes analysiert und optimiert. Die folgenden Optionen der GNU-Compiler könnten nützlich sein:

- E Precompile only, der Output wird auf stdout geschrieben
- S Assembleroutput, ohne Objectfile erzeugen
- c nur compilieren
- O0 keine Optimierung
- O1 Optimierungsstufe 1 (siehe g++ -help für Details)
- O2 Optimierungsstufe 2
- O3 Optimierungsstufe 3
- Os Optimierung auf Codegrösse

Hinweise zum x86-Instruktionssatz finden Sie unter anderem im Manual 64-ia-32-architectures-software-developer-manual-325462.pdf und auf folgenden Websites:

http://en.wikipedia.org/wiki/X86_instruction_listings

<http://www.cs.uaf.edu/2005/fall/cs301/support/x86/index.html>

<http://ref.x86asm.net/>

7.1 Aufgabe 1: inline-Methoden

Bei sehr kurzen Methoden (Einzeiler) ist der Overhead eines Funktionsaufrufs recht gross. Durch Inlining kann der Funktionsaufruf vermieden werden, der Code (Funktionsrumpf) wird vom Compiler direkt anstelle des Funktionsaufrufs gesetzt. Der Compiler wird üblicherweise keinen Inlinecode erzeugen, falls die Funktion rekursiv aufgerufen wird oder falls ein Pointer auf diese Funktion verwendet wird. Eine weitere Schwierigkeit ergibt sich, wenn die Inlinefunktionen in mehreren Sourcefiles verwendet werden sollen, der Linker kann aus einer bestehenden Objektdatei kaum Inlinecode erzeugen.

Für diese Aufgabe sind zusätzlich die folgenden Optionen der GNU-Compiler nützlich:

-Winline Erzeugt eine Warnung, falls von einer mit inline spezifizierten Funktion kein Inlinecode erzeugt werden konnte. Verwenden Sie für diese Untersuchungen die Linux-Konsole im Ubuntu-Image, Eclipse bietet keine Vorteile.

1. Wenn bei Klassendeklarationen der Code direkt definiert wird, sind diese Funktionen implizit inline. Verwenden Sie den im Verzeichnis ./Vorgabe/Rectangle zur Verfügung gestellte C++-Code. Wie Sie feststellen, ist die Methode getArea() nicht inline. Compilieren Sie dieses Programm, ein Makefile steht zur Verfügung.
2. Betrachten Sie die erzeugten Assemblerfiles. Welche Methoden sind inline, welche nicht? Hinweis: betrachten Sie dazu die call-Befehle.
3. Wenn Sie die einzelnen *.s-Dateien betrachten, können Sie nicht feststellen, ob der Linker allenfalls auch noch optimieren kann. Relevant ist einzig, wie der Code in der Programmdatei aussieht. Verwenden Sie den Debugger gdb von der Kommandozeile, um dies festzustellen. Starten Sie die Debugsession mit dem Befehl gdb ./main. Anschliessend müssen Sie das Programm starten mit start. Den disassemblierten Code sehen Sie, wenn Sie disassem eintippen. Im Folgenden sehen Sie den Ausschnitt einer Debugsession.

```
1  gdb ./main
2  GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
3  ...
4  (gdb) start
5  Temporary breakpoint 1 at 0x66e
6  ...
7  Temporary breakpoint 1, 0x000055555555466e in main ()
8  (gdb) disassem
9  Dump of assembler code for function main:
10      0x000055555555466a <+0>: push    %rbp
11      0x000055555555466b <+1>: mov     %rsp,%rbp
12      => 0x000055555555466e <+4>: sub     $0x40,%rsp
13      ...
```

4. Wahrscheinlich haben Sie festgestellt, dass alle Methoden mit einem Call aufgerufen werden. Ändern Sie die Optimierungsstufe bis alle Methoden ausser `getArea()` inline sind.
5. Sie möchten nun sowohl die impliziten Inlinefunktionen ins cpp-File zügeln, als auch von `getArea()` In-linecode erhalten. Sie müssen die Funktionen mit `inline` kennzeichnen. Häufig werden die Implementa-tionen der Inlinefunktionen direkt unter die Klassendeklaration verschoben.
6. Testen Sie, ob alles richtig funktioniert, indem Sie ein Projekt mit mehreren cpp-Files erstellen, welche die Inlinefunktionen verwenden, d.h. `main.cpp` plus ein weiteres File. Betrachten Sie die Assemblerfiles, es sollten keine Calls auf Memberfunktionen mehr geben.

7.1.1 Lösung

1. just do it
2. Bei Optimierungsstufe 0 sind keine Funktionen inline (siehe `./Loesung/A1-b`). Man erkennt das daran, dass bei den einzelnen Funktionen Labels definiert sind und mit einem Return (`ret`) abgeschlossen werden. Die Funktionen werden mit einem `call`-Befehl aufgerufen (siehe folgenden Ausschnitt)

```

        .file                "main.cpp"
* ...
main:
* ...
        call                _ZN9RectangleC1Edd
        leaq                -32(%rbp), %rax
        movq                %rax, %rdi
        call                _ZNK9Rectangle4getAEv
        movq                %xmm0, %rax
        movq                %rax, -48(%rbp)
        leaq                -32(%rbp), %rax
        movq                %rax, %rdi
        call                _ZNK9Rectangle7getAreaEv
* ...
_ZNK9Rectangle4getAEv:
.LFB3:
        .cfi_startproc
        pushq                %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq                %rsp, %rbp
        .cfi_def_cfa_register 6
        movq                %rdi, -8(%rbp)
        movq                -8(%rbp), %rax
        movsd                (%rax), %xmm0
        popq                %rbp
        .cfi_def_cfa 7, 8
        ret
* ...

```

3. Der Debugger-Output zeigt, dass bei Optimierungsstufe 0 alle Funktionen mit `call` aufgerufen werden.


```
(gdb) disassem
Dump of assembler code for function main:
0x000055555555466a <+0>:      push    %rbp
0x000055555555466b <+1>:      mov     %rsp,%rbp
=> 0x000055555555466e <+4>:      sub     $0x40,%rsp
0x0000555555554672 <+8>:      mov     %fs:0x28,%rax
0x000055555555467b <+17>:     mov     %rax,-0x8(%rbp)
0x000055555555467f <+21>:     xor     %eax,%eax
0x0000555555554681 <+23>:     movsd   0x14f(%rip),%xmm0      # 0x5555555547d8
0x0000555555554689 <+31>:     mov     0x150(%rip),%rdx      # 0x5555555547e0
0x0000555555554690 <+38>:     lea     -0x20(%rbp),%rax
0x0000555555554694 <+42>:     movapd %xmm0,%xmm1
0x0000555555554698 <+46>:     mov     %rdx,-0x38(%rbp)
0x000055555555469c <+50>:     movsd   -0x38(%rbp),%xmm0
0x00005555555546a1 <+55>:     mov     %rax,%rdi
0x00005555555546a4 <+58>:     callq   0x5555555546ee <_ZN9RectangleC2Edd>
0x00005555555546a9 <+63>:     lea     -0x20(%rbp),%rax
0x00005555555546ad <+67>:     mov     %rax,%rdi
0x00005555555546b0 <+70>:     callq   0x55555555471e <_ZNK9Rectangle4getAEv>
0x00005555555546b5 <+75>:     movq    %xmm0,%rax
0x00005555555546ba <+80>:     mov     %rax,-0x30(%rbp)
0x00005555555546be <+84>:     lea     -0x20(%rbp),%rax
0x00005555555546c2 <+88>:     mov     %rax,%rdi
0x00005555555546c5 <+91>:     callq   0x555555554730 <_ZNK9Rectangle7getAreaEv>
0x00005555555546ca <+96>:     movq    %xmm0,%rax
0x00005555555546cf <+101>:    mov     %rax,-0x28(%rbp)
0x00005555555546d3 <+105>:    mov     $0x0,%eax
0x00005555555546d8 <+110>:    mov     -0x8(%rbp),%rcx
0x00005555555546dc <+114>:    xor     %fs:0x28,%rcx
0x00005555555546e5 <+123>:    je      0x5555555546ec <main+130>
0x00005555555546e7 <+125>:    callq   0x555555554540 <__stack_chk_fail@plt>
0x00005555555546ec <+130>:    leaveq
0x00005555555546ed <+131>:    retq
```

4. Ab Optimierungsstufe 1 sind die impliziten inline-Funktionen alle inline, auch der Konstruktor (siehe Zeile 0x00005555555546a4 <+58>: in Aufgabe c). Einzig die Funktion `getArea()` wird immer noch aufgerufen, da diese Funktion in einer anderen Objectdatei liegt (siehe `./Loesung/A1-d/*.s`). Der Assemblerdump zeigt das:

```
(gdb) disassem
Dump of assembler code for function main:
=> 0x000055555555466a <+0>:      sub     $0x28,%rsp
0x000055555555466e <+4>:      mov     %fs:0x28,%rax
0x0000555555554677 <+13>:     mov     %rax,0x18(%rsp)
0x000055555555467c <+18>:     xor     %eax,%eax
0x000055555555467e <+20>:     mov     0xd3(%rip),%rax      # 0x555555554758
0x0000555555554685 <+27>:     mov     %rax,(%rsp)
0x0000555555554689 <+31>:     mov     0xd0(%rip),%rax      # 0x555555554760
0x0000555555554690 <+38>:     mov     %rax,0x8(%rsp)
0x0000555555554695 <+43>:     mov     %rsp,%rdi
0x0000555555554698 <+46>:     callq   0x5555555546bc <_ZNK9Rectangle7getAreaEv>
0x000055555555469d <+51>:     mov     0x18(%rsp),%rdx
0x00005555555546a2 <+56>:     xor     %fs:0x28,%rdx
0x00005555555546ab <+65>:     jne     0x5555555546b7 <main+77>
0x00005555555546ad <+67>:     mov     $0x0,%eax
0x00005555555546b2 <+72>:     add     $0x28,%rsp
0x00005555555546b6 <+76>:     retq
0x00005555555546b7 <+77>:     callq   0x555555554540 <__stack_chk_fail@plt>
```

5. Wenn das bestehende `main()` genommen wird, so optimiert der Compiler ab Optimierungsstufe 1 alle Variablen weg, da sie nicht verwendet werden (siehe `./Loesung/A1-e1/main.s`).

```
(gdb) disassem
Dump of assembler code for function main:
=> 0x00005555555545fa <+0>:      mov     $0x0,%eax
    0x00005555555545ff <+5>:      retq
```

In der beiliegenden Lösung (siehe ./Loesung/A1-e2) werden die Daten deshalb auf cout geschrieben oder die Variablen als volatile deklariert, d.h. die Berechnungen können nicht wegoptimiert werden. Die Methoden können bei der Deklaration, bei der Definition oder bei beiden Stellen mit inline gekennzeichnet werden.

6. siehe ./Loesung/A1-f Eine neue Funktion `getRectangleB(const Rectangle& r)` wurde im File `rectangleB.cpp` eingeführt. Die Funktion verwendet inline Methoden der `Rectangle`-Klasse um `b` zu berechnen. Im gdb-Dump kann nachgeprüft werden das im main nur ein einziger Call vorkommt (`getRectangleB()`):

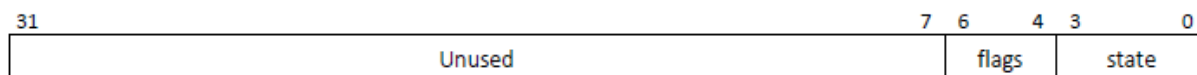
```
(gdb) disassem
Dump of assembler code for function main:
=> 0x0000555555554560 <+0>:      sub     $0x38,%rsp
    0x0000555555554564 <+4>:      movapd 0x214(%rip),%xmm0      # 0x555555554780
    0x000055555555456c <+12>:     lea     0x10(%rsp),%rdi
    0x0000555555554571 <+17>:     mov     %fs:0x28,%rax
    0x000055555555457a <+26>:     mov     %rax,0x28(%rsp)
    0x000055555555457f <+31>:     xor     %eax,%eax
    0x0000555555554581 <+33>:     movaps  %xmm0,0x10(%rsp)
    0x0000555555554586 <+38>:     movsd  0x202(%rip),%xmm0      # 0x555555554790
    0x000055555555458e <+46>:     movsd  %xmm0,0x8(%rsp)
    0x0000555555554594 <+52>:     callq  0x5555555546d0 <_Z13getRectangleBRK9Rectangle>
    0x0000555555554599 <+57>:     movsd  %xmm0,0x8(%rsp)
    0x000055555555459f <+63>:     mov     0x28(%rsp),%rdx
    0x00005555555545a4 <+68>:     xor     %fs:0x28,%rdx
    0x00005555555545ad <+77>:     jne     0x5555555545b6 <main+86>
    0x00005555555545af <+79>:     xor     %eax,%eax
    0x00005555555545b1 <+81>:     add     $0x38,%rsp
    0x00005555555545b5 <+85>:     retq
```

Zudem darf in der Funktion `getRectangleB()` kein Call vorkommen, da die verwendeten Methoden inline sein müssen:

```
(gdb) disass _Z13getRectangleBRK9Rectangle
Dump of assembler code for function _Z13getRectangleBRK9Rectangle:
    0x00000000004005c0 <+0>:      movsd  (%rdi),%xmm1
    0x00000000004005c4 <+4>:      movsd  0x8(%rdi),%xmm0
    0x00000000004005c9 <+9>:      mulsd  %xmm1,%xmm0
    0x00000000004005cd <+13>:     divsd  %xmm1,%xmm0
    0x00000000004005d1 <+17>:     retq
```

7.2 Aufgabe 2: Bitfelder

Gegeben ist das nachfolgende Register



- Definieren Sie dieses Register mit Hilfe eines Bitfelds. Die Variable (z.B. `r`) müssen Sie mit volatile kennzeichnen. Wieso?
- Setzen Sie nun `state` auf den Wert 15 und `flags` auf den Wert 3. Untersuchen Sie, wie der Assembler-code aussieht.

3. Inkrementieren Sie nun flags mittels `++r.flags`. Achten Sie nun auf den Code.
4. Führen Sie mit flags eine übliche Bitoperation durch, z.B. `r.flags &= 6`. Achten Sie nun auf den Code.
5. Diskutieren Sie die erhaltenen Ergebnisse. Entspricht das Resultat Ihren Erwartungen? Worüber sind Sie überrascht? Sollen Bitfelder so verwendet werden? Sollen sie überhaupt verwendet werden?

7.2.1 Lösung

1. volatile muss verwendet werden, weil die Variable auf ein Hardwareregister gemappt wird und nicht wegoptimiert werden darf. Zudem können die Werte von Registern ebenfalls durch die Hardware geändert werden. Deshalb muss der Code so generiert werden, dass das Register jedesmal zuerst gelesen wird bevor damit gearbeitet wird. Wenn dies nicht gemacht würde, könnte passieren, dass mit einer Ko-pie des Registerinhalts gearbeitet wird, der in der Zwischenzeit durch die Hardware geändert wurde.

```

1  .file      "main.cpp"
2  .text
3  .section   .text.startup,"ax",@progbits
4  .p2align  4,,15
5  .globl    main
6  .type     main,@function
7  main:
8  .LFB0:
9      .cfi_startproc
10     movl   $0, -4(%rsp)
11     movl   -4(%rsp), %eax
12     orl    $15, %eax
13     movl   %eax, -4(%rsp)
14     movl   -4(%rsp), %eax
15     andl   $-113, %eax
16     orl    $48, %eax
17     movl   %eax, -4(%rsp)
18     movl   -4(%rsp), %eax
19     movl   -4(%rsp), %edx
20     shrl   $4, %eax
21     addl   $1, %eax
22     andl   $-113, %edx
23     andl   $7, %eax
24     sall   $4, %eax
25     orl    %edx, %eax
26     movl   %eax, -4(%rsp)
27     movl   -4(%rsp), %eax
28     andl   $96, %eax
29     movl   %eax, %edx
30     movl   -4(%rsp), %eax
31     andl   $-113, %eax
32     orl    %edx, %eax
33     movl   %eax, -4(%rsp)
34     xorl   %eax, %eax
35     ret
36     .cfi_endproc
37 .LFE0:
38     .size   main,.-main
39     .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
40     .section .note.GNU-stack,"",@progbits

```

2. Die Registervariable wird auf `-4(%ebp)` gelegt, d.h. auf den Stack. Aus dem Zugriff auf state wird direkt eine OR-Operation (`orl`). Da die Variable volatile ist, werden die Werte laufend zwischen dem Stack und dem Register `eax` hin- und hergeschoben. Der Wert auf dem Stack muss immer der richtige und aktuelle sein.

```

1  movl   $0, -4(%rsp)
2  ...
3  movl   -4(%rsp), %eax
4  orl    $15, %eax
5  movl   %eax, -4(%rsp)

```

Das direkte Setzen von flags wird ebenfalls in eine OR-Operation umgewandelt, wobei die flags-Bits zuerst ausmaskiert werden (`andl $-113,% eax`).

```

1  movl   -4(%rsp), %eax
2  andl   $-113, %eax
3  orl    $48, %eax
4  movl   %eax, -4(%rsp)

```

3. Beim Inkrementieren von flags wird mit Schiebeoperationen (zuerst nach rechts, dann nach links) und den Registern A und D gearbeitet. Dieser Zugriff wird sehr ineffizient.

```

1  movl   -4(%rsp), %eax
2  movl   -4(%rsp), %edx
3  shrl   $4, %eax
4  addl   $1, %eax
5  andl   $-113, %edx
6  andl   $7, %eax
7  sall   $4, %eax
8  orl    %edx, %eax
9  movl   %eax, -4(%rsp)

```

Pseudocode:

- (a) Wert von r in die Register A und D kopieren
- (b) Register D um 4 Bits nach rechts schieben, um eins Inkrementieren, mit 7 ausmaskieren und wieder um 4 Bits nach links schieben

- (c) Flags-Bits in Register A ausmaskieren und mit Register D verodern
- (d) Register A in r speichern

Hinweis für den Registerzugriff (Beispiel Register D):

```

1 |63..32|31..16|15-8|7-0|
2 |DH.|DL.| <- DH / DL adressieren das High- / Low-Byte der untern 16
3 |      |      |      | Bits von Register D
4 |DX.....| <- adressiert die untern 16 Bits von Register D
5 |EDX.....| <- EDX adressiert die untern 32 Bits von Register
6 |RDX.....| <- RDX adressiert alle 64 Bits von Register D

```

4. Auch die Operation `r.flags &= 6` wird sehr ineffizient durchgeführt, da das Muster zuerst an Bitposition 0 geschoben wird, dann wird die Operation durchgeführt und abschliessend wieder zurückgeschoben.
5. Bitfelder sind bekanntlich nicht standardisiert. Wenn die einzelnen Felder direkt gesetzt werden, dann erzeugt der GNU-Compiler effizienten Code, er nimmt direkt eine Bitoperation. Wenn hingegen Operationen durchgeführt werden wie `r.flags &= 6`, dann wird sehr ineffizienter Code generiert. Die meisten anderen Compiler können das auch nicht besser. Bitfelder sollten deshalb aus meiner Sicht nicht verwendet werden, da nebst der nicht vorhandenen Portabilität zudem sehr ineffizienter Code entsteht.

7.3 Aufgabe 3: Bitmasken

1. Lösen Sie die Aufgabe 3 mit allen Operationen ausser der Addition direkt mittels (inline-) Operationen mit Bitmasken. Vergleichen Sie nun den Assemblercode mit dem Code aus Aufgabe 3.
2. Welche Erkenntnisse haben Sie aus den Resultaten der Aufgaben 3 und 4 gewonnen?

7.3.1 Lösung

1. Bei dieser Variante entsteht sehr effizienter Code. Eine Anweisung wie `r &= 6 << 4`; wird direkt in ein AND umgewandelt, die Operation `6 << 4` berechnet der Compiler, nicht das Laufzeitsystem: `lstinputlisting[language=C++, style=C++, multicol=2]900-Praktika/prak07/Loesung/A3/main.s`

```

1 movl    $0, -4(%rsp)
2 movl    $15, -4(%rsp)
3 movl    -4(%rsp), %eax
4 orl     $48, %eax
5 movl    %eax, -4(%rsp)
6 movl    -4(%rsp), %eax
7 andl    $96, %eax
8 movl    %eax, -4(%rsp)

```

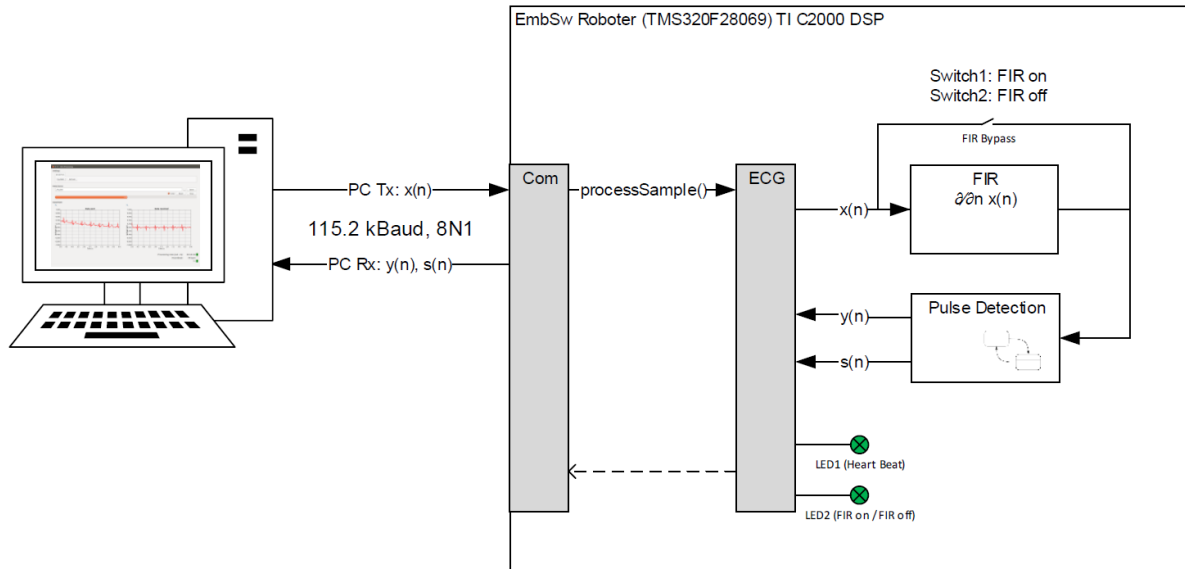
2. Bitmasken verwenden, Bitfelder nicht.

8 Lab 8 Digitale Ableitung und Pulsdetektion auf dem EmbSW-Roboter

In diesem Praktikum soll eine EKG-Messung auf dem EmbSW-Roboter verarbeitet werden. Das Ziel ist, eine robuste Detektion der Herzpulse zu implementieren, damit der korrekte Puls auf der PC-Software angezeigt wird.

Systemübersicht

Die EKG-Messdaten werden vom PC einzeln über die serielle Schnittstelle auf die Hardware übertragen. Jedes Sample wird durch den Aufruf von `Ecg::processSample()` sofort verarbeitet, nachdem es vollständig empfangen wurde. Anschliessend wird es mit der detektierten Phase des Herzpulses zu einem Telegramm zusammengepackt und wieder an die PC-Software zurückgeschickt. Der Signalfussplan ist in Abbildung 1 dargestellt.



Die einzelnen Verarbeitungsblöcke (hier FIR und Pulse Detection) sind seriell zusammengeschaltet und können ein- bzw. ausgeschaltet werden. Mit dem Switch1 auf dem Roboter kann das FIR-Filter eingeschalten bzw. mit Switch2 wieder ausgeschaltet werden.

Klassendiagramm der Vorgabe

Das Vorgabeprojekt besteht aus den Klassen `Ecg`, `Fir` und `PulseDetection`. Zudem wird der Unit Test für diese Einheit zur Verfügung gestellt. Die Datenverarbeitung startet mit dem Aufruf von `Ecg::processSample()`. Die Klasse `Ecg` beinhaltet ein Array von Algorithmen, die für jedes Sample der Reihe nach abgearbeitet werden. Der neu berechnete Wert und ein detektierter Herzschlag wird schliesslich von der Funktion `Ecg::processSample()` zurück gegeben. Alle Algorithmen besitzen dieselbe Basisklasse `Algorithm`. In den Unterklassen muss die virtuelle Funktion `Algorithm::process(float, float&)` überschrieben werden, um die nötige Funktion zu implementieren. Wird diese Funktion von der abgeleiteten Klasse nicht überschrieben, so hat dieser Algorithmus keinen Einfluss auf die Signalverarbeitung, da der Ausgangswert gleich dem Eingangswert gesetzt wird.

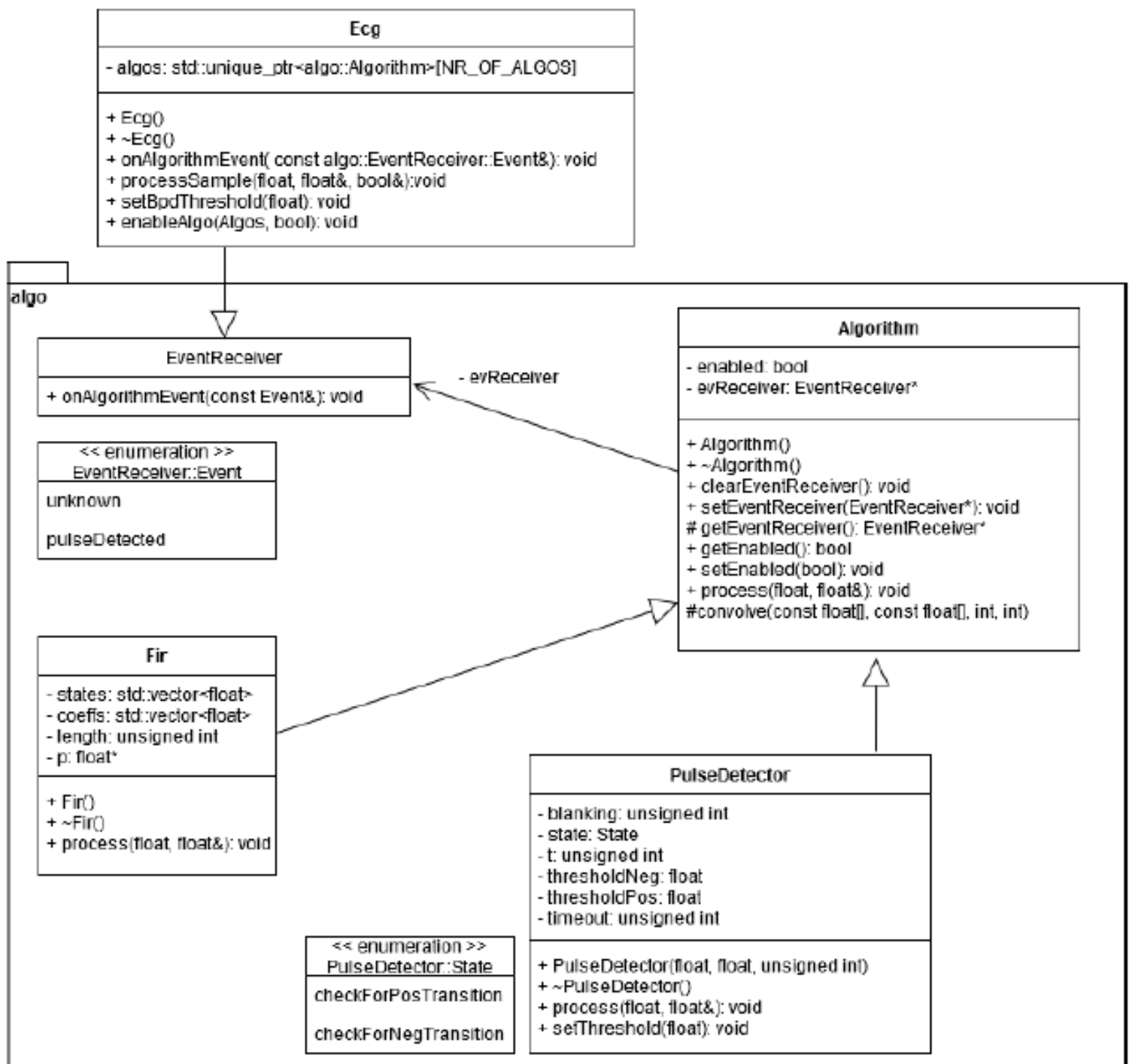


Abbildung 2: Klassendiagramm der EKG

8.1 Aufgabe 1: Implementation des FIR-Filters

Für eine robuste Detektion der Herzpulse soll das EKG-Signal zuerst über einen geeigneten Filter abgeleitet werden. Der Filter wird in Form eines FIR-Filters implementiert und erbt von der Klasse `Algorithm`. Die einfachste Variante wäre ein Filter mit den Koeffizienten $[1 \ -1]$, was einem Differenziator entspricht. Das Ausgangssignal berechnet sich nach: $y[n] = 1 \cdot x[n] - 1 \cdot x[n - 1]$; Dieses Filter reagiert stark auf die Rauschkomponente des Signals und liefert eine unbrauchbare Ableitung. Besser ist es, die Ableitung mit einer gleichzeitigen Bandpass-Filterung zu kombinieren. Eine Möglichkeit dafür ist das Savitzky-Golay-Filter (Savitzky & Golay, 1964). Eine Analyse in MATLAB zeigt, dass eine Fensterbreite von 5 Samples und eine Ordnung von 3 eine optimale Ableitung generiert. Die resultierenden Koeffizienten sind unten abgebildet.

```
static const float sgFilterCoeffs[] = { 0.0833, -0.6667, 0.0000, 0.6667, -0.0833 };
```

1. Definieren Sie die oben erwähnten Koeffizienten in der Klasse Ecg.
2. Vervollständigen Sie die Methode process() der Klasse FIR indem Sie ein FIR-Filter implementieren. Mögliche Implementationen finden Sie im Buch *Introduction to Signal Processing* von Sophocles J. Orfanidis, Kapitel 4.2.4 *Hardware Realizations and Circular Buffers*.
3. Testen Sie die korrekte Funktion mit dem Test Case FirTest des ECG Unit Tests. (Hinweis: Test Case PulseDetectorTest kann für diese Aufgabe auskommentiert werden.)

8.1.1 Lösung

1. siehe Eclipse-Projekt in ./Loesung/Ecg/, im speziellen die Klassen ./Loesung/Ecg/algos/FIR.cpp und Filterkoeffizienten in ./Loesung/Ecg/Ecg.cpp: `const float Ecg::sgFilterCoeffs[] = {0.0833, -0.6667, 0.0, 0.6667, -0.0833}`; Die Implementation des FIR Filters ist für das bessere Verständnis sehr allgemein gehalten. Die Basis der Implementation stammt vom Buch *Introduction to Signal Processing* von Sophocles J. Orfanidis, Kapitel 4.2.4 *Hardware Realizations and Circular Buffers*. Für den verwendeten DSP gibt es eine Implementation eines FIR Filters von TI selbst in einer DSP-Library. Zu empfehlen sind die DSP Algorithmus-Implementationen von den Herstellern, da sie meistens sehr effizient codiert sind.
2. Die Unit Test Cases DisabledTest und FirTest muss fehlerfrei ausgeführt werden können. Siehe Eclipse-Projekt in ./Loesung/Ecg

```

1  /*
2  * Ecg.h
3  *
4  * (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
5  */
6
7  #ifndef ECG_H_
8  #define ECG_H_
9
10 #include <memory>
11
12 #include "algo/EventReceiver.h"
13 #include "algo/Algorithm.h"
14
15 class Ecg : public algo::EventReceiver
16 {
17 public:
18     ///< Used algorithms
19     enum class Algos
20     {
21         fir = 0, ///< FIR algorithm
22         pulseDet ///< pulse detection algorithm
23     };
24
25     /**
26      * Ctor
27      */
28     Ecg();
29
30     /**
31      * Dtor
32      */
33     ~Ecg() override = default;
34
35     // overrides EventReceiver::onAlgorithmEvent
36     void onAlgorithmEvent(const algo::EventReceiver::Event
37         ↳ & ev) override;

```

```

38
39     /**
40      * Enables/disables the algorithm.
41      * @param[in] algo algorithm to enable/disable
42      * @param[in] en true to enable and false to disable
43      * ↳ the algorithm
44      */
45     void enableAlgo(Algos algo, bool en);
46
47     /**
48      * Sets the balanced photodetector (BPD) threshold.
49      * @param[in] value threshold
50      */
51     void setBpdThreshold(float value);
52
53     /**
54      * Processes a sample.
55      * @param[in] input input sample
56      * @param[out] output output sample
57      * @param[out] state pulse detector state (true: new
58      * ↳ pulse detected)
59      */
60     void processSample(float input, float& output, bool&
61         ↳ state);
62
63 private:
64
65     enum { NR_OF_ALGOS = static_cast<int>(Algos::pulseDet)
66         ↳ + 1 };
67
68     static const float sgFilterCoeffs[];
69     static const unsigned int sgFilterLength;
70
71     bool isNewPulseDetected;
72
73     std::unique_ptr<algo::Algorithm> algos[NR_OF_ALGOS];
74 };
75
76 #endif /* ECG_H_ */

```

```

1  /*
2  * Ecg.cpp
3  *
4  * (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
5  */
6
7  #include "Ecg.h"
8
9  #include <cassert>
10
11 #include "algo/FIR.h"
12 #include "algo/PulseDetector.h"
13

```

```

14 using algo::FIR;
15 using algo::PulseDetector;
16
17 const float Ecg::sgFilterCoeffs[] =
18 {
19     // TODO: Add the correct filter coefficients
20     ↳ to get a robust derivation
21     //SOLUTION_BEGIN
22     -0.0833,
23     0.6667,
24     //SOLUTION_END
25     0.0,
26     //SOLUTION_BEGIN

```

```

26         -0.6667,
27         0.0833,
28         //SOLUTION_END
29     };
30
31     const unsigned int Ecg::sgFilterLength = sizeof(sgFilterCoeffs
32         ↪ ) / sizeof(float);
33
34     Ecg::Ecg()
35     {
36         algos[static_cast<int>(Algos::fir)] = std::make_unique
37             ↪ <FIR>( sgFilterCoeffs, sgFilterLength );
38         algos[static_cast<int>(Algos::pulseDet)] = std:::
39             ↪ make_unique<PulseDetector>(0.1, -0.1);
40
41         algos[static_cast<int>(Algos::pulseDet)]->
42             ↪ setEventReceiver(this);
43
44         algos[static_cast<int>(Algos::fir)]->setEnabled(false)
45             ↪ ;
46         algos[static_cast<int>(Algos::pulseDet)]->setEnabled(
47             ↪ false);
48     }
49
50     void Ecg::processSample(float input, float& output, bool&
51         ↪ state)
52     {
53         isNewPulseDetected = false;
54         output = input;
55         for (int i = 0; i < NR_OF_ALGOS; ++i)
56         {
57             if (algos[i] != 0)
58             {
59                 algos[i]->process(output, output);
60             }
61         }
62         state = isNewPulseDetected;
63     }
64
65     void Ecg::setBpdThreshold(float value)
66     {
67         PulseDetector& pd = *static_cast<PulseDetector*>(algos
68             ↪ [static_cast<int>(Algos::pulseDet)].get());
69         pd.setThreshold(value);
70     }
71
72     void Ecg::onAlgorithmEvent(const EventReceiver::Event& ev)
73     {
74         switch (ev)
75         {
76             case EventReceiver::Event::pulseDetected:
77                 isNewPulseDetected = true;
78                 break;
79             default:
80                 break;
81         }
82     }
83
84     void Ecg::enableAlgo(Algos algo, bool en)
85     {
86         switch (algo)
87         {
88             case Algos::fir:
89                 algos[static_cast<int>(Algos::fir)]->
90                     ↪ setEnabled( en );
91                 break;
92             case Algos::pulseDet:
93                 algos[static_cast<int>(Algos::pulseDet)]->
94                     ↪ setEnabled( en );
95                 break;
96             default:
97                 break;
98         }
99     }

```

```

1  //
2  // testEcg.cpp
3  //
4  // implements the tests for the Ecg class.
5  //
6  // (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
7  //
8
9  #include <iostream>
10 #include <fstream>
11 #include <memory>
12
13 #include <gtest/gtest.h>
14
15 #include "../Ecg.h"
16
17 using namespace std;
18
19 TEST(EcgTest, DisabledTest)
20 {
21     unique_ptr<ifstream> fEcgTestData = make_unique<ifstream>( "
22         ↪ ecg.dat" );
23     ASSERT_TRUE( fEcgTestData->is_open() );
24     istream& in = *fEcgTestData;
25
26     // drop comment lines
27     while (in.peek() == '#')
28     {
29         string s;
30         getline(in, s);
31     }
32
33     Ecg uut;
34     uut.enableAlgo(Ecg::Algos::fir, false);
35     uut.enableAlgo(Ecg::Algos::pulseDet, false);
36
37     // store data in buffer and convolve
38     float x;
39     float y;
40     float expY;
41     bool state;
42     bool expState;
43     int cnt = 0;
44     while ( in >> x )
45     {
46         uut.processSample( x, y, state );
47
48         EXPECT_FLOAT_EQ( y, x );
49         EXPECT_FALSE( state );
50
51         ++cnt;
52     }
53     EXPECT_EQ(500, cnt); // check if all samples are processed
54 }
55
56 TEST(EcgTest, FirTest)
57 {
58     unique_ptr<ifstream> fEcgTestData = make_unique<ifstream>( "
59         ↪ ecg.dat" );
60     ASSERT_TRUE( fEcgTestData->is_open() );
61     istream& in = *fEcgTestData;
62
63     // drop comment lines
64     while (in.peek() == '#')
65     {
66         string s;
67         getline(in, s);
68     }
69
70     unique_ptr<ifstream> fExpectedFirOutput = make_unique<
71         ↪ ifstream>( "ecgExpected.dat" );
72     ASSERT_TRUE( fExpectedFirOutput->is_open() );
73     istream& expectedYIn = *fExpectedFirOutput;
74
75     Ecg uut;
76     uut.enableAlgo(Ecg::Algos::fir, true);
77     uut.enableAlgo(Ecg::Algos::pulseDet, false);
78
79     // store data in buffer and convolve
80     float x;
81     float y;
82     float expY;
83     bool state;
84     bool expState;
85     int cnt = 0;
86     while ( in >> x &&
87         expectedYIn >> expY )
88     {
89         uut.processSample( x, y, state );
90
91         EXPECT_NEAR( y, expY, 1e-6 );
92         EXPECT_FALSE( state );
93         ++cnt;
94     }
95     EXPECT_EQ(500, cnt); // check if all samples are processed
96 }
97
98 TEST(EcgTest, PulseDetectorTest)
99 {

```



```

96     unique_ptr<ifstream> fEcgTestData = make_unique<ifstream>( " 116
           ↳ ecg.dat" );
97     ASSERT_TRUE( fEcgTestData->is_open() );
98     istream& in = *fEcgTestData;
99
100    // drop comment lines
101    while (in.peek() == '#')
102    {
103        string s;
104        getline(in, s);
105    }
106
107    unique_ptr<ifstream> fExpectedFirOutput = make_unique<
           ↳ ifstream>( "ecgExpected.dat" );
108    ASSERT_TRUE( fExpectedFirOutput->is_open() );
109    istream& expectedYIn = *fExpectedFirOutput;
110
111    unique_ptr<ifstream> fExpectedState = make_unique<ifstream>(
           ↳ "stateExpected.dat" );
112    ASSERT_TRUE( fExpectedState->is_open() );
113    istream& expectedStateIn = *fExpectedState;
114
115    Ecg uut;
116
117    uut.enableAlgo(Ecg::Algos::fir, true);
118    uut.enableAlgo(Ecg::Algos::pulseDet, true);
119
120    // store data in buffer and convolve
121    float x;
122    float y;
123    float expY;
124    bool state;
125    bool expState;
126    int cnt = 0;
127    while ( in >> x &&
           expectedYIn >> expY &&
           expectedStateIn >> expState )
128    {
129        uut.processSample( x, y, state );
130
131        EXPECT_NEAR( y, expY, 1e-6 );
132        EXPECT_EQ( state, expState );
133        ++cnt;
134    }
135    EXPECT_EQ(500, cnt); // check if all samples are processed
136
137 }

```

```

1  /*
2  * EventReceiver.h
3  *
4  * (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
5  */
6
7  #ifndef ALGO_EVENTRECEIVER_H_
8  #define ALGO_EVENTRECEIVER_H_
9
10 namespace algo
11 {
12
13 class EventReceiver
14 {
15 public:
16     ///> Algorithm event type.
17     enum class Event
18     {
19         unknown, ///< unknown event
20
21         pulseDetected ///< pulse detected event
22     };
23
24     /**
25      * Is called on each algorithm event.
26      * @param ev event
27      */
28     virtual void onAlgorithmEvent(const Event& ev) = 0;
29
30     /**
31      * Dtor
32      */
33     virtual ~EventReceiver() = default;
34 };
35
36 } /* namespace algo */
37
38 #endif /* ALGO_EVENTRECEIVER_H_ */

```

```

1  /*
2  * FIR.h
3  *
4  * (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
5  */
6
7  #ifndef ALGO_FIR_H_
8  #define ALGO_FIR_H_
9
10 #include <vector>
11
12 #include "../Algorithm.h"
13
14 namespace algo
15 {
16
17 class FIR : public Algorithm
18 {
19 public:
20     /**
21      * Ctor
22      */
23
24     FIR(const float* coeffs, unsigned int length);
25
26     /**
27      * Dtor
28      */
29     ~FIR() override = default;
30
31     // overrides Algorithm::process
32     void process(float input, float& output) override;
33
34 private:
35     std::vector<float> states;
36     std::vector<float> coeffs;
37     unsigned int length;
38     float* p;
39 };
40
41 } /* namespace algo */
42
43 #endif /* ALGO_FIR_H_ */

```

```

1  /*
2  * FIR.cpp
3  *
4  * (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
5  */
6
7  #include "../FIR.h"
8
9 namespace algo
10 {
11
12 FIR::FIR(const float* coeffs, unsigned int length) :
13     coeffs(coeffs, coeffs+length), // pointer are treated as
           ↳ iterator
           length(length)
14 {
15     states.resize(length, 0.0f);
16     p = states.data();
17 }
18
19 void FIR::process(float input, float& output)
20 {
21

```

```

22 if (getEnabled())
23 {
24     // TODO: Implement FIR filtering algorithm here.
25     //SOLUTION_BEGIN
26
27     *p = input;
28
29     output = 0.0;
30     int i;
31     for (i = 0; i < length-1; ++i)
32     {
33         output += coeffs[i] * (*p++);
34         if (p >= (states.data() + length)) //circular buffer
            ↳ implementation in SW
35
36         p = states.data();
37     }
38     output += coeffs[i] * (*p);
39
40     //SOLUTION_END
41 }
42 else
43 {
44     output = input; //set output value as input value --> "
            ↳ block short circuit"
45 }
46
47 } /* namespace algo */

```

```

1  /*
2   * PulseDetector.h
3   *
4   * (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
5   */
6
7 #ifndef ALGO_PULSEDETECTOR_H_
8 #define ALGO_PULSEDETECTOR_H_
9
10 #include "../Algorithm.h"
11
12 namespace algo
13 {
14
15 class PulseDetector : public Algorithm {
16 public:
17
18     /**
19      * Ctor
20      * @param[in] thPos positive going threshold
21      * @param[in] thNeg negative going threshold
22      * @param[in] blanking blanking number (default 3)
23      */
24     PulseDetector(float thPos, float thNeg, unsigned int
            ↳ blanking = 3);
25
26     /**
27      * Dtor
28      */
29     ~PulseDetector() override = default;
30
31     // overrides Algorithm::process
32
33     void process(float input, float& output) override;
34
35     /**
36      * Sets the pulse detection threshold.
37      * @param[in] value pulse detection threshold
38      */
39     void setThreshold(float value);
40
41 private:
42
43     float thresholdNeg;
44     float thresholdPos;
45
46     // TODO: Private declarations
47     //SOLUTION_BEGIN
48     enum class State
49     {
50         checkForNegTransition,
51         checkForPosTransition
52     };
53
54     unsigned int blanking;
55     unsigned int t;
56
57     State state;
58     //SOLUTION_END
59 };
60 } /* namespace algo */
61
62 #endif /* ALGO_PULSEDETECTOR_H_ */

```

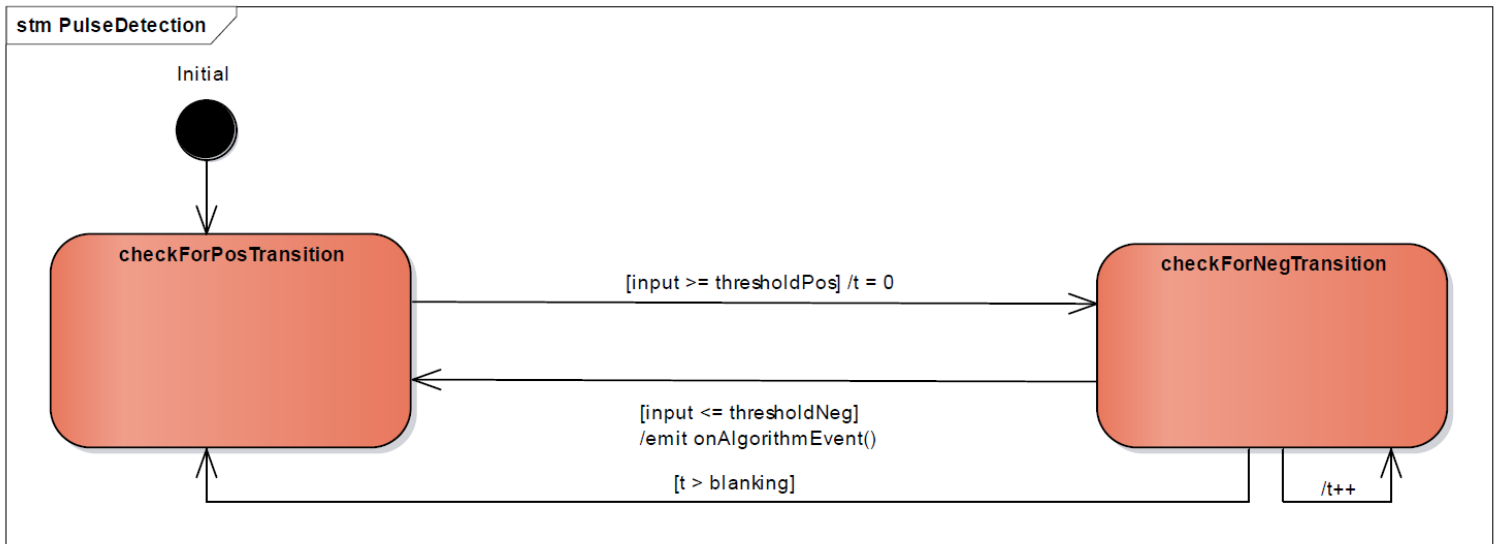
```

1  /*
2   * PulseDetector.cpp
3   *
4   * (C) G. Danuser, HSR Hochschule Rapperswil, April 2020
5   */
6
7 #include "../PulseDetector.h"
8
9 namespace algo
10 {
11
12 PulseDetector::PulseDetector(float thPos, float thNeg,
            ↳ unsigned int blanking)
13 // TODO: Initialize variables
14 //SOLUTION_BEGIN
15 : thresholdPos(thPos),
16   thresholdNeg(thNeg),
17   blanking(blanking),
18   state(State::checkForPosTransition),
19   t(0)
20 //SOLUTION_END
21 {
22
23 }
24
25 void PulseDetector::process(float input, float& output)
26 {
27     // TODO: Implement detection state machine
28     // Note: To fire an event call --> getEventReceiver()
            ↳ ->onAlgorithmEvent(AlgorithmEvReceiver::
            ↳ EvPulseDetected);
29
30     //SOLUTION_BEGIN
31     output = input;
32     if (getEnabled())
33     {
34         switch (state)
35         {
36             case State::checkForPosTransition:
37                 if (input >= thresholdPos )
38                 {
39                     t = 0;
40                     state = State::checkForNegTransition;
41                 }
42                 break;
43             case State::checkForNegTransition:
44                 if (input <= thresholdNeg)
45                 {
46                     if ( getEventReceiver() ) {
47                         getEventReceiver()->onAlgorithmEvent(EventReceiver
            ↳ ::Event::pulseDetected);
48                     }
49                     state = State::checkForPosTransition;
50                 }
51                 else if (t >= blanking)
52                 {
53                     state = State::checkForPosTransition;
54                 }
55                 break;
56             }
57         ++t;
58     }
59     //SOLUTION_END
60 }
61
62 void PulseDetector::setThreshold(float value)
63 {
64     thresholdPos = value;
65     thresholdNeg = -1*value;
66 }
67
68 } /* namespace algo */

```

8.2 Aufgabe 2: Implementation der Pulsdetektion

Vervollständigen Sie die Klasse `PulseDetector`. Die Methode `PulseDetector::process()` gibt das Eingangssignal wieder an den Ausgang weiter und soll nur für die Detektion der Pulse verwendet werden. Die Klasse `Ecg` registriert sich über `Algorithm::setEventReceiver()` und wird über diesen Callback-Mechanismus über Zustandswechsel des Pulsdetektors informiert. Sie müssen dafür sorgen, dass dieser Callback an den nötigen Stellen gefeuert wird. In Abbildung 3 ist die Finite State Machine (FSM) für die Pulsdetektion dargestellt. Die FSM besteht aus zwei Zuständen. Im Zustand `checkForNegTransition` wird das gefilterte EKG-Signal auf negative Steigungen und in `checkForPosTransition` auf positive Steigungen überprüft. Um die Robustheit der Pulsdetektion zu erhöhen, lohnt es sich, die Zustandswechsel nicht nur über das abgeleitete Signal vorzunehmen, sondern noch Zeitkriterien einzufügen. Auf einen negativen Puls in der Ableitung muss beispielsweise unmittelbar ein positiver in der gleichen Größenordnung folgen, damit kein DC-Offset entsteht.



1. Implementieren Sie die Finite State Machine.
2. Testen Sie die korrekte Funktion mit dem Test Case `PulseDetectorTest` des ECG Unit Tests.

8.2.1 Lösung

Die Detektion basiert auf den Samples der Ableitung des EKG-Signals, einer positiven und einer negativen Detektionsschwelle (im Diagramm *input*, *thresholdPos* und *thresholdNeg*). Zusätzlich existiert ein zeitliches Kriterium, damit sichergestellt ist, dass keine Störungen als Puls detektiert werden: unterschreitet die Ableitung die positive Schwelle, muss innerhalb von *blanking*-Samples auch die negative Schwelle überschritten werden. Ansonsten handelt es sich mit grosser Wahrscheinlichkeit nicht um einen Herzschlag.

1. siehe Klasse `PulseDetector` in `./Loesung/Ecg/algo/PulseDetector.h`
2. Alle Unit Test Cases muss fehlerfrei ausgeführt werden können. Siehe Eclipse-Projekt in `./Loesung/Ecg`

```
1  /*
2   * Algorithm.h
3   *
4   * Created on: 08.04.2015
5   * Author: phoerler
6   */
7
8  #ifndef ALGOS_ALGORITHM_H_
9  #define ALGOS_ALGORITHM_H_
10
11  #include "../EventReceiver.h"
12
13  namespace algo
```

```
14  {
15
16  class Algorithm
17  {
18  public:
19      /**
20       * Ctor
21       */
22      Algorithm() :
23          enabled(false),
24          evReceiver( nullptr )
25      {
26      }
```

```

27  /**
28   * Dtor
29   */
30  virtual ~Algorithm() = default;
31
32  /**
33   * Processing function of the algorithm. Normally, this
34   *   ↳ function will be overwritten by the derived
35   *   ↳ classes, otherwise the input is passed to the
36   *   ↳ output directly.
37   * @param[in] input input sample
38   * @param[out] output output sample
39   */
40  virtual void process(float input, float& output)
41  {
42      output = input;
43  }
44
45  /**
46   * Enable the algorithm
47   * @param[in] val enable or disable the algorithm
48   */
49  void setEnabled(bool val)
50  {
51      enabled = val;
52  }
53
54  /**
55   * Returns whether the algorithm is enabled or not.
56   * @return returns whether the algorithm is enabled or not
57   */
58  bool getEnabled() const
59  {
60      return enabled;
61  }
62
63  /**
64   * Stores the given object as receiver for algorithm
65   *   ↳ events.
66   * @param[in] receiver reference to the object which is
67   *   ↳ called if an algorithm event occurred
68   */
69  void setEventReceiver(EventReceiver* receiver)
70  {
71      evReceiver = receiver;
72  }

```

```

69  /**
70   * Removes the event receiver.
71   */
72  void clearEventReceiver()
73  {
74      evReceiver = nullptr;
75  }
76
77  protected:
78  EventReceiver* getEventReceiver() const
79  {
80      return evReceiver;
81  }
82
83  /**
84   * Convolves input buffer and filter with a given length
85   *   ↳ at a running index.
86   * @param[in] filter filter coefficients of length lf
87   * @param[in] buffer sample buffer of length lf
88   * @param[in] lf length of the sample and filter
89   *   ↳ coefficient buffers
90   * @return returns the result of the discrete convolution
91   */
92  static float convolve( const float filter[],
93                        const float buffer[],
94                        int lf,
95                        int pos )
96  {
97      float sum = 0;
98      int k;
99      for (k = 0; k < lf; ++k)
100      {
101          sum += filter[k] * buffer[(lf + pos - k) % lf];
102      }
103      return sum;
104  }
105
106  private:
107      bool enabled;
108      EventReceiver* evReceiver;
109  };
110
111  } /* namespace algo */
112
113 #endif /* ALGOS_ALGORITHM_H_ */

```

9 Lab 9 Code Bloat bei Templates, Code Hoisting

9.1 Aufgabe 1: Untersuchung von Assemblercode bei Templates

Die folgenden Optionen der GNU-Compiler könnten nützlich sein:

- E Precompile only, der Output wird auf stdout geschrieben
- S Assembleroutput, ohne Objectfile erzeugen
- c nur compilieren
- O0 keine Optimierung
- O1 Optimierungsstufe 1 (siehe g++ - Help für Details)
- O2 Optimierungsstufe 2
- O3 Optimierungsstufe 3
- Os Optimierung auf Codegrösse

Hinweis Jede Funktion in einem Objectfile oder Binary liegt an einer bestimmten Adresse und ist mit einem Symbol benannt, sofern im Debugmodus compiliert wurde. Mit dem Befehl `nm [-C] [objfile]` kann der Inhalt einer solchen Datei ausgegeben werden. Mit der Option `-C` werden die Symbolnamen demangled. Siehe dazu auch die entsprechende `man` page.

1. Untersuchen Sie den Code im Verzeichnis `./Vorgabe/StackTemplate`. Verschaffen Sie sich einen Überblick und werden Sie vertraut mit der Templateprogrammierung. Beachten Sie, dass einzelne Elementfunktionen implizit inline sind.
2. Compilieren Sie den Code mit den Optimierungsstufen 0 und 3. Untersuchen Sie jeweils den entstandenen Assemblercode sowie die Codegrösse. Achten Sie vor allem darauf, ob die Funktionen aufgerufen werden (mit `call`) oder inline sind. Sind die Funktionen einfach oder mehrfach vorhanden?
3. Ändern Sie den Code nun so ab, dass im Hauptprogramm ein zweiter `int`-Stack mit unterschiedlicher Grösse definiert wird. Compilieren Sie den Code wiederum unter Verwendung der Optimierungsstufen 0 und 3. Untersuchen Sie jeweils den entstandenen Assemblercode sowie die Codegrösse. Achten Sie darauf, ob die Funktionen aufgerufen werden (mit `call`) oder inline sind. Sind die Funktionen einfach oder mehrfach vorhanden?

9.1.1 Lösung

Alle Programme wurden mit dem Compiler `g++ v7.4.0` übersetzt.

1. just do it.
2. Wenn mit Optimierungsstufe 0 compiliert wird, sind keine Elementfunktionen inline. Alle Elementfunktionen sind einfach vorhanden und werden mit `call` aufgerufen. Bei Optimierungsstufe 3 sind sämtliche Funktionen der Klasse `Stack` inline. In der folgenden Tabelle sind die Codegrössen ersichtlich.

| Optimierung O... | g++ v4.7.3 Cygwin | g++ v4.8.4 | g++ v5.4.0 | g++ v7.4.0 |
|------------------|-------------------|------------|------------|------------|
| 0 | 61'738 | 13'913 | 14'024 | 13'712 |
| 3 | 60'043 | 13'369 | 13'480 | 13'120 |

3. siehe `./Loesung/A1`

```
1  /*
2  * Stack.h
3  *
4  * Created on: 24.04.2015
5  * Author: rbondere
6  */
7
8  #ifndef STACK_H_
9  #define STACK_H_
10
11  template<typename ElemType, int size = 10>
12  class Stack
13  {
```

```
14  public:
15      Stack();
16      // Default-Konstruktor
17
18      void push(const ElemType& e);
19      // legt ein Element auf den Stack, falls der Stack noch
20      // ↳ nicht voll ist
21      // wasError() gibt Auskunft, ob push() erfolgreich war
22
23      ElemType pop();
24      // nimmt ein Element vom Stack, falls der Stack nicht
25      // ↳ leer ist
26      // wasError() gibt Auskunft, ob pop() erfolgreich war
```

```

25     ElemType peek() const;
26     // liest das oberste Element vom Stack, falls der Stack
27     // ↳ nicht leer ist
28     // wasError() gibt Auskunft, ob peek() erfolgreich war
29
30     bool isEmpty() const {return top == 0;}
31     // return: true: Stack ist leer
32     //         false: sonst
33
34     bool isFull() const;
35     // return: true: Stack ist voll
36     //         false: sonst
37
38     bool wasError() const {return error;}
39     // return: true: Operation war fehlerhaft
40
41     //         false: sonst
42
43     private:
44         ElemType elems[size]; // Speicher fuer Speicherung des
45         // ↳ Stacks
46         int top; // Arrayindex des naechsten freien
47         // ↳ Elements
48         mutable bool error; // true: Fehler passiert; false:
49         // ↳ sonst
50         // mutable: auch const-Methoden koennen dieses Attribut
51         // ↳ setzen
52 };
53
54 // ugly include
55 #include "Stack.cpp"
56 #endif // STACK_H_

```

```

1  /*
2  * Stack.cpp
3  *
4  * Created on: 24.04.2015
5  * Author: rbondere
6  */
7
8
9  template<typename ElemType, int size>
10 Stack<ElemType, size>::Stack() :
11     top(0), error(false)
12 {
13 }
14
15 template<typename ElemType, int size>
16 void Stack<ElemType, size>::push(const ElemType& e)
17 {
18     error = isFull();
19     if (!error)
20     {
21         elems[top] = e;
22         ++top;
23     }
24 }
25
26 template<typename ElemType, int size>
27 ElemType Stack<ElemType, size>::pop()
28 {
29     error = isEmpty();
30     if (!error)
31     {
32         --top;
33         return elems[top];
34     }
35     else
36         return 0;
37 }
38
39 template<typename ElemType, int size>
40 ElemType Stack<ElemType, size>::peek() const
41 {
42     error = isEmpty();
43     if (!error)
44         return elems[top - 1];
45     else
46         return 0;
47 }
48
49 template<typename ElemType, int size>
50 bool Stack<ElemType, size>::isFull() const
51 {
52     return top == size;
53 }

```

```

1  /*
2  * StackTest.cpp
3  *
4  * Created on: 09.05.2012
5  * Author: rbondere
6  */
7
8  #include "StackUI.h"
9  using namespace std;
10
11 enum
12 {
13     stackSize = 3
14 };
15
16 int main(void)
17 {
18     StackUI<int, stackSize> sUI;
19     sUI.dialog();
20     StackUI<int, 4> sUI4;
21     sUI4.dialog();
22
23     return 0;
24 }

```

```

1  /*
2  * StackUI.h
3  *
4  * User Interface for Stack application
5  * Created on: 09.05.2012
6  * Author: rbondere
7  */
8
9  #ifndef STACKUI_H_
10 #define STACKUI_H_
11 #include "Stack.h"
12
13 template<typename ElemType, int size = 10>
14 class StackUI
15 {
16     public:
17         void dialog();
18         // starts the user dialog
19     private:
20         Stack<ElemType, size> s;
21 };
22
23 // ugly include
24 #include "StackUI.cpp"
25
26 #endif /* STACKUI_H_ */

```

```

1  /*
2   * StackUI.cpp
3   *
4   * Created on: 09.05.2012
5   * Author: rbondere
6   */
7
8  #include <iostream>
9
10 // DO NOT #include "StackUI.h" !!
11 using namespace std;
12
13 template<typename ElemType, int size>
14 void StackUI<ElemType, size>::dialog()
15 {
16     char ch = 0;
17     ElemType e;
18     do
19     {
20         cout << "\n\nOperation (Quit, pUsh, pOp, peeK, isEmpty)
21             ↪ ";
22         cin >> ch;
23         switch (ch)
24         {
25             case 'q':
26             case 'Q': // quit
27                 break;
28             case 'u':
29             case 'U': // push
30                 cout << "\nElement to push: ";
31                 cin >> e;
32                 s.push(e);
33                 if (s.wasError())
34                     cout << "\nError: Stack full.";
35
36                 break;
37             case 'o':
38             case 'O': // pop
39                 e = s.pop();
40                 if (s.wasError())
41                     cout << "\nError: Stack is empty (nothing to pop).
42                     ↪ ";
43                 else
44                     cout << "\nPopped element " << e;
45                 break;
46             case 'k':
47             case 'K': // peek
48                 e = s.peek();
49                 if (s.wasError())
50                     cout << "\nError: Stack is empty (nothing to peek)
51                     ↪ .";
52                 else
53                     cout << "\nPeeked element " << e;
54                 break;
55             case 'e':
56             case 'E': // isEmpty
57                 if (s.isEmpty())
58                     cout << "\nStack is empty.";
59                 else
60                     cout << "\nStack contains elements.";
61                 break;
62             default:
63                 cout << "\nInvalid operation.";
64                 break;
65         }
66     } while (ch != 'Q' && ch != 'q');
67 }

```

Wenn ohne Optimierung kompiliert wird, sind keine Elementfunktionen inline. Alle Elementfunktionen inklusive die Konstruktoren sind doppelt mit praktisch identischem Code vorhanden und werden mit call aufgerufen. Die Texte, die mit cout ausgegeben werden, sind nur einfach vorhanden.

```

1  Build mit -O0:
2  $ nm -C StackTest
3  ...
4  00000000000000f90 W Stack<int, 3>::pop()
5  00000000000000f1e W Stack<int, 3>::push(int const&)
6  00000000000000ab2 W Stack<int, 3>::Stack()
7  00000000000000ab2 W Stack<int, 3>::Stack()
8  00000000000000c2 W Stack<int, 4>::pop()
9  000000000000001050 W Stack<int, 4>::push(int const&)
10 00000000000000ce8 W Stack<int, 4>::Stack()
11 00000000000000ce8 W Stack<int, 4>::Stack()
12 00000000000000ad0 W StackUI<int, 3>::dialog()
13 00000000000000a7a W StackUI<int, 3>::StackUI()
14 00000000000000a7a W StackUI<int, 3>::StackUI()
15 00000000000000d06 W StackUI<int, 4>::dialog()
16 00000000000000a96 W StackUI<int, 4>::StackUI()
17 00000000000000a96 W StackUI<int, 4>::StackUI()
18 00000000000000fec W Stack<int, 3>::peek() const
19 000000000000001182 W Stack<int, 3>::isFull() const
20 00000000000000103a W Stack<int, 3>::isEmpty() const
21 00000000000000f7e W Stack<int, 3>::wasError() const
22 00000000000000111e W Stack<int, 4>::peek() const
23 00000000000000119a W Stack<int, 4>::isFull() const
24 00000000000000116c W Stack<int, 4>::isEmpty() const
25 0000000000000010b0 W Stack<int, 4>::wasError() const
26 ...
27
28 Build mit -O3:
29 $ nm -C StackTest
30 ...
31 00000000000000a80 W StackUI<int, 3>::dialog()
32 00000000000000d50 W StackUI<int, 4>::dialog()
33 ...
34

```

Bei älteren Compilerversionen (z.B. g++-Version v3.4.4) waren mit Optimierungstufe 3 nur die impliziten inline-Elementfunktionen inline. Alle weiteren Elementfunktionen waren ebenfalls doppelt vorhanden und wurden mit call aufgerufen. Wenn hingegen g++ v4.8.4, g++ v5.4.0 oder g++ v7.4.0 genommen wird, sind alle Elementfunktionen ausser `StackUI::dialog()` inline. Diese Funktion ist dann zweifach vorhanden, je für einen Stack der Grösse 3 und der Grösse 4, und wird wie folgt aufgerufen:

```

1  call _ZN7StackUIiLi3EE6dialogEv
2  ...
3  call _ZN7StackUIiLi4EE6dialogEv

```

In der folgenden Tabelle sind die Codegrößen ersichtlich.

| Optimierung O... | g++ v4.7.3 Cygwin | g++ v4.8.4 | g++ v5.4.0 | g++ v7.4.0 |
|------------------|-------------------|------------|------------|------------|
| 0 | 66'052 | 14'440 | 14'544 | 14'240 |
| 3 | 61'389 | 13'420 | 13'528 | 13'168 |

9.2 Aufgabe 2: Verhindern von Code Bloat durch Code Hoisting

In dieser Aufgabe soll untersucht werden, wie mittels Code Hoisting der bei Templates häufig entstehende Code Bloat vermieden werden kann. Die heutigen Compiler erzeugen zwar immer besseren Code. Es ist trotzdem sinnvoll, mittels Code Hoisting möglichen Code Bloat zu vermeiden.

Die Klasse `StackUI` lassen wir unverändert. Analysieren Sie, welche Teile in der Klasse `Stack` unabhängig von der Grösse sind. Lagern Sie alle diese Teile in eine Basisklasse aus. Compilieren Sie den Code wiederum mit den Optimierungsstufen 0 und 3. Untersuchen Sie jeweils den entstandenen Assemblercode. Achten Sie darauf, ob die Funktionen aufgerufen werden (mit `call`) oder inline sind. Sind die Funktionen einfach oder mehrfach vorhanden?

9.2.1 Lösung

Code hoisting, siehe `./Loesung/A2`. Beachten Sie auch die Kommentare im Source Code.

In der Klasse `Stack` sollen alle von der Grösse unabhängigen Teile in eine Basisklasse ausgelagert werden. Die Analyse, was unabhängig von der Grösse ist, ist nicht so trivial. Sind beispielsweise die Elementfunktionen `isEmpty()` und `isFull()` unabhängig von der Grösse oder nicht? Weiss man das bereits bei der Deklaration oder ist es erst bei der Implementation bekannt? Aus der Implementation geht hervor, dass die Methode `isEmpty()` unabhängig ist, `isFull()` jedoch nicht. Alle weiteren Elementfunktionen, die `isFull()` benötigen, sind demnach auch nicht unabhängig, das ist z.B. `push()`.

Damit die Unterklassen einfach auf die Attribute zugreifen können, sollten `top` und `error` als `protected` statt `private` definiert werden. In der Basisklasse ist der Array `elems[]` nicht bekannt, da dieser ja eben abhängig von der Grösse ist. Also müssten beinahe alle Elementfunktionen (alle, die auf den Array zugreifen) in die Unterklasse. Um das zu verhindern, kann im Ctor der Basisklasse ein Pointer auf den `elems`-Array gesetzt werden. In der Basisklasse wird dann damit gearbeitet. Dies ist zwar nicht gerade schön, allerdings löst es den Code Bloat. Man beachte, dass der Ctor von `StackNoSize` `protected` gesetzt wird, damit nur eine Unterklasse diesen aufrufen kann.

Wenn ohne Optimierung kompiliert wird, sind keine Elementfunktionen inline. Alle Elementfunktionen und Konstruktoren der Basisklasse sind nur noch einfach vorhanden, alle anderen Elementfunktionen sind doppelt vorhanden. Bei älteren Compiler-Versionen (z.B. g++-Version v3.4.4) waren mit Optimierungsstufe 3 die impliziten inline-Elementfunktionen inline. Funktionen der Basisklasse waren nur noch einfach, alle anderen Funktionen doppelt vorhanden. Wenn hingegen g++ v7.4.0 genommen wird, sind alle Elementfunktionen ausser `StackUI::dialog()` inline. Diese Funktion ist dann zweifach vorhanden, je für einen Stack der Grösse 3 und der Grösse 4.

```
1  /*
2   * Stack.h
3   *
4   * Created on: 24.04.2015
5   * Author: rbondere
6   */
7
8  #ifndef STACK_H_
9  #define STACK_H_
10 #include "StackNoSize.h"
11
12 template<typename ElemType, int size = 10>
13 class Stack: public StackNoSize<ElemType>
14 {
15     public:
16         Stack();
17         // Default-Konstruktor
18 }
```

```
19     void push(const ElemType& e);
20     // legt ein Element auf den Stack, falls der Stack noch
21     //      ↳ nicht voll ist
22     // wasError() gibt Auskunft, ob push() erfolgreich war
23
24     bool isFull() const;
25     // return: true: Stack ist voll
26     //      ↳ false: sonst
27
28     private:
29         ElemType elems[size]; // Speicher fuer Speicherung des
30                                //      ↳ Stacks
31 };
32
33 // ugly include
34 #include "Stack.cpp"
35 #endif // STACK_H_
```

```
1  /*
2   * Stack.cpp
3   *
4   * Created on: 24.04.2015
5   * Author: rbondere
6   */
7
8
9  template<typename ElemType, int size>
10 Stack<ElemType, size>::Stack() :
11     StackNoSize<ElemType>(&elems[0])
12 {
13 }
14 }
```

```
15 template<typename ElemType, int size>
16 void Stack<ElemType, size>::push(const ElemType& e)
17 {
18     StackNoSize<ElemType>::error = isFull(); // Compiler kennt
19     //      ↳ error nicht ohne Class-Qualifier
20     if (!StackNoSize<ElemType>::error)
21     {
22         elems[StackNoSize<ElemType>::top] = e; // dito top
23         ++StackNoSize<ElemType>::top;
24     }
25 }
26
27 template<typename ElemType, int size>
28 bool Stack<ElemType, size>::isFull() const
```



```
28 {
29     return StackNoSize<ElemType>::top == size;
30 }

1  /*
2  * StackNoSize.h
3  *
4  * Created on: 09.05.2012
5  * Author: rbondere
6  */
7
8 #ifndef STACKNOSIZE_H_
9 #define STACKNOSIZE_H_
10
11 template<typename ElemType>
12 class StackNoSize
13 {
14 public:
15     ElemType pop();
16     // nimmt ein Element vom Stack, falls der Stack nicht leer
17     // ↳ ist
18     // wasError() gibt Auskunft, ob pop() erfolgreich war
19
20     ElemType peek() const;
21     // liest das oberste Element vom Stack, falls der Stack
22     // ↳ nicht leer ist
23     // wasError() gibt Auskunft, ob peek() erfolgreich war
24
25     bool isEmpty() const {return top == 0;}
26     // return: true: Stack ist leer
27
28     // false: sonst
29
30     bool wasError() const {return error;}
31     // return: true: Operation war fehlerhaft
32     // false: sonst
33
34 protected:
35     StackNoSize(ElemType* pArr = 0);
36     // Default-Konstruktor (nur Unterklassen sollen Objekte
37     // ↳ gruenden koennen)
38     int top; // Arrayindex des naechsten freien
39     // ↳ Elements
40     mutable bool error; // true: Fehler passiert; false:
41     // ↳ sonst
42     // mutable: auch const-Methoden koennen dieses Attribut
43     // ↳ setzen
44
45 private:
46     ElemType* pElems; // Pointer auf Array in Unterklasse
47     // ↳ (nicht ganz sauber)
48 };
49
50 // ugly include
51 #include "StackNoSize.cpp"
52 #endif // STACKNOSIZE_H_
```

```
1  /*
2  * StackNoSize.cpp
3  *
4  * Created on: 24.04.2015
5  * Author: rbondere
6  */
7
8
9 template<typename ElemType>
10 StackNoSize<ElemType>::StackNoSize(ElemType* pArr) :
11     top(0), error(false), pElems(pArr)
12 {
13 }
14
15 template<typename ElemType>
16 ElemType StackNoSize<ElemType>::pop()
17 {
18     error = isEmpty();
19
20     if (!error)
21     {
22         --top;
23         return pElems[top];
24     }
25     else
26         return ElemType(0);
27 }
28
29 template<typename ElemType>
30 ElemType StackNoSize<ElemType>::peek() const
31 {
32     error = isEmpty();
33     if (!error)
34         return pElems[top - 1];
35     else
36         return ElemType(0);
37 }
```

```
1  /*
2  * StackTest.cpp
3  *
4  * Created on: 09.05.2012
5  * Author: rbondere
6  */
7
8 #include "StackUI.h"
9 using namespace std;
10
11 enum
12 {
13     stackSize = 3
14 };
15
16 int main(void)
17 {
18     StackUI<int, stackSize> sUI;
19     sUI.dialog();
20     StackUI<int, 4> sUI4;
21     sUI4.dialog();
22
23     return 0;
24 }
```

```
1  /*
2  * StackUI.h
3  *
4  * User Interface for Stack application
5  * Created on: 09.05.2012
6  * Author: rbondere
7  */
8
9 #ifndef STACKUI_H_
10 #define STACKUI_H_
11 #include "Stack.h"
12
13 template<typename ElemType, int size = 10>
14 class StackUI
15 {
16 public:
17     void dialog();
18     // starts the user dialog
19 private:
20     Stack<ElemType, size> s;
21 };
22
23 // ugly include
24 #include "StackUI.cpp"
25
26 #endif /* STACKUI_H_ */
```

```

1  /*
2  * StackUI.cpp
3  *
4  * Created on: 09.05.2012
5  * Author: rbondere
6  */
7
8  #include <iostream>
9
10 // DO NOT #include "StackUI.h" !!
11 using namespace std;
12
13 template<typename ElemType, int size>
14 void StackUI<ElemType, size>::dialog()
15 {
16     char ch = 0;
17     ElemType e;
18     do
19     {
20         cout << "\n\nOperation (Quit, pUsh, pOp, peeK, isEmpty) ";
21         cin >> ch;
22         switch (ch)
23         {
24             case 'q':
25                 case 'Q': // quit
26                     break;
27             case 'u':
28                 case 'U': // push
29                     cout << "\nElement to push: ";
30                     cin >> e;
31                     s.push(e);
32                     if (s.wasError())
33                         cout << "\nError: Stack full.";
34
35                     break;
36             case 'o':
37                 case 'O': // pop
38                     e = s.pop();
39                     if (s.wasError())
40                         cout << "\nError: Stack is empty (nothing to pop).";
41                     else
42                         cout << "\nPopped element " << e;
43                     break;
44             case 'k':
45                 case 'K': // peek
46                     e = s.peek();
47                     if (s.wasError())
48                         cout << "\nError: Stack is empty (nothing to peek).";
49                     else
50                         cout << "\nPeeked element " << e;
51                     break;
52             case 'e':
53                 case 'E': // isEmpty
54                     if (s.isEmpty())
55                         cout << "\nStack is empty.";
56                     else
57                         cout << "\nStack contains elements.";
58                     break;
59             default:
60                 cout << "\nInvalid operation.";
61                 break;
62         }
63     } while (ch != 'q' && ch != 'Q');
64 }

```

10 Lab 10 Dynamic Memory Management

10.1 Aufgabe 1: Fixed-size Pool

Ihre Aufgabe ist es, einen Fixed-size Pool in Anlehnung an die in der Vorlesung gezeigte Variante zu implementieren. Sie müssen die Klassen `PoolAllocator`, `HeapException`, `HeapSizeMismatch` und `OutOfHeap` realisieren, wobei die letzten beiden Klassen Unterklassen von `HeapException` sind. Die Exceptionklassen müssen möglichst schlank implementiert werden (was heisst das?). Packen Sie alle Klassendeklarationen in das File `PoolAllocator.h`, definieren Sie sehr kurze Elementfunktionen implizit `inline`, alle weiteren separat, ebenfalls im File `PoolAllocator.h`. Alle Klassen müssen in den Namespace `dynamicMemory` gelegt werden. Nachfolgend finden Sie einen Ausschnitt aus der Deklaration der Klasse `dynamicMemory::PoolAllocator`.

```
1  template<std::size_t heapSize, std::size_t elemSize>
2  class PoolAllocator
3  {
4      public:
5          PoolAllocator(/* TODO: params (heap address */);
6          void* allocate(std::size_t bytes);
7          // TODO: implement this
8          // throw a HeapSizeMismatch Exception if 'bytes' doesn't match elemSize
9          // throw a OutOfHeap Exception if requested 'bytes' aren't available
10         void deallocate(/* TODO: params */) noexcept;
11         // TODO: implement this
12         // add this element to the freelist
13         // set the pointer to nullptr
14     private:
15         union Node
16         {
17             uint8_t data[elemSize]; // sizeof(data) should be >= sizeof(Node*)
18             Node* next;
19         };
20         Node* freeList;
21     };
22     template<std::size_t heapSize, std::size_t elemSize>
23     PoolAllocator<heapSize, elemSize>::PoolAllocator(uint8_t* heapAddr)
24     {
25         // TODO: implement this
26     }
```

Das Testprogramm finden Sie im Verzeichnis `./Vorgabe`. Der Output des Testprogramms soll wie folgt aussehen (Die Anfangsadresse kann unterschiedlich sein):

```
1  p[0] = 0x7ffe93438070
2  p[1] = 0x7ffe93438078
3  p[2] = 0x7ffe93438080
4  p[1] = 0
5  p[3] = 0x7ffe93438078
6  Heapsize mismatch exception occurred
```

1. Implementieren und testen Sie die Klassen vollständig. Erweitern Sie das Testprogramm so, dass auch eine `OutOfHeap`-Exception geprüft wird.
2. Erweitern Sie die Elementfunktion `dynamicMemory::PoolAllocator::allocate()` so, dass die Anzahl der angeforderten Bytes nicht mehr genau der Elementgrösse entsprechen muss, sondern kleiner oder gleich der Elementgrösse sein kann.

10.1.1 Lösung

1. siehe `./Loesung/FixedPool` Beachten Sie den Referenzparameter in `void deallocate(void*& ptr) noexcept` Dadurch ist es ohne Doppelpointer möglich, `ptr` auf `nullptr` zu setzen.

```
1  //=====
2  // Name      : FixedPool.cpp
3  // Author    : Reto Bonderer
4  // Version   : 02.06.2016
5  // Description : Fixed-size pool implementation
6  //=====
7
8  #include <iostream>
9  #include "PoolAllocator.h"
10 using namespace std;
11 using namespace dynamicMemory;
12
13 int main()
14 {
```

```
15     enum
16     {
17         testHeapSize = 200, testElemSize = 8,
18         nElems = testHeapSize / testElemSize,
19         nPtrs = 80
20     };
21     static uint8_t buffer[testHeapSize]; // our heap
22     void* p[nPtrs] = { nullptr }; // some void-pointers
23     PoolAllocator<testHeapSize, testElemSize> pool(&buffer[0])
24         ↪ ;
25
26     try
27     {
28         p[0] = pool.allocate(testElemSize);
```

```

28     cout << "p[0] = " << hex << p[0] << endl;
29     p[1] = pool.allocate(testElemSize);
30     cout << "p[1] = " << hex << p[1] << endl;
31     p[2] = pool.allocate(testElemSize);
32     cout << "p[2] = " << hex << p[2] << endl;
33     pool.deallocate(p[1]);
34     cout << "p[1] = " << hex << p[1] << endl;
35     p[3] = pool.allocate(testElemSize);
36     cout << "p[3] = " << hex << p[3] << endl; // should be
        ↳ former address of p[1]
37     p[4] = pool.allocate(testElemSize - 1);
38     cout << "p[4] = " << hex << p[4] << endl; // should
        ↳ throw HeapSizeMismatch exception
39 }
40 catch (const HeapSizeMismatch& e)
41 {
42     cout << "Heapsize mismatch exception occurred" << endl;
43 }
44 catch (const OutOfHeap& e)
45 {
46     cout << "Out of heap exception occurred" << endl;
47 }
48 catch (...)
49 {
50     cout << "Shouldn't get here" << endl;
51 }
52
53 try
54 {
55     for (int i = 5; i < nElems + 2; ++i) // there are 3
        ↳ objects so far => (nElems-3) available
56         p[i] = pool.allocate(testElemSize);
57     cout << dec << "p[" << nElems + 1 << "] = " << hex << p[
        ↳ nElems + 1] << endl;
58
59     cout << dec << "p[" << nElems + 2 << "] = " << hex << p[
        ↳ nElems + 2] << endl; // = 0
60     p[nElems + 3] = pool.allocate(testElemSize); // should
        ↳ throw OutOfHeap exception
61 }
62 catch (const HeapSizeMismatch& e)
63 {
64     cout << "Heapsize mismatch exception occurred" << endl;
65 }
66 catch (const OutOfHeap& e)
67 {
68     cout << "Out of heap exception occurred" << endl;
69 }
70 catch (...)
71 {
72     cout << "Shouldn't get here" << endl;
73 }
74
75 cout << "Deallocate all";
76 for (int i = 0; i < nPtrs; ++i)
77 {
78     try
79     {
80         pool.deallocate(p[i]);
81         cout << ".";
82     }
83     catch (...)
84     {
85         cout << "Shouldn't get here" << endl;
86     }
87 }
88 return 0;

```

```

1  /*
2  * PoolAllocator.h
3  *
4  * Created on: 27.04.2020
5  * Author: Reto Bonderer
6  */
7
8  #ifndef POOLALLOCATOR_H_
9  #define POOLALLOCATOR_H_
10
11  #include <stdint.h>
12
13  namespace dynamicMemory
14  {
15  class HeapException
16  {
17  };
18
19  class HeapSizeMismatch: public HeapException
20  {
21  };
22
23  class OutOfHeap: public HeapException
24  {
25  };
26
27  template<std::size_t heapSize, std::size_t elemSize>
28  class PoolAllocator
29  {
30  public:
31     PoolAllocator(uint8_t* heapAddr);
32     void* allocate(std::size_t bytes);
33     void deallocate(void*& ptr) noexcept; // void*& allows
        ↳ writing ptr
34 private:
35     union Node
36     {
37         uint8_t data[elemSize];
38         Node* next;
39     };
40     Node* freeList;
41 };
42
43 template<std::size_t heapSize, std::size_t elemSize>
44 PoolAllocator<heapSize, elemSize>::PoolAllocator(uint8_t*
        ↳ heapAddr) :
45     freeList(reinterpret_cast<Node*>(heapAddr))
46 {
47     const std::size_t nElems = heapSize / sizeof(Node);
48     for (std::size_t i = 0; i < nElems - 1; ++i) // link array
        ↳ elements together
49         freeList[i].next = &freeList[i + 1];
50     freeList[nElems - 1].next = nullptr;
51 }
52
53 template<std::size_t heapSize, std::size_t elemSize>
54 void* PoolAllocator<heapSize, elemSize>::allocate(std::
        ↳ size_t bytes)
55 {
56     if (bytes != elemSize)
57         throw HeapSizeMismatch();
58     if (freeList != nullptr)
59     {
60         void* pMem = freeList;
61         freeList = freeList->next;
62         return pMem;
63     }
64     else
65         throw OutOfHeap();
66 }
67
68 template<std::size_t heapSize, std::size_t elemSize>
69 void PoolAllocator<heapSize, elemSize>::deallocate(void*&
        ↳ ptr) noexcept
70 {
71     if (ptr == nullptr)
72         return;
73     Node* p = static_cast<Node*>(ptr);
74     p->next = freeList;
75     freeList = p;
76     ptr = nullptr; // increases safety
77 }
78
79 } /* namespace dynamicMemory */
80
81 #endif /* POOLALLOCATOR_H_ */

```

2. siehe ./Loesung/FixedPoolCeil Die einzige Änderung muss in der Elementfunktion void* allocate() vorgenommen werden. Der Operator != muss durch den >-Operator ersetzt werden: if (bytes > elemSize) // objects smaller than elemSize are allowed
throw HeapSizeMismatch();

```

1 //=====
2 // Name      : FixedPoolCeil.cpp
3 // Author    : Reto Bonderer
4 // Version   : 02.06.2016
5 // Description : Fixed-size pool with upper limit
6 //             ↳ implementation
7 //=====
8 #include <iostream>
9 #include "PoolAllocator.h"
10 using namespace std;
11 using namespace dynamicMemory;
12
13 int main()
14 {
15     enum
16     {
17         testHeapSize = 200, testElemSize = 8,
18         nElems = testHeapSize / testElemSize,
19         nPtrs = 80
20     };
21     static uint8_t buffer[testHeapSize]; // our heap
22     void* p[nPtrs] = { nullptr }; // some void-pointers
23
24     PoolAllocator<testHeapSize, testElemSize> pool(&buffer[0]);
25     ↳ ;
26     try
27     {
28         p[0] = pool.allocate(testElemSize);
29         cout << "p[0] = " << hex << p[0] << endl;
30         p[1] = pool.allocate(testElemSize);
31         cout << "p[1] = " << hex << p[1] << endl;
32         p[2] = pool.allocate(testElemSize - 1);
33         ↳ // should work
34         cout << "p[2] = " << hex << p[2] << endl;
35         pool.deallocate(p[1]);
36         cout << "p[1] = " << hex << p[1] << endl;
37         p[3] = pool.allocate(testElemSize);
38         cout << "p[3] = " << hex << p[3] << endl; // should be
39         ↳ former address of p[1]
40         p[4] = pool.allocate(testElemSize + 1);
41         cout << "p[4] = " << hex << p[4] << endl; // should
42         ↳ throw HeapSizeMismatch exception
43     }
44     catch (const HeapSizeMismatch& e)
45     {
46         cout << "Heapsize mismatch exception occurred" << endl;
47     }
48     catch (const OutOfHeap& e)
49     {
50         cout << "Out of heap exception occurred" << endl;
51     }
52     catch (...)
53     {
54         cout << "Shouldn't get here" << endl;
55     }
56
57     cout << "Deallocate all";
58     for (int i = 0; i < nPtrs; ++i)
59     {
60         try
61         {
62             pool.deallocate(p[i]);
63             cout << ".";
64         }
65         catch (...)
66         {
67             cout << "Shouldn't get here" << endl;
68         }
69     }
70     return 0;
71 }

```

```

1 /*
2  * PoolAllocator.h
3  *
4  * Created on: 27.04.2020
5  * Author: Reto Bonderer
6  */
7
8 #ifndef POOLALLOCATOR_H_
9 #define POOLALLOCATOR_H_
10
11 #include <stdint.h>
12
13 namespace dynamicMemory
14 {
15     class HeapException
16     {
17     };
18
19     class HeapSizeMismatch: public HeapException
20     {
21     };
22
23     class OutOfHeap: public HeapException
24     {
25     };
26
27     template<std::size_t heapSize, std::size_t elemSize>
28     class PoolAllocator
29     {
30     public:
31         PoolAllocator(uint8_t* heapAddr);
32         void* allocate(std::size_t bytes);
33         void deallocate(void*& ptr) noexcept; // void*& allows
34         ↳ writing ptr
35     private:
36         union Node
37         {
38             uint8_t data[elemSize];
39             Node* next;
40         };
41         Node* freeList;
42
43     template<std::size_t heapSize, std::size_t elemSize>
44     PoolAllocator<heapSize, elemSize>::PoolAllocator(uint8_t*
45     ↳ heapAddr) :
46         freeList(reinterpret_cast<Node*>(heapAddr))
47     {
48         const std::size_t nElems = heapSize / sizeof(Node);
49         for (std::size_t i = 0; i < nElems - 1; ++i) // link array
50             ↳ elements together
51             freeList[i].next = &freeList[i + 1];
52         freeList[nElems - 1].next = nullptr;
53     }
54
55     template<std::size_t heapSize, std::size_t elemSize>
56     void* PoolAllocator<heapSize, elemSize>::allocate(std::
57     ↳ size_t bytes)
58     {
59         if (bytes > elemSize)
60             throw HeapSizeMismatch();
61         if (freeList != nullptr)
62         {
63             void* pMem = freeList;
64             freeList = freeList->next;
65             return pMem;
66         }
67         else
68             throw OutOfHeap();
69     }
70
71     template<std::size_t heapSize, std::size_t elemSize>
72     void PoolAllocator<heapSize, elemSize>::deallocate(void*&
73     ↳ ptr) noexcept

```

```

70 {
71     if (ptr == nullptr)
72         return;
73     Node* p = static_cast<Node*>(ptr);
74     p->next = freeList;
75     freeList = p;
76     ptr = nullptr; // increases safety
77 }
78
79 } /* namespace dynamicMemory */
80
81 #endif /* POOLALLOCATOR_H_ */

```

10.2 Aufgabe 2: Block Allocator

Implementieren Sie nun einen Block Allocator. Die Klasse `BlockAllocator` soll aus 4 Fixed-size Pools gemäss Aufgabe 1b) bestehen. Die einzelnen Elementgrössen können der Klasse `BlockAllocator` mittels Templateparameter mitgegeben werden. Der Block Allocator nimmt jeweils den kleinstmöglichen Pool, um die Anforderungen zu erfüllen. Das bedeutet auch, dass auf den nächstgrösseren Pool zugegriffen wird, falls der kleinere Pool voll ist.

```

1 //=====
2 // Name      : BlockAllocation.cpp
3 // Author    : Reto Bonderer
4 // Version   : 02.06.2016
5 // Description : Block allocation with 4 fixed-size pools
6 //=====
7
8 #include <iostream>
9 #include "BlockAllocator.h"
10 using namespace std;
11 using namespace dynamicMemory;
12
13 int main()
14 {
15     enum
16     {
17         testPool1Size = 200, testElem1Size = 4,
18         testPool2Size = 200, testElem2Size = 8,
19         testPool3Size = 512, testElem3Size = 32,
20         testPool4Size = 1024, testElem4Size = 128,
21         nPtrs = 20
22     };
23     static uint8_t pool1Heap[testPool1Size];
24     static uint8_t pool2Heap[testPool2Size];
25     static uint8_t pool3Heap[testPool3Size];
26     static uint8_t pool4Heap[testPool4Size];
27
28     BlockAllocator<testPool1Size, testElem1Size,
29                   testPool2Size, testElem2Size,
30                   testPool3Size, testElem3Size,
31                   testPool4Size, testElem4Size>
32         block(&pool1Heap[0], &pool2Heap[0],
33             &pool3Heap[0], &pool4Heap[0]);
34
35     void* p[nPtrs] = { nullptr }; // some void-pointers
36     cout << "Memory Map" << endl;
37     cout << "Pool 1: " << hex << (void*)pool1Heap << "-"
38         << (void*)(pool1Heap + testPool1Size - 1) << " size: "
39         << testPool1Size << endl;
40     cout << "Pool 2: " << hex << (void*)pool2Heap << "-"
41         << (void*)(pool2Heap + testPool2Size - 1) << " size: "
42         << testPool2Size << endl;
43     cout << "Pool 3: " << hex << (void*)pool3Heap << "-"
44         << (void*)(pool3Heap + testPool3Size - 1) << " size: "
45         << testPool3Size << endl;
46     cout << "Pool 4: " << hex << (void*)pool4Heap << "-"
47         << (void*)(pool4Heap + testPool4Size - 1) << " size: "
48
49         << testPool4Size << endl;
50
51     try // ... some allocations
52     {
53         p[0] = block.allocate(4);
54         cout << "p[0] = " << hex << p[0] << endl;
55         p[1] = block.allocate(4);
56         cout << "p[1] = " << hex << p[1] << endl;
57         p[2] = block.allocate(3); // should work
58         cout << "p[2] = " << hex << p[2] << endl;
59         block.deallocate(p[1]);
60         cout << "p[1] = " << hex << p[1] << endl;
61         p[3] = block.allocate(4);
62         cout << "p[3] = " << hex << p[3] << endl; // should be
63             << former address of p[1]
64         p[4] = block.allocate(128);
65         cout << "p[4] = " << hex << p[4] << endl;
66         p[5] = block.allocate(140);
67         cout << "p[5] = " << hex << p[5] << endl; // should throw
68             << HeapSizeMismatch exception
69         p[6] = block.allocate(123);
70         cout << "p[6] = " << hex << p[6] << endl;
71     }
72     catch (const HeapSizeMismatch& e)
73     {
74         cout << "Heapsize mismatch exception occurred" << endl;
75     }
76     catch (const OutOfHeap& e)
77     {
78         cout << "Out of heap exception occurred" << endl;
79     }
80     catch (...)
81     {
82         cout << "Shouldn't get here" << endl;
83     }
84
85     cout << "Deallocate all";
86     for (int i = 0; i < nPtrs; ++i)
87     {
88         try
89         {
90             block.deallocate(p[i]);
91             cout << ".";
92         }
93         catch (...)
94         {
95             cout << "Shouldn't get here" << endl;
96         }
97     }
98     return 0;
99 }

```

```

1 /*
2  * PoolAllocator.h
3  *
4  * Created on: 27.04.2020
5  * Author: Reto Bonderer
6  */
7
8 #ifndef POOLALLOCATOR_H_
9 #define POOLALLOCATOR_H_
10
11 #include <stdint.h>
12
13 namespace dynamicMemory
14 {
15     class HeapException
16     {
17     };
18
19     class HeapSizeMismatch: public HeapException
20     {
21     };
22
23     class OutOfHeap: public HeapException
24     {
25     };
26
27     template<std::size_t heapSize, std::size_t elemSize>
28     class PoolAllocator
29     {
30     public:

```

```

31     PoolAllocator(uint8_t* heapAddr);
32     void* allocate(std::size_t bytes);
33     void deallocate(void*& ptr) noexcept; // void*& allows
      ↳ writing ptr
34 private:
35     union Node
36     {
37         uint8_t data[elemSize];
38         Node* next;
39     };
40     Node* freeList;
41 };
42
43 template<std::size_t heapSize, std::size_t elemSize>
44 PoolAllocator<heapSize, elemSize>::PoolAllocator(uint8_t*
      ↳ heapAddr) :
45     freeList(reinterpret_cast<Node*>(heapAddr))
46 {
47     const std::size_t nElems = heapSize / sizeof(Node);
48     for (std::size_t i = 0; i < nElems - 1; ++i) // link array
      ↳ elements together
49         freeList[i].next = &freeList[i + 1];
50     freeList[nElems - 1].next = nullptr;
51 }
52
53 template<std::size_t heapSize, std::size_t elemSize>
54 void* PoolAllocator<heapSize, elemSize>::allocate(std::size_t
      ↳ bytes)
55 {
56     if (bytes > elemSize)
57         throw HeapSizeMismatch();
58     if (freeList != nullptr)
59     {
60         void* pMem = freeList;
61         freeList = freeList->next;
62         return pMem;
63     }
64     else
65         throw OutOfHeap();
66 }
67
68 template<std::size_t heapSize, std::size_t elemSize>
69 void PoolAllocator<heapSize, elemSize>::deallocate(void*& ptr)
      ↳ noexcept
70 {
71     if (ptr == nullptr)
72         return;
73     Node* p = static_cast<Node*>(ptr);
74     p->next = freeList;
75     freeList = p;
76     ptr = nullptr; // increases safety
77 }
78
79 } /* namespace dynamicMemory */
80
81 #endif /* POOLALLOCATOR_H_ */

```

```

1  /*
2  * BlockAllocator.h
3  *
4  * Created on: 27.04.2020
5  * Author: Reto Bonderer
6  */
7
8 #ifndef BLOCKALLOCATOR_H_
9 #define BLOCKALLOCATOR_H_
10
11 #include "PoolAllocator.h"
12
13 namespace dynamicMemory
14 {
15
16 template<std::size_t pool1Size, std::size_t elem1Size, //
      ↳ elem1Size <= elem2Size <= ...
17     std::size_t pool2Size, std::size_t elem2Size, //
      ↳ elem3Size <= elem4Size!!
18     std::size_t pool3Size, std::size_t elem3Size,
19     std::size_t pool4Size, std::size_t elem4Size>
20 class BlockAllocator
21 {
22 public:
23     BlockAllocator(uint8_t* pool1Heap,
24                   uint8_t* pool2Heap,
25                   uint8_t* pool3Heap,
26                   uint8_t* pool4Heap) :
27         pool1(pool1Heap),
28         pool2(pool2Heap),
29         pool3(pool3Heap),
30         pool4(pool4Heap),
31         pool1Addr(pool1Heap),
32         pool2Addr(pool2Heap),
33         pool3Addr(pool3Heap),
34         pool4Addr(pool4Heap)
35     {
36     }
37     void* allocate(std::size_t bytes);
38     void deallocate(void*& ptr) noexcept;
39
40 private:
41     PoolAllocator<pool1Size, elem1Size> pool1;
42     PoolAllocator<pool2Size, elem2Size> pool2;
43     PoolAllocator<pool3Size, elem3Size> pool3;
44     PoolAllocator<pool4Size, elem4Size> pool4;
45     uint8_t* pool1Addr;
46     uint8_t* pool2Addr;
47     uint8_t* pool3Addr;
48     uint8_t* pool4Addr;
49 };
50
51 template<std::size_t pool1Size, std::size_t elem1Size,
52     std::size_t pool2Size, std::size_t elem2Size,
53     std::size_t pool3Size, std::size_t elem3Size,
54     std::size_t pool4Size, std::size_t elem4Size>
55 void* BlockAllocator<pool1Size, elem1Size, pool2Size,
      ↳ elem2Size,
56     pool3Size, elem3Size, pool4Size,
      ↳ elem4Size>::allocate(std::size_t
      ↳ bytes)
57 {
58     void* p;
59     try
60     {
61         p = pool1.allocate(bytes);
62     }
63     catch (const HeapException& e)
64     {
65         try
66         {
67             p = pool2.allocate(bytes);
68         }
69         catch (const HeapException& e)
70         {
71             try
72             {
73                 p = pool3.allocate(bytes);
74             }
75             catch (const HeapException& e)
76             {
77                 p = pool4.allocate(bytes);
78             }
79         }
80     }
81     return p;
82 }
83
84 template<std::size_t pool1Size, std::size_t elem1Size,
85     std::size_t pool2Size, std::size_t elem2Size,
86     std::size_t pool3Size, std::size_t elem3Size,
87     std::size_t pool4Size, std::size_t elem4Size>
88 void BlockAllocator<pool1Size, elem1Size, pool2Size, elem2Size
      ↳ ,
89     pool3Size, elem3Size, pool4Size, elem4Size
      ↳ >::deallocate(void*& ptr) noexcept
90 {
91     if (ptr == nullptr)
92         return;
93
94     if (pool1Addr <= ptr && ptr < pool1Addr + pool1Size)
95         pool1.deallocate(ptr);
96     else if (pool2Addr <= ptr && ptr < pool2Addr + pool2Size)
97         pool2.deallocate(ptr);
98     else if (pool3Addr <= ptr && ptr < pool3Addr + pool3Size)
99         pool3.deallocate(ptr);
100     else if (pool4Addr <= ptr && ptr < pool4Addr + pool4Size)
101         pool4.deallocate(ptr);
102 }
103
104 } /* namespace dynamicMemory */
105 #endif /* BLOCKALLOCATOR_H_ */

```


11 Lab 11 C++ and ROMability, Hardware Abstraction Layer (HAL)

11.1 Aufgabe 1: C++ and ROMability, Hardware Abstraction Layer (HAL)

In der Vorlesung haben wir gesehen, dass unterschiedliche Konstrukte ROMable sind, jedoch nicht jeder Compiler diese Konstrukte auch im ROM platziert. In dieser Aufgabe untersuchen Sie die ROMability und die Optimierung von ROMable Konstrukten des g++-Compilers. Für die Untersuchung müssen Sie unter Umständen Optimierungsstufen setzen.

Die folgenden Optionen des GNU-Compilers können für diese Aufgabe nützlich sein:

- E Precompile only, der Output wird auf stdout geschrieben
- S Assembleroutput, ohne Objectfile erzeugen
- c nur compilieren
- O0 keine Optimierung
- O1 Optimierungsstufe 1 (siehe g++ -help oder man g++ für Details)
- O2 Optimierungsstufe 2
- O3 Optimierungsstufe 3
- Os Optimierung auf Codegrösse

Achten Sie bei allen Untersuchungen darauf, dass Ihre kleinen Testprogramme nicht vollständig wegoptimiert werden, da die definierten Variablen nicht weiterverwendet werden. Damit das nicht passiert, können Sie den Inhalt der Variablen in die Console schreiben.

1. Umsetzung von Strings: Untersuchen Sie, wie die untenstehenden Stringdefinitionen umgesetzt werden. Wird *World* mit *Hello World* gemeinsam verwendet? Gibt es allenfalls Unterschiede in Abhängigkeit der Optimierungsstufen?

```
const char* pc1 = "Hello World";
const char* const pc2 = "World";
```

2. Wie werden Tabellen umgesetzt? Gibt es einen Unterschied in der Umsetzung der folgenden beiden Definitionen? Was passiert, wenn Sie beiden Definitionen das Schlüsselwort static voranstellen?

```
const int table1[] = 1, 2, 3;
int table2[] = 1, 2, 3;
```

3. Integerkonstanten: Für die Definition von Integerkonstanten stehen Ihnen drei Möglichkeiten zur Verfügung: const int, enum und #define. Wie werden diese Varianten umgesetzt? Geben Sie in Ihrem Testprogramm dem Compiler die Chance, eine Immediate-Adressierung zu verwenden. Um herauszufinden, ob eine Konstante mehrfach im Speicher angelegt wird (ohne Immediate-Adressierung), müssen Sie die definierte Konstante mehrfach im Programm verwenden.

4. Floating Point-Konstanten: Für die Definition von Floating Point-Konstanten stehen Ihnen const double und #define zur Verfügung. Wie werden diese Varianten umgesetzt? Eine Immediate-Adressierung ist mit doubles kaum möglich. Um herauszufinden, ob eine Konstante mehrfach im Speicher angelegt wird, müssen Sie die definierte Konstante mehrfach im Programm verwenden.

5. Können Sie herausfinden, ob vtbl's im ROM abgelegt werden?

11.1.1 Lösung

1. siehe ./Loesung/A1-0 Solange die Strings völlig identisch sind (Formatstring), wird der String nur einmal im Speicher abgelegt. Bei Teilstrings werden bei jeder Optimierungsstufe beide Strings vollständig gespeichert (doppelt). Wenn nur der String ausgegeben wird, ersetzt der Compiler printf() durch puts(). Bei dieser speziellen Anwendung spart man sich dadurch den Formatstring (hier ist g++ smart).

```

1 #include <stdio>
2 int main(void)
3 {
4     const char* pc1 = "Hello World";
5     const char* pc2 = "World";

```

```

6     printf("%s\n", pc1);
7     printf("%s\n", pc2);
8     return 0;
9 }

```

```

1 .file "main.cpp"
2 .text
3 .section .rodata
4 .LC0:
5     .string "Hello World"
6 .LC1:
7     .string "World"
8 .text
9 .globl main
10 .type main, @function
11 main:
12 .LFB0:
13     .cfi_startproc
14     pushq %rbp
15     .cfi_def_cfa_offset 16
16     .cfi_offset 6, -16
17     movq %rsp, %rbp
18     .cfi_def_cfa_register 6
19     subq $16, %rsp

```

```

20     leaq .LC0(%rip), %rax
21     movq %rax, -16(%rbp)
22     leaq .LC1(%rip), %rax
23     movq %rax, -8(%rbp)
24     movq -16(%rbp), %rax
25     movq %rax, %rdi
26     call puts@PLT
27     leaq .LC1(%rip), %rdi
28     call puts@PLT
29     movl $0, %eax
30     leave
31     .cfi_def_cfa 7, 8
32     ret
33 .cfi_endproc
34 .LFE0:
35     .size main, .-main
36     .ident "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
37     .section .note.GNU-stack,"",@progbits

```

2. siehe ./Loesung/A1-1 Solange von beiden Tabellen nur gelesen wird, könnten beide Tabellen geROMt werden, bzw. sogar nur einfach gespeichert werden, da der Inhalt identisch ist. Diese kurzen Tabellen werden sogar auf dem Stack abgespeichert. Sobald sie grösser sind, werden sie ebenfalls im Datenbereich gespeichert. Interessant sind vor allem noch die folgenden Deklarationen mit static. static bewirkt, dass die Tabellen auf jeden Fall im Datenbereich gespeichert werden, d.h. keinesfalls mehr auf dem Stack. const bewirkt, dass die Tabellen in den read-only Bereich gehen.

static const int table1[] = 1, 2, 3; static int table2[] = 1, 2, 3;

```

1 #include <stdio>
2 int main(void)
3 {
4     int table0[] = {1, 2, 3};
5     const int table1[] = {1, 2, 3};
6     static const int table2[] = {1, 2, 3};
7     static int table3[] = {1, 2, 3};
8
9     for(int i=0; i<3; ++i)

```

```

10 {
11     printf("%d\n", table0[i]);
12     printf("%d\n", table1[2-i]);
13     printf("%d\n", table2[i+2]);
14     printf("%d\n", table3[2-i]);
15 }
16 return 0;
17 }

```

```

1 .file "main.cpp"
2 .text
3 .section .rodata.str1.1,"aMS",@progbits,1
4 .LC0:
5     .string "%d\n"
6 .section .text.startup,"ax",@progbits
7 .p2align 4,,15
8 .globl main
9 .type main, @function
10 main:
11 .LFB30:
12     .cfi_startproc
13     pushq %r15
14     .cfi_def_cfa_offset 16
15     .cfi_offset 15, -16
16     pushq %r14
17     .cfi_def_cfa_offset 24
18     .cfi_offset 14, -24
19     leaq _ZZ4mainE6table2(%rip), %r14
20     pushq %r13
21     .cfi_def_cfa_offset 32
22     .cfi_offset 13, -32
23     pushq %r12
24     .cfi_def_cfa_offset 40
25     .cfi_offset 12, -40
26     leaq _ZZ4mainE6table3(%rip), %r13
27     pushq %rbp

```

```

28     .cfi_def_cfa_offset 48
29     .cfi_offset 6, -48
30     pushq %rbx
31     .cfi_def_cfa_offset 56
32     .cfi_offset 3, -56
33     leaq .LC0(%rip), %rbp
34     movl $8, %ebx
35     xorl %r12d, %r12d
36     subq $56, %rsp
37     .cfi_def_cfa_offset 112
38     movq %fs:40, %rax
39     movq %rax, 40(%rsp)
40     xorl %eax, %eax
41     leaq 16(%rsp), %rcx
42     movabsq $8589934593, %rax
43     leaq 28(%rsp), %r15
44     movq %rax, 16(%rsp)
45     movl $3, 24(%rsp)
46     movq %rax, 28(%rsp)
47     movl $3, 36(%rsp)
48 .L2:
49     movl (%rcx,%r12), %edx
50     movq %rbp, %rsi
51     movl $1, %edi
52     xorl %eax, %eax
53     movq %rcx, 8(%rsp)
54     call __printf_chk@PLT

```

```

55     movl    (%r15,%rbx), %edx
56     movq    %rbp, %rsi
57     movl    $1, %edi
58     xorl    %eax, %eax
59     call    __printf_chk@PLT
60     movl    (%r14,%r12), %eax
61     movq    %rbp, %rsi
62     movl    $1, %edi
63     addq    $4, %r12
64     leal    2(%rax), %edx
65     xorl    %eax, %eax
66     call    __printf_chk@PLT
67     movl    0(%r13,%rbx), %edx
68     xorl    %eax, %eax
69     movq    %rbp, %rsi
70     movl    $1, %edi
71     subq    $4, %rbx
72     call    __printf_chk@PLT
73     cmpq    $-4, %rbx
74     movq    8(%rsp), %rcx
75     jne     .L2
76     xorl    %eax, %eax
77     movq    40(%rsp), %rcx
78     xorq    %fs:40, %rcx
79     jne     .L7
80     addq    $56, %rsp
81     .cfi_restore_state
82     .cfi_def_cfa_offset 56
83     popq    %rbx

```

```

84     .cfi_def_cfa_offset 48
85     popq    %rbp
86     .cfi_def_cfa_offset 40
87     popq    %r12
88     .cfi_def_cfa_offset 32
89     popq    %r13
90     .cfi_def_cfa_offset 24
91     popq    %r14
92     .cfi_def_cfa_offset 16
93     popq    %r15
94     .cfi_def_cfa_offset 8
95     ret
96
.L7:
97     .cfi_restore_state
98     call    __stack_chk_fail@PLT
99     .cfi_endproc
100
.LFE30:
101     .size    main, .-main
102     .section .rodata
103     .align 8
104     .type    _ZZ4mainE6table2, @object
105     .size    _ZZ4mainE6table2, 12
106 _ZZ4mainE6table2:
107     .long    1
108     .long    2
109     .long    3
110     .set     _ZZ4mainE6table3, _ZZ4mainE6table2
111     .ident   "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
112     .section .note.GNU-stack,"",@progbits

```

3. siehe ./Loesung/A1-2 Integerkonstanten, die mit #define oder enum definiert sind, jedoch nicht gebraucht werden, ergeben weder Code noch Daten. Integerkonstanten, die mit const definiert sind, werden auch dann gespeichert, wenn sie gar nicht verwendet werden, damit die Konstanten vorhanden sind, falls eine andere Compilationseinheit diese Konstanten verwenden würde. Innerhalb des .cpp-Files der Konstantendefinition werden diese Konstanten ebenfalls mittels Immediate-Adressierung verwendet. Die beste Variante, um Integerkonstanten zu definieren, ist eindeutig mittels enum. #defines haben die bekannten Makroprobleme, consts brauchen unnötigerweise Speicher.

```

1  #include <stdio>
2  #include "ints.h"
3
4  const int const1 = 64;
5  const int const2 = 128;
6  const int const3 = 256;
7
8  int main(void)
9  {
10     int tmp1;
11     int tmp2 = 20;
12     tmp1 = MAKRO_DEF1;
13     tmp2 = tmp2 + MAKRO_DEF1;
14     printf("Mit Makros: tmp1 = %d, tmp2 = %d\n", tmp1, tmp2);
15
16     tmp2 = 20;

```

```

17     tmp1 = en1;
18     tmp2 = tmp2 + en1;
19     printf("Mit Enums: tmp1 = %d, tmp2 = %d\n", tmp1, tmp2);
20
21     tmp2 = 20;
22     tmp1 = const3;
23     tmp2 = tmp2 + const3;
24     printf("Mit Const: tmp1 = %d, tmp2 = %d\n", tmp1, tmp2);
25
26     tmp2 = foo(tmp1);
27     printf("Unterprogramm: tmp1 = %d, tmp2 = %d\n", tmp1, tmp2
28         ↪ );
29     return 0;
30 }

```

```

1  .file     "main.cpp"
2  .text
3  .globl   const1
4  .section .rodata
5  .align 4
6  .type    const1, @object
7  .size    const1, 4
8  const1:
9  .long    64
10 .globl   const2
11 .align 4
12 .type    const2, @object
13 .size    const2, 4
14 const2:
15 .long    128
16 .globl   const3
17 .align 4
18 .type    const3, @object
19 .size    const3, 4
20 const3:
21 .long    256

```

```

22 .align 8
23
.LC0:
24 .string  "Mit Makros: tmp1 = %d, tmp2 = %d\n"
25 .align 8
26
.LC1:
27 .string  "Mit Enums: tmp1 = %d, tmp2 = %d\n"
28 .align 8
29
.LC2:
30 .string  "Mit Const: tmp1 = %d, tmp2 = %d\n"
31 .align 8
32
.LC3:
33 .string  "Unterprogramm: tmp1 = %d, tmp2 = %d\n"
34 .text
35 .globl   main
36 .type    main, @function
37 main:
38 .LFB0:
39 .cfi_startproc
40 pushq    %rbp
41 .cfi_def_cfa_offset 16
42 .cfi_offset 6, -16

```

```

43     movq    %rsp, %rbp
44     .cfi_def_cfa_register 6
45     subq    $16, %rsp
46     movl    $20, -8(%rbp)
47     movl    $1, -4(%rbp)
48     addl    $1, -8(%rbp)
49     movl    -8(%rbp), %edx
50     movl    -4(%rbp), %eax
51     movl    %eax, %esi
52     leaq    .LC0(%rip), %rdi
53     movl    $0, %eax
54     call    printf@PLT
55     movl    $20, -8(%rbp)
56     movl    $8, -4(%rbp)
57     addl    $8, -8(%rbp)
58     movl    -8(%rbp), %edx
59     movl    -4(%rbp), %eax
60     movl    %eax, %esi
61     leaq    .LC1(%rip), %rdi
62     movl    $0, %eax
63     call    printf@PLT
64     movl    $20, -8(%rbp)
65     movl    $256, -4(%rbp)
66     addl    $256, -8(%rbp)
67     movl    -8(%rbp), %edx

```

```

68     movl    -4(%rbp), %eax
69     movl    %eax, %esi
70     leaq    .LC2(%rip), %rdi
71     movl    $0, %eax
72     call    printf@PLT
73     movl    -4(%rbp), %eax
74     movl    %eax, %edi
75     call    _Z3fooi@PLT
76     movl    %eax, -8(%rbp)
77     movl    -8(%rbp), %edx
78     movl    -4(%rbp), %eax
79     movl    %eax, %esi
80     leaq    .LC3(%rip), %rdi
81     movl    $0, %eax
82     call    printf@PLT
83     movl    $0, %eax
84     leave
85     .cfi_def_cfa 7, 8
86     ret
87     .cfi_endproc
88 .LFE0:
89     .size    main, .-main
90     .ident   "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
91     .section .note.GNU-stack,"",@progbits

```

```

1  #define MAKRO_DEF1 1
2  #define MAKRO_DEF2 2
3  #define MAKRO_DEF3 4
4
5  enum
6  {
7      en1 = 8,
8      en2 = 16,

```

```

9      en3 = 32
10 };
11
12 extern const int const1;
13 extern const int const2;
14 extern const int const3;
15
16 int foo(int arg);

```

```

1  #include "ints.h"
2
3  int foo(int arg)

```

```

4  {
5      return arg + MAKRO_DEF1 + en1 + const3;
6  }

```

```

1  .file      "subs.cpp"
2  .text
3  .globl     _Z3fooi
4  .type      _Z3fooi, @function
5  _Z3fooi:
6  .LFB0:
7
8      .cfi_startproc
9      pushq   %rbp
10     .cfi_def_cfa_offset 16
11     .cfi_offset 6, -16
12     movq    %rsp, %rbp
13     .cfi_def_cfa_register 6
14     movl    %edi, -4(%rbp)

```

```

14     movl    -4(%rbp), %eax
15     leal    9(%rax), %edx
16     movl    const3(%rip), %eax
17     addl    %edx, %eax
18     popq    %rbp
19     .cfi_def_cfa 7, 8
20     ret
21     .cfi_endproc
22 .LFE0:
23     .size    _Z3fooi, .-_Z3fooi
24     .ident   "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
25     .section .note.GNU-stack,"",@progbits

```

4. siehe ./Loesung/A1-3 Gleitpunktkonstanten können normalerweise (speziell bei Mikrocontrollern) nicht mittels Immediate-Adressierung verwendet werden, d.h. es ist immer eine Konstante im Datenbereich vorhanden. Bei 64 Bit-Systemen (z.B. unsere Laborrechner) kann ein Register einen ganzen double-Wert (8 Byte) beinhalten, die Instruktion MOVABSQ (Move quad word to register) lädt einen double-Wert mittels Immediateadressierung direkt in ein solches Register. Konstanten, die mit #define definiert sind, jedoch nicht gebraucht werden, ergeben weder Code noch Daten. Gleitpunktkonstanten, die mit const definiert sind, werden auch dann gespeichert, wenn sie gar nicht verwendet werden, damit die Konstanten vorhanden sind, falls eine andere Compilationseinheit diese Konstanten verwenden würde. Allerdings sind diese Konstanten immer nur einfach vorhanden. Konstanten, die mit #define definiert wurden, sind in jeder Compilationseinheit separat, d.h. mehrfach vorhanden. Die beste Variante, um Gleitpunktkonstanten zu definieren, ist nicht so eindeutig festzulegen. #defines haben die bekannten Makroprobleme und benötigen allenfalls mehrfach Speicher für denselben Wert, jedoch nur, wenn sie wirklich verwendet werden. consts brauchen leider auch Speicher, wenn

sie gar nicht verwendet werden, allerdings nur einmal. Solange nicht zu viele consts auf Vorrat definiert werden, schlage ich vor, auch hier auf `#defines` zu verzichten und `consts` zu verwenden. Gute Compiler/ Linker mit Link-Time Optimization (LTO) sollten nicht verwendete Konstanten entfernen.

5. siehe ./Loesung/A1-4

```

1  #include <stdio>
2  #include "floats.h"
3
4  const double const1 = 432.22128;
5  const double const2 = 23.55128;
6  const double const3 = 256.56;
7
8  int main(void)
9  {
10     double tmp1;
11     double tmp2 = 2.5;
12     tmp1 = MAKRO_DEF1;
13     tmp2 = tmp2 + MAKRO_DEF1;
14
15     printf("Mit Makros: tmp1 = %f, tmp2 = %f\n", tmp1, tmp2);
16
17     tmp2 = 2.5;
18     tmp1 = const3;
19     tmp2 = tmp2 + const3;
20     printf("Mit Const: tmp1 = %f, tmp2 = %f\n", tmp1, tmp2);
21
22     tmp2 = foo(tmp1);
23     printf("Unterprogramm: tmp1 = %f, tmp2 = %f\n", tmp1, tmp2
24           ↪ );
25     return 0;
26 }

```

```

1  .file "main.cpp"
2  .text
3  .globl const1
4  .section .rodata
5  .align 8
6  .type const1, @object
7  .size const1, 8
8  const1:
9  .long 1558557732
10 .long 1081803658
11 .globl const2
12 .align 8
13 .type const2, @object
14 .size const2, 8
15 const2:
16 .long 2946691162
17 .long 1077382432
18 .globl const3
19 .align 8
20 .type const3, @object
21 .size const3, 8
22 const3:
23 .long 3264175145
24 .long 1081084149
25 .align 8
26 .LC2:
27 .string "Mit Makros: tmp1 = %f, tmp2 = %f\n"
28 .align 8
29 .LC4:
30 .string "Mit Const: tmp1 = %f, tmp2 = %f\n"
31 .align 8
32 .LC5:
33 .string "Unterprogramm: tmp1 = %f, tmp2 = %f\n"
34 .text
35 .globl main
36 .type main, @function
37 main:
38 .LFB0:
39 .cfi_startproc
40 pushq %rbp
41 .cfi_def_cfa_offset 16
42 .cfi_offset 6, -16
43 movq %rsp, %rbp
44 .cfi_def_cfa_register 6
45 subq $32, %rsp
46 movsd .LC0(%rip), %xmm0
47 movsd %xmm0, -16(%rbp)
48 movsd .LC1(%rip), %xmm0
49 movsd %xmm0, -8(%rbp)
50 movsd -16(%rbp), %xmm1
51 movsd .LC1(%rip), %xmm0
52 addsd %xmm1, %xmm0
53 movsd %xmm0, -16(%rbp)
54 movsd -16(%rbp), %xmm0
55 movq -8(%rbp), %rax
56 movapd %xmm0, %xmm1
57 movq %rax, -24(%rbp)
58 movsd -24(%rbp), %xmm0
59 leaq .LC2(%rip), %rdi
60 movl $2, %eax
61 call printf@PLT
62 movsd .LC0(%rip), %xmm0
63 movsd %xmm0, -16(%rbp)
64 movsd .LC3(%rip), %xmm0
65 movsd %xmm0, -8(%rbp)
66 movsd -16(%rbp), %xmm1
67 movsd .LC3(%rip), %xmm0
68 addsd %xmm1, %xmm0
69 movsd %xmm0, -16(%rbp)
70 movsd -16(%rbp), %xmm0
71 movq -8(%rbp), %rax
72 movapd %xmm0, %xmm1
73 movq %rax, -24(%rbp)
74 movsd -24(%rbp), %xmm0
75 leaq .LC4(%rip), %rdi
76 movl $2, %eax
77 call printf@PLT
78 movq -8(%rbp), %rax
79 movq %rax, -24(%rbp)
80 movsd -24(%rbp), %xmm0
81 call _Z3food@PLT
82 movq %xmm0, %rax
83 movq %rax, -16(%rbp)
84 movsd -16(%rbp), %xmm0
85 movq -8(%rbp), %rax
86 movapd %xmm0, %xmm1
87 movq %rax, -24(%rbp)
88 movsd -24(%rbp), %xmm0
89 leaq .LC5(%rip), %rdi
90 movl $2, %eax
91 call printf@PLT
92 movl $0, %eax
93 leave
94 .cfi_def_cfa 7, 8
95 ret
96 .cfi_endproc
97 .LFE0:
98 .size main, .-main
99 .section .rodata
100 .align 8
101 .LC0:
102 .long 0
103 .long 1074003968
104 .align 8
105 .LC1:
106 .long 1271310320
107 .long 1074039095
108 .align 8
109 .LC3:
110 .long 3264175145
111 .long 1081084149
112 .ident "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
113 .section .note.GNU-stack,"",@progbits

```

```

1 #define MAKRO_DEF1 2.567
2 #define MAKRO_DEF2 234.6532
3 #define MAKRO_DEF3 4345.634
4
5 extern const double const1;

```

```

6 extern const double const2;
7 extern const double const3;
8
9 double foo(double arg);

```

```

1 #include "floats.h"
2
3 double foo(double arg)

```

```

4 {
5     return arg + MAKRO_DEF1 + const3;
6 }

```

```

1 .file "subs.cpp"
2 .text
3 .globl _Z3food
4 .type _Z3food, @function
5 _Z3food:
6 .LFB0:
7     .cfi_startproc
8     pushq %rbp
9     .cfi_def_cfa_offset 16
10    .cfi_offset 6, -16
11    movq %rsp, %rbp
12    .cfi_def_cfa_register 6
13    movsd %xmm0, -8(%rbp)
14    movsd -8(%rbp), %xmm1
15    movsd .LC0(%rip), %xmm0
16    addsd %xmm1, %xmm0

```

```

17    movsd const3(%rip), %xmm1
18    addsd %xmm1, %xmm0
19    popq %rbp
20    .cfi_def_cfa 7, 8
21    ret
22    .cfi_endproc
23 .LFE0:
24     .size _Z3food, .-_Z3food
25     .section .rodata
26     .align 8
27 .LC0:
28     .long 1271310320
29     .long 1074039095
30     .ident "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
31     .section .note.GNU-stack,"",@progbits

```

Die vtbl's werden in den Abschnitt rodata, d.h. read-only data gelegt (_ZTV1A und _ZTV1B). Bei einem Embedded System mit ROM könnte dieser Bereich ins ROM gelegt werden.

11.2 Aufgabe 2: Ohne Hardware Abstraction Layer

Im Vorgabeordner finden Sie eine Beispielapplikation für den EswRobot. Die Applikation initialisiert den Roboter und lässt LED 1 leuchten solange Switch 1 gedrückt wird und lässt LED 2 leuchten solange Switch 2 gedrückt wird. Die komplette Funktionalität wurde im File main.c implementiert, ohne eine HAL zu verwenden. Die folgenden Aufgaben sollen aufzeigen wie eine HAL eingesetzt und implementiert werden kann. Durch den HAL soll der Code leserlich und erweiterbar gestaltet werden können, ohne dabei auf Performance verzichten zu müssen. Die Implementation ohne HAL stellt die Baseline für die Implementation mit HAL dar.

1. Importieren Sie das Projekte NoHAL (./Vorgabe/NoHAL) ins Code Composer Studio und studieren Sie den Code im File main.c.
2. Definieren Sie die fehlenden GPIO A Registeradressen. Informationen über die Register finden Sie im "Technical Reference Manual". Das Memory Map und die Registeradressen finden Sie im Datenblatt des TMS320F2806x Microcontrollers. In "Table 6-70. GPIO Registers" sind die Registeradressen für die GPIO Peripherie aufgelistet.
3. Verifizieren Sie die Funktionalität der Applik

11.2.1 Lösung

1. just do it

```

2 /*
3  * main.c
4  *
5  * Created on: April 15, 2019
6  * Author: Gian Danuser
7  */
8 #include <stdint.h>
9

```

```

10
11 #define EALLOW __asm(" EALLOW")
12 #define EDIS __asm(" EDIS")
13
14 // portA
15 #define GP_A_CTRL (*((volatile uint32_t *)((uint32_t)0
16     ↳ x00006F80)))
17 #define GP_A_SEL1 (*((volatile uint32_t *)((uint32_t)0

```

```

18     ↪ x00006F82)))
19 #define GP_A_SEL2 (*((volatile uint32_t *)((uint32_t)0
    ↪ x00006F84)))
20 #define GP_A_MUX1 (*((volatile uint32_t *)((uint32_t)0
    ↪ x00006F86)))
21 #define GP_A_MUX2 (*((volatile uint32_t *)((uint32_t)0
    ↪ x00006F88)))
22 #define GP_A_DIR (*((volatile uint32_t *)((uint16_t)0
    ↪ x00006F8A)))
23 #define GP_A_PUD (*((volatile uint32_t *)((uint32_t)0
    ↪ x00006F8C)))
24 #define GP_A_DATA (*((volatile uint32_t *)((uint32_t)0
    ↪ x00006FC0)))
25 #define GP_A_SET (*((volatile uint32_t *)((uint32_t)0
    ↪ x00006FC2)))
26 #define GP_A_CLR (*((volatile uint32_t *)((uint32_t)0
    ↪ x00006FC4)))
27 #define GP_A_TOG (*((volatile uint32_t *)((uint32_t)0
    ↪ x00006FC6)))
28
29 // bit manipulation
30 #define SET_BIT(reg, bit) ((reg) |= (((uint32_t)1u) << ((
    ↪ uint32_t)(bit))))
31 #define CLR_BIT(reg, bit) ((reg) &= ~(((uint32_t)1u) << ((
    ↪ uint32_t)(bit))))
32 #define GET_BIT(reg, bit) (((reg) & (((uint32_t)1u) << ((
    ↪ uint32_t)(bit)))) >> ((uint32_t)(bit)))
33
34 #define SET_BITS(reg, bits) ((reg) |= (bits))
35 #define CLR_BITS(reg, bits) ((reg) &= ~(bits))
36 #define ARE_BITS_SET(reg, bits) (((reg) & (bits)) == (bits))
37
38 // pin configuration
39 #define LED1_PIN 6u
40 #define LED2_PIN 2u
41 #define SWITCH1_PIN 26u
42 #define SWITCH2_PIN 30u
43
44 int main(void)
45 {
46
47
48     EALLOW; // allow register changes
49
50     // init GPIOs
51     GP_A_CTRL = (uint32_t)0xFFFFFFFF; // control register
52     GP_A_SEL1 = (uint32_t)0; // Sync for Pin 0 - Pin 15
53     GP_A_SEL2 = (uint32_t)0; // Sync for Pin 16 - Pin 31
54     GP_A_MUX1 = (uint32_t)0; // GPIO for Pin 0 - Pin 15
55     GP_A_MUX2 = (uint32_t)0; // GPIO for Pin 16 - Pin 31
56     GP_A_DIR = (uint32_t)0; // all GPIO's as input
57     GP_A_PUD = (uint32_t)0xFFFFFFFF; // disable GPIO pullup
    ↪ for all GPIOs
58
59     // LED 1 - statusIndicator
60     SET_BIT(GP_A_DIR, LED1_PIN); // configure direction output
61
62     // LED 2 - errorIndicator
63     SET_BIT(GP_A_DIR, LED2_PIN); // configure direction output
64
65     // SWITCH 1 - statusSwitch
66     CLR_BIT(GP_A_DIR, SWITCH1_PIN); // configure direction
    ↪ input
67
68     // SWITCH 2 - errorSwitch
69     CLR_BIT(GP_A_DIR, SWITCH2_PIN); // configure direction
    ↪ input
70
71     EDIS; // disallow register changes
72
73     while(1)
74     {
75         if(!ARE_BITS_SET(GP_A_DATA, 1u << SWITCH1_PIN))
76             SET_BITS(GP_A_SET, 1u << LED1_PIN);
77         else
78             SET_BITS(GP_A_CLR, 1u << LED1_PIN);
79
80         if(!ARE_BITS_SET(GP_A_DATA, 1u << SWITCH2_PIN))
81             SET_BITS(GP_A_SET, 1u << LED2_PIN);
82         else
83             SET_BITS(GP_A_CLR, 1u << LED2_PIN);
84     }
85
86     return 0;
87 }

```

11.3 Aufgabe 3: Hardware Abstraction Layer in C

Um die Applikation leserlich und erweiterbar zu gestalten, wird der Microcontroller in einer Chip Support Library (CSL) und das PCB des EswRobot in einer Board Support Library (BSL) abstrahiert. Diese beiden Libraries bilden den Hardware Abstraction Layer (HAL). Diese Abstraktion wurde für den EswRobot im Projekt CHAL in C vorgenommen.

1. Importieren Sie das Projekte CHAL (./Vorgabe/CHAL) ins Code Composer Studio und studieren Sie die CSL und die BSL.
2. Implementieren Sie die gleiche Funktionalität wie im Projekt NoHAL mit Hilfe des vorgegebenen HALs in main.c.
3. Verifizieren Sie, ob Ihr Testprogramm einwandfrei funktioniert.

11.3.1 Lösung

1. just do it

```

2. /*
3   * main.c
4   *
5   * Created on: April 15, 2019
6   * Author: Gian Danuser
7   */
8 #include <bsl/include/led.h>
9 #include <bsl/include/switch.h>
10
11 int main(void)
12 {
13     bsl_Led statusIndicator;
14     bsl_ledInit(&statusIndicator, bsl_led1);
15
16     bsl_Led errorIndicator;
17     bsl_ledInit(&errorIndicator, bsl_led2);
18
19     bsl_Switch statusSwitch;
20     bsl_switchInit(&statusSwitch, bsl_switch1);
21
22     bsl_Switch errorSwitch;
23     bsl_switchInit(&errorSwitch, bsl_switch2);
24
25     while(1)
26     {
27         if(bsl_switchPressed(&statusSwitch))
28             bsl_ledOn(&statusIndicator);
29         else
30             bsl_ledOff(&statusIndicator);
31
32         if(bsl_switchPressed(&errorSwitch))

```

```

33     bsl_ledOn(&errorIndicator);
34     else
35         bsl_ledOff(&errorIndicator);
36 }

```

```

37
38     return 0;
39 }

```

```

1  /*
2  * led.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #ifndef BSL_LED_H_
9  #define BSL_LED_H_
10
11 #include <csl/include/pin.h>
12
13 /**
14  * @brief The EswRobot LED ID.
15  */
16 typedef enum
17 {
18     bsl_led1,
19     bsl_led2
20 } bsl_LedId;
21
22 /**
23  * @brief The EswRobot LED data.
24  */
25 typedef struct
26 {
27     csl_Pin pin; ///< GPIO pin
28 } bsl_Led;
29
30 /**
31  * @brief Initializes the EswRobot LED.
32  *
33  * @param led LED data
34  * @param id LED ID
35  *
36  * @pre @p led pointer must be valid, validity is not

```

```

37     ↪ checked!
38 */
39 static inline void bsl_ledInit(bsl_Led* led, bsl_LedId id)
40 {
41     csl_pinInit(&(led->pin), (bsl_led1 == id) ? csl_pin6 :
42         ↪ csl_pin2, csl_pinFun1);
43     csl_pinSetDirection(&(led->pin), csl_pinOut);
44 }
45
46 /**
47  * @brief Turns off the EswRobot LED.
48  *
49  * @param led LED data
50  *
51  * @pre @p led pointer must be valid, validity is not
52     ↪ checked!
53 */
54 static inline void bsl_ledOff(bsl_Led* led)
55 {
56     csl_pinClear(&(led->pin));
57 }
58
59 /**
60  * @brief Turns on the EswRobot LED.
61  *
62  * @param led LED data
63  *
64  * @pre @p led pointer must be valid, validity is not
65     ↪ checked!
66 */
67 static inline void bsl_ledOn(bsl_Led* led)
68 {
69     csl_pinSet(&(led->pin));
70 }
71
72 #endif /* BSL_LED_H_ */

```

```

1  /*
2  * switch.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #ifndef BSL_SWITCH_H_
9  #define BSL_SWITCH_H_
10
11 #include <csl/include/pin.h>
12
13 /**
14  * @brief The EswRobot switch ID.
15  */
16 typedef enum
17 {
18     bsl_switch1,
19     bsl_switch2
20 } bsl_SwitchId;
21
22 /**
23  * @brief The EswRobot switch data.
24  */
25 typedef struct
26 {
27     csl_Pin pin;
28 } bsl_Switch;
29
30 /**

```

```

31  * @brief Initializes the EswRobot switch.
32  *
33  * @param sw switch data
34  * @param id switch ID
35  *
36  * @pre @p sw pointer must be valid, validity is not checked
37     ↪ !
38 */
39 static inline void bsl_switchInit(bsl_Switch* sw,
40     ↪ bsl_SwitchId id)
41 {
42     csl_pinInit(&(sw->pin), (bsl_switch1 == id) ? csl_pin26 :
43         ↪ csl_pin30, csl_pinFun1);
44     csl_pinSetDirection(&(sw->pin), csl_pinIn);
45 }
46
47 /**
48  * @brief Returns if the EswRobot switch is pressed.
49  *
50  * @param sw switch data
51  *
52  * @pre @p sw pointer must be valid, validity is not checked
53     ↪ !
54 */
55 static inline bool bsl_switchPressed(const bsl_Switch* sw)
56 {
57     return !csl_pinGet(&(sw->pin));
58 }
59
60 #endif /* BSL_SWITCH_H_ */

```

```

1  /*
2  * bits.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7

```

```

8  #ifndef CSL_BITS_H_
9  #define CSL_BITS_H_
10
11 /**
12  * @brief Bits
13  */
14 enum

```



```

15 {
16     csl_bit0 = 0x00000001, csl_bit1 = 0x00000002, csl_bit2
17     ↪ = 0x00000004, csl_bit3 = 0x00000008,
18     csl_bit4 = 0x00000010, csl_bit5 = 0x00000020, csl_bit6
19     ↪ = 0x00000040, csl_bit7 = 0x00000080,
20     csl_bit8 = 0x00000100, csl_bit9 = 0x00000200, csl_bit10
21     ↪ = 0x00000400, csl_bit11 = 0x00000800,
22     csl_bit12 = 0x00001000, csl_bit13 = 0x00002000, csl_bit14
23     ↪ = 0x00004000, csl_bit15 = 0x00008000,
24     csl_bit16 = 0x00010000, csl_bit17 = 0x00020000, csl_bit18
25     ↪ = 0x00040000, csl_bit19 = 0x00080000,
26     csl_bit20 = 0x00100000, csl_bit21 = 0x00200000, csl_bit22
27     ↪ = 0x00400000, csl_bit23 = 0x00800000,
28     csl_bit24 = 0x01000000, csl_bit25 = 0x02000000, csl_bit26
29     ↪ = 0x04000000, csl_bit27 = 0x08000000,
30     csl_bit28 = 0x10000000, csl_bit29 = 0x20000000, csl_bit30
31     ↪ = 0x40000000, csl_bit31 = 0x80000000
32 };
33 #endif /* CSL_BITS_H_ */

```

```

1  /*
2  * hwReg.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #ifndef CSL_HWREG_H_
9  #define CSL_HWREG_H_
10
11  #include <stdint.h>
12  #include <stdbool.h>
13
14  #include "../bits.h"
15
16  /**
17   * @brief Sets the specified bits in the 32 bit register.
18   *
19   * @param reg register
20   * @param bits bits to set
21   *
22   * @pre @p reg pointer must be valid, validity is not
23   * ↪ checked!
24   */
25  static inline void csl_hwReg32SetBits(volatile uint32_t* reg
26   ↪ , uint32_t bits)
27  {
28     *reg |= bits;
29  }
30
31  /**
32   * @brief Clears the specified bits in the 32 bit register.
33   *
34   * @param reg register
35   * @param bits bits to clear
36   *
37   * @pre @p reg pointer must be valid, validity is not
38   * ↪ checked!
39   */
40  static inline void csl_hwReg32ClearBits(volatile uint32_t*
41   ↪ reg, uint32_t bits)
42  {
43     *reg &= ~bits;
44  }
45
46  /**
47   * @brief Returns if the bits are set in the 32 bit register
48   * ↪ .
49   *
50   * @param reg register
51   * @param bits bits to check
52   * @return true if the bits are set and false otherwise
53   *
54   * @pre @p reg pointer must be valid, validity is not
55   * ↪ checked!
56   */
57  static inline bool csl_hwReg32AreBitsSet(const volatile
58   ↪ uint32_t* reg, uint32_t bits)
59  {
60     return (*reg & bits) == bits;
61  }
62
63  /**
64   * @brief Sets the value in the 32 bit register.
65   *
66   * @param reg register
67   * @param value value to set
68   *
69   * @pre @p reg pointer must be valid, validity is not
70   * ↪ checked!
71   */
72  static inline void csl_hwReg32SetValue(volatile uint32_t*
73   ↪ reg, uint32_t value)
74  {
75     *reg = value;
76  }
77
78  /**
79   * @brief Sets the 2 bit sub-value in the 32 bit register.
80   *
81   * @param reg register
82   * @param value sub-value to set
83   * @param shift shift for the sub-value
84   *
85   * @pre @p reg pointer must be valid, validity is not
86   * ↪ checked!
87   */
88  static inline void csl_hwReg32Set2BitSubValue(volatile
89   ↪ uint32_t* reg, uint32_t value, unsigned int shift)
90  {
91     csl_hwReg32ClearBits(reg, (uint32_t)((1 << 2) - 1) <<
92     ↪ shift);
93     csl_hwReg32SetBits(reg, value << shift);
94  }
95
96  /**
97   * @brief Sets the specified bits in the 16 bit register.
98   *
99   * @param reg register
100   * @param bits bits to set
101   *
102   * @pre @p reg pointer must be valid, validity is not
103   * ↪ checked!
104   */
105  static inline void csl_hwReg16SetBits(volatile uint16_t* reg
106   ↪ , uint16_t bits)
107  {
108     *reg |= bits;
109  }
110
111  /**
112   * @brief Clears the specified bits in the 16 bit register.
113   *
114   * @param reg register
115   * @param bits bits to clear
116   *
117   * @pre @p reg pointer must be valid, validity is not
118   * ↪ checked!
119   */
120  static inline void csl_hwReg16ClearBits(volatile uint16_t*
121   ↪ reg, uint16_t bits)
122  {
123     *reg &= ~bits;
124  }
125
126  /**
127   * @brief Returns if the bits are set in the 16 bit register
128   * ↪ .
129   *
130   * @param reg register
131   * @param bits bits to check
132   * @return true if the bits are set and false otherwise
133   *
134   * @pre @p reg pointer must be valid, validity is not
135   * ↪ checked!
136   */
137  static inline bool csl_hwReg16AreBitsSet(const volatile
138   ↪ uint16_t* reg, uint16_t bits)
139  {
140     return (*reg & bits) == bits;
141  }
142
143  /**
144   * @brief Sets the value in the 16 bit register.
145   *
146   * @param reg register
147   * @param value value to set
148   *
149   * @pre @p reg pointer must be valid, validity is not
150   * ↪ checked!
151   */
152  static inline void csl_hwReg16SetValue(volatile uint16_t*
153   ↪ reg, uint16_t value)
154  {
155     *reg = value;
156  }

```

```

133     ↪ reg, uint16_t value)
134 {
135     *reg = value;
136 }
137 /**
138  * @brief Sets the 2 bit sub-value in the 16 bit register.
139  *
140  * @param reg register
141  * @param value sub-value to set
142  * @param shift shift for the sub-value
143  */
144
145 * @pre @p reg pointer must be valid, validity is not
146     ↪ checked!
147
148 */
149 static inline void csl_hwReg16Set2BitSubValue(volatile
150     ↪ uint16_t* reg, uint16_t value, unsigned int shift)
151 {
152     csl_hwReg16ClearBits(reg, (uint16_t)((1 << 2) - 1) <<
153     ↪ shift);
154     csl_hwReg16SetBits(reg, value << shift);
155 }
156 #endif /* CSL_HWREG_H_ */

```

```

1  /*
2  * pin.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8 #ifndef CSL_PIN_H_
9 #define CSL_PIN_H_
10
11 #include "../bits.h"
12 #include "../hwReg.h"
13 #include "../port.h"
14
15 /**
16  * @brief The C2000 pin ID.
17  */
18 typedef enum
19 {
20     csl_pin0 = 0, // must be 0
21     csl_pin1,
22     csl_pin2,
23     csl_pin3,
24     csl_pin4,
25     csl_pin5,
26     csl_pin6,
27     csl_pin7,
28     csl_pin8,
29     csl_pin9,
30     csl_pin10,
31     csl_pin11,
32     csl_pin12,
33     csl_pin13,
34     csl_pin14,
35     csl_pin15,
36     csl_pin16,
37     csl_pin17,
38     csl_pin18,
39     csl_pin19,
40     csl_pin20,
41     csl_pin21,
42     csl_pin22,
43     csl_pin23,
44     csl_pin24,
45     csl_pin25,
46     csl_pin26,
47     csl_pin27,
48     csl_pin28,
49     csl_pin29,
50     csl_pin30,
51     csl_pin31,
52     csl_pin32,
53     csl_pin33,
54     csl_pin34,
55     csl_pin35,
56     csl_pin36,
57     csl_pin37,
58     csl_pin38,
59     csl_pin39,
60     csl_pin40,
61     csl_pin41,
62     csl_pin42,
63     csl_pin43,
64     csl_pin44,
65     csl_pin45,
66     csl_pin46,
67     csl_pin47,
68     csl_pin48,
69     csl_pin49,
70     csl_pin50,
71     csl_pin51,
72     csl_pin52,
73     csl_pin53,
74     csl_pin54,
75     csl_pin55,
76     csl_pin56,
77     csl_pin57,
78     csl_pin58,
79     csl_pin59,
80     csl_pin60,
81     csl_pin61,
82     csl_pin62,
83     csl_pin63
84 } csl_PinId;
85
86 /**
87  * @brief The pin data.
88  */
89 typedef struct
90 {
91     csl_Port port;
92     uint32_t bit;
93 } csl_Pin;
94
95 /**
96  * @brief The pin direction.
97  */
98 typedef enum
99 {
100     csl_pinIn,
101     csl_pinOut
102 } csl_PinDirection;
103
104 /**
105  * @brief The pin mux.
106  */
107 typedef enum
108 {
109     csl_pinFun1 = 0, ///< GPIO
110     csl_pinFun2 = csl_bit0,
111     csl_pinFun3 = csl_bit1,
112     csl_pinFun4 = csl_bit1 | csl_bit0
113 } csl_PinMux;
114
115 /**
116  * @brief Initializes the C2000 pin to the default
117     ↪ configuration.
118  *
119  * @param pin pin data
120  * @param id pin ID
121  * @param mux pin mux
122  *
123  * @pre @p pin pointer must be valid, validity is not
124     ↪ checked!
125  */
126 void csl_pinInit(csl_Pin* pin, csl_PinId id, csl_PinMux mux)
127     ↪ ;
128
129 /**
130  * @brief Sets the direction of the C2000 pin.
131  *
132  * @param pin pin data
133  * @param dir direction
134  *
135  * @pre @p pin pointer must be valid, validity is not
136     ↪ checked!
137  */
138 void csl_pinSetDirection(csl_Pin* pin, csl_PinDirection dir)
139     ↪ ;
140
141 /**
142  * @brief Sets the the C2000 pin.
143  *
144  * @param pin pin data

```

```

140  *
141  * @pre @p pin pointer must be valid, validity is not
142  *      ↪ checked!
143  */
144  static inline void csl_pinSet(csl_Pin* pin)
145  {
146      csl_hwReg32SetBits(&(pin->port.data->set), pin->bit);
147  }
148  /**
149   * @brief Clears the C2000 pin.
150   *
151   * @param pin pin data
152   *
153   * @pre @p pin pointer must be valid, validity is not
154   *      ↪ checked!
155  */
156  static inline void csl_pinClear(csl_Pin* pin)
157  {
158      csl_hwReg32SetBits(&(pin->port.data->clear), pin->bit);
159  }
160  /**
161   * @brief Toggles the C2000 pin.
162   *
163   * @param pin pin data

```

```

164  *
165  * @pre @p pin pointer must be valid, validity is not
166  *      ↪ checked!
167  */
168  static inline void csl_pinToggle(csl_Pin* pin)
169  {
170      csl_hwReg32SetBits(&(pin->port.data->toggle), pin->bit);
171  }
172  /**
173   * @brief Returns the state of the C2000 pin.
174   *
175   * @param pin pin data
176   * @return pin state
177   *
178   * @pre @p pin pointer must be valid, validity is not
179   *      ↪ checked!
180  */
181  static inline bool csl_pinGet(const csl_Pin* pin)
182  {
183      return csl_hwReg32AreBitsSet(&(pin->port.data->dat), pin->
184                                  ↪ bit);
185  }
186  #endif /* CSL_PIN_H_ */

```

```

1  /*
2  * port.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #ifndef CSL_PORT_H_
9  #define CSL_PORT_H_
10
11  #include <stdint.h>
12  #include <stdbool.h>
13
14  /**
15   * @brief The C2000 port control register.
16   */
17  typedef volatile struct
18  {
19      uint32_t ctrl;
20      uint32_t sel[2];
21      uint32_t mux[2];
22      uint32_t dir;
23      uint32_t pud;
24  } csl_PortCtrlRegs;
25
26  /**
27   * @brief The C2000 port data register.
28   */
29  typedef volatile struct
30  {
31      const uint32_t dat;
32      uint32_t set;
33      uint32_t clear;
34      uint32_t toggle;
35  } csl_PortDataRegs;
36

```

```

37  /**
38   * @brief The port ID.
39   */
40  typedef enum
41  {
42      csl_portA,
43      csl_portB,
44  } csl_PortId;
45
46  /**
47   * @brief The port data.
48   */
49  typedef struct
50  {
51      csl_PortCtrlRegs* ctrl;
52      csl_PortDataRegs* data;
53  } csl_Port;
54
55  /**
56   * @brief Initializes the C2000 port to the preset sample
57   *      ↪ period.
58   *
59   * @param port port data
60   * @param id port ID
61   * @param initSamplePeriod initializes the sample period if
62   *      ↪ set
63   *
64   * @note This function is called automatically by \ref
65   *      ↪ csl_pinInit.
66   *
67   * @pre @p port pointer must be valid, validity is not
68   *      ↪ checked!
69  */
70  void csl_portInit(csl_Port* port, csl_PortId id, bool
71                  ↪ initSamplePeriod);
72
73  #endif /* CSL_PORT_H_ */

```

```

1  /*
2  * pin.c
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #include "../include/pin.h"
9
10 #include "../protHwRegAccess.h"
11
12 enum
13 {
14     numPorts = 2, ///< number of C2000 ports
15     numPinsPerPort = 32, ///< number of pins per C2000 port
16     muxBitWidth = 2, ///< bit width per mux setting
17     syncBitWidth = 2, ///< bit width per sync setting
18     syncToSysClk = 0 ///< sync to system clock
19 };
20
21 static bool initDonePort[numPorts] = {};

```

```

22
23 void csl_pinInit(csl_Pin* pin, csl_PinId id, csl_PinMux mux)
24 {
25     // The pin and bit number is the same for portA.
26     // In case of portB, the pin number has to be adjusted
27     // by the number of pins per port to get the bit number.
28     if(id < numPinsPerPort)
29     {
30         // Ensures that port A is initialized only once.
31         csl_portInit(&(pin->port), csl_portA, !initDonePort[0]);
32         initDonePort[0] = true;
33         pin->bit = ((uint32_t)1) << id;
34     }
35     else
36     {
37         // Ensures that port B is initialized only once.
38         csl_portInit(&(pin->port), csl_portB, !initDonePort[1]);
39         initDonePort[1] = true;
40         pin->bit = ((uint32_t)1) << (id - numPinsPerPort);
41     }
42 }

```

```

43     protHwRegAccessAllow();
44
45     unsigned int bitNr = (id < numPinsPerPort) ? id : (id -
46         ↪ numPinsPerPort);
47
48     // sync
49     bool highReg = bitNr >= (numPinsPerPort/syncBitWidth);
50     unsigned int shift = (highReg ? bitNr - (numPinsPerPort/
51         ↪ syncBitWidth) : bitNr) * syncBitWidth;
52     cs1_hwReg32Set2BitSubValue(&(pin->port.ctrl->sel[highReg])
53         ↪ , syncToSysClk, shift);
54
55     // mux
56     highReg = bitNr >= (numPinsPerPort/muxBitWidth);
57     shift = (highReg ? bitNr - (numPinsPerPort/muxBitWidth) :
58         ↪ bitNr) * muxBitWidth;
59     cs1_hwReg32Set2BitSubValue(&(pin->port.ctrl->mux[highReg])
60         ↪ , mux, shift);
61
62     // disable pull-up
63     cs1_hwReg32SetBits(&(pin->port.ctrl->pud), pin->bit);
64
65     protHwRegAccessDisallow();
66
67     // default set to in
68     cs1_pinSetDirection(pin, cs1_pinIn);
69 }
70
71 void cs1_pinSetDirection(cs1_Pin* pin, cs1_PinDirection dir)
72 {
73     protHwRegAccessAllow();
74
75     if(dir == cs1_pinIn)
76     {
77         cs1_hwReg32ClearBits(&(pin->port.ctrl->dir), pin->bit);
78     }
79     else
80     {
81         cs1_hwReg32SetBits(&(pin->port.ctrl->dir), pin->bit);
82     }
83
84     protHwRegAccessDisallow();
85 }

```

```

1  /*
2  * port.c
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #include "../include/port.h"
9
10 #include "../include/hwReg.h"
11 #include "../protHwRegAccess.h"
12
13 enum
14 {
15     addrPortACtrlRegs = 0x00006F80, ///< base address of the
16         ↪ control register of port A
17     addrPortBCtrlRegs = 0x00006F90, ///< base address of the
18         ↪ control register of port B
19
20     addrPortADDataRegs = 0x00006FC0, ///< base address of the
21         ↪ data register of port A
22     addrPortBDDataRegs = 0x00006FC8, ///< base address of the
23         ↪ data register of port B
24 };
25
26 void cs1_portInit(cs1_Port* port, cs1_PortId id, bool
27     ↪ initSamplePeriod)
28 {
29     if(cs1_portA == id)
30     {
31         port->ctrl = (cs1_PortCtrlRegs*)addrPortACtrlRegs;
32         port->data = (cs1_PortDataRegs*)addrPortADDataRegs;
33     }
34     else
35     {
36         port->ctrl = (cs1_PortCtrlRegs*)addrPortBCtrlRegs;
37         port->data = (cs1_PortDataRegs*)addrPortBDDataRegs;
38     }
39
40     if(initSamplePeriod)
41     {
42         protHwRegAccessAllow();
43
44         // Sets the sample period for the entire port to 510 *
45         ↪ T_sysclk.
46         // This is done, because the sample period cannot be set
47         ↪ for each GIO pin separately.
48         // More information can be found in the user manual of
49         ↪ the C2000 DSP.
50         if(!cs1_hwReg32AreBitsSet(&(port->ctrl->ctrl), ~((
51             ↪ uint32_t)0)))
52         {
53             cs1_hwReg32SetValue(&(port->ctrl->ctrl), ~((uint32_t)
54                 ↪ 0)); // sample period = 510 * T_sysclk
55         }
56
57         protHwRegAccessDisallow();
58     }
59 }

```

```

1  /*
2  * protHwRegAccess.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #ifndef CSL_PROTHWREGACCESS_H_
9  #define CSL_PROTHWREGACCESS_H_
10
11 /**
12 * @brief Allows write access to protected C2000 register.
13 */
14 static inline void protHwRegAccessAllow(void)
15 {
16     __asm(" EALLOW");
17 }
18
19 /**
20 * @brief Disallows write access to protected C2000 register
21     ↪ .
22 */
23 static inline void protHwRegAccessDisallow(void)
24 {
25     __asm(" EDIS");
26 }
27 #endif /* CSL_PROTHWREGACCESS_H_ */

```

11.4 Aufgabe 4: Hardware Abstraction Layer in C++

Im Projekt CHAL wurde die Applikation vom Projekt NoHAL mit einem HAL in C implementiert. In dieser Aufgabe wird dieselbe Funktionalität mit einem HAL in C++ implementiert.

3. Importieren Sie das Projekt CPPHAL (./Vorgabe/CPPHAL) ins Code Composer Studio und studieren Sie die CSL, die BSL und die main()-Funktion.
2. Die Schnittstelle für die BSL Klassen Led und Switch sind definiert, aber nicht implementiert. Implementieren Sie die beiden Klassen vollständig. Alle Funktionen müssen implizit inline implementiert werden.
3. Verifizieren Sie, ob das vorgegebene Testprogramm einwandfrei funktioniert.

11.4.1 Lösung

1. just do it

```

2.  /*
   2  * Led.h
   3  *
   4  * Created on: April 15, 2019
   5  * Author: Gian Danuser
   6  */
   7
   8  #ifndef BSL_LED_H_
   9  #define BSL_LED_H_
  10
  11  #include <cs1/include/Pin.h>
  12
  13  namespace bsl
  14  {
  15
  16  /**
  17   * @brief Abstracts EswRobot LEDs.
  18   */
  19  class Led
  20  {
  21  public:
  22      /**
  23       * @brief The LED ID.
  24       */
  25      enum Id
  26      {
  27          led1,
  28          led2
  29      };
  30
  31      /**
  32       * @brief Initializes the LED.
  33       *
  34
  35       * @param id LED ID
  36       */
  37  Led(Id id) :
  38      pin(led1 == id ? cs1::Pin::pin6 : cs1::Pin::pin2,
  39          cs1::Pin::out)
  40      {
  41      }
  42
  43  /**
  44   * @brief Turns off the LED.
  45   */
  46  void off()
  47  {
  48      pin.clear();
  49  }
  50
  51  /**
  52   * @brief Turns on the LED.
  53   */
  54  void on()
  55  {
  56      pin.set();
  57  }
  58
  59  private:
  60      cs1::Pin pin; ///< GPIO pin
  61  };
  62
  63  } // namespace bsl
  64
  65  #endif /* BSL_LED_H_ */
  66

```

```

1  /*
2  * Switch.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #ifndef BSL_SWITCH_H_
9  #define BSL_SWITCH_H_
10
11  #include <cs1/include/Pin.h>
12
13  namespace bsl
14  {
15
16  /**
17   * @brief Abstracts EswRobot switches.
18   */
19  class Switch
20  {
21  public:
22      /**
23       * @brief The switch ID.
24       */
25      enum Id
26      {
27          switch1,
28          switch2
29      };
30
31
32      /**
33       * @brief Initializes the switch.
34       *
35       * @param id switch ID
36       */
37  Switch(Id id) :
38      pin(switch1 == id ? cs1::Pin::pin26 : cs1::Pin::pin30,
39          cs1::Pin::in)
40      {
41      }
42
43  /**
44   * @brief Returns if the switch is pressed.
45   *
46   * @return true if the switch is pressed and false
47   *         ↪ otherwise
48   */
49  bool pressed() const
50  {
51      return !pin.get();
52  }
53
54  private:
55      cs1::Pin pin; ///< GPIO pin
56  };
57
58  } // namespace bsl
59
60  #endif /* BSL_SWITCH_H_ */

```

11.5 Aufgabe 5: Hardware Abstraction Layer in C++ mit Memory-mapped IO (MMIO)

Bei Microcontrollern werden die peripheren Module direkt in den Speicher eingebunden und können über Adressen angesteuert werden. Die Klassen Pin und Port aus der CSL abstrahieren die GPIO-Module des Microcontrollers und verwenden für die Registerzugriffe Instanzen der Klasse HwReg, welche direkt an die entsprechende Registeradresse platziert werden.

1. Importieren Sie das Projekt CPPHAL_advanced (./Vorgabe/CPPHAL_advanced) ins Code Composer Studio und studieren Sie die CSL, die BSL und die main()-Funktion.
2. Implementieren Sie die Klasse HwReg vollständig. Alle Funktionen müssen implizit inline implementiert werden.
3. Platzieren Sie die benötigten HwReg-Objekte in der Pin Klasse an der korrekten Stelle und implementieren Sie damit die Stubs. Verwenden Sie die bereits implementierte toggle Methode als Inspiration. Überlegen Sie sich, ob für die Platzierung placement new oder ein reinterpret_cast bevorzugt wird? Hinweis: Für placement new müssen Sie den Header <new> inkludieren.
4. Verifizieren Sie, ob das vorgegebene Testprogramm einwandfrei funktioniert.
5. Compilieren Sie ihren Code mit der Optimierungsstufe O3 und beantworten Sie die folgenden Fragen. Um die Optimierungsstufe festzulegen, müssen Sie in den Projekteinstellungen unter Build -> C2000 Compiler -> Optimization das Optimization level 3-Interprocedure Optimization in der Dropdownliste auswählen.
 - (a) Wie wird die Abfrage if(statusSwitch.pressed()) im main auf Zeile 22 umgesetzt?
 - (b) Wie wird die Anweisung statusIndicator.off(); im main auf Zeile 25 umgesetzt?

11.5.1 Lösung

1. just do it
2. siehe ./Loesung/CPPHAL_advanced/csl/include/HwReg.h

```
1  /*
2  * HwReg.h
3  *
4  * Created on: April 15, 2019
5  * Author: Gian Danuser
6  */
7
8  #ifndef CSL_HWREG_H_
9  #define CSL_HWREG_H_
10
11 extern "C"
12 {
13 #include <stdint.h> // C2000 does not define <cstdint>
14 }
15
16 #include "./bits.h"
17
18 namespace csl
19 {
20
21 /**
22  * @brief Abstracts C2000 hardware registers.
23  *
24  * @tparam RegType register unsigned int type must match the
25  *             ↪ integer width of the
26  *             used architecture
27  */
28 template<typename RegType>
29 class HwReg
30 {
31 public:
32     /**
33      * @brief Sets the specified bits in the register.
34      *
35      * @param bits bits to set
36      */
37     void setBits(const RegType& bits)
38     {
39         reg |= bits;
40     }
41
42     /**
43      * @brief Clears the specified bits in the register.
44      *
45      * @param bits bits to clear
46      */
47     void clearBits(const RegType& bits)
48     {
49         reg &= ~bits;
50     }
51
52     /**
53      * @brief Returns if the bits are set in the register.
54      *
55      * @param bits bits to check
56      * @return true if the bits are set and false otherwise
57      */
58     bool areBitsSet(const RegType& bits) const
59     {
60         return (reg & bits) == bits;
61     }
62
63     /**
64      * @brief Sets the value in the register.
65      *
66      * @param value value to set
67      */
68     void setValue(const RegType& value)
69     {
70         reg = value;
71     }
72
73     /**
74      * @brief Sets the sub-value in the register.
75      *
76      * @tparam bitWidth bit width of the sub-value
77      * @tparam shift shift for the sub-value
78      *
79      * @param value sub-value to set
```

```

80  */
81  template<unsigned int bitWidth, unsigned int shift>
82  void setSubValue(RegType value) // const RegType&
83      ↳ results in an linking error
84  {
85      clearBits(((static_cast<RegType>(1) << bitWidth) -
86                  ↳ static_cast<RegType>(1)) << shift);
87      setBits(value << shift);
88  }
89  /**
90   * @brief Sets the sub-value in the register.
91   *
92   * @tparam bitWidth bit width of the sub-value
93   * @param value sub-value to set
94   * @param shift shift for the sub-value
95   */
96  template<unsigned int bitWidth>
97  void setSubValue(const RegType& value, unsigned int
98                  ↳ shift)
99  {
100      clearBits(((static_cast<RegType>(1) << bitWidth) -
101                  ↳ static_cast<RegType>(1)) << shift);
102      setBits(value << shift);
103  }
104
105  private:
106  /**
107   * @brief The destructor.
108   */
109  ~HwReg(); // prevent destructing of directly mapped
110             ↳ register
111
112  /**
113   * @brief The copy-constructor.
114   */
115  HwReg(const HwReg&); // prevent copying of directly
116                       ↳ mapped register
117
118  /**
119   * @brief The assignment operator.
120   */
121  HwReg& operator=(const HwReg&); // prevent copying of
122                                  ↳ directly mapped register
123
124  volatile RegType reg; ///< register which is directly
125                          ↳ mapped to the register address
126
127 };
128
129 } // namespace csl
130
131 #endif /* CSL_HWREG_H_ */

```

3. In diesem Fall macht ein reinterpret_cast< HwRegister;uint32_t*> mehr Sinn, da keine Konstruktoraufwurf erwünscht ist. Zudem wird Placement new nicht auf null Instruktionen reduziert. Es wird ein Call und ein Return durchgeführt, obwohl der Placement new Operator nur die übergebene Adresse zurückgibt. Immerhin wird der Konstruktoraufwurf auf null Instruktionen reduziert. Siehe ./Loesung/CPPHAL_advanced/csl/include/Pin.h

```

1  /*
2   * Pin.h
3   *
4   * Created on: April 15, 2019
5   * Author: Gian Danuser
6   */
7
8  #ifndef CSL_PIN_H_
9  #define CSL_PIN_H_
10
11  extern "C"
12  {
13  #include <stdint.h> // C2000 does not define <cstdint>
14  }
15
16  #include "../bits.h"
17  #include "../ProtHwRegAccess.h"
18  #include "../HwReg.h"
19  #include "../Port.h"
20
21  namespace csl
22  {
23
24  namespace pin
25  {
26
27  /**
28   * @brief The pin ID.
29   */
30  enum Id
31  {
32      pin0 = 0, // must be 0
33      pin1,
34      pin2,
35      pin3,
36      pin4,
37      pin5,
38      pin6,
39      pin7,
40      pin8,
41      pin9,
42      pin10,
43      pin11,
44      pin12,
45      pin13,
46      pin14,
47      pin15,
48      pin16,
49      pin17,
50      pin18,
51      pin19,
52      pin20,
53      pin21,
54      pin22,
55      pin23,
56      pin24,
57      pin25,
58      pin26,
59      pin27,
60      pin28,
61      pin29,
62      pin30,
63      pin31,
64      pin32,
65      pin33,
66      pin34,
67      pin35,
68      pin36,
69      pin37,
70      pin38,
71      pin39,
72      pin40,
73      pin41,
74      pin42,
75      pin43,
76      pin44,
77      pin45,
78      pin46,
79      pin47,
80      pin48,
81      pin49,
82      pin50,
83      pin51,
84      pin52,
85      pin53,
86      pin54,
87      pin55,
88      pin56,
89      pin57,
90      pin58,
91      pin59,
92      pin60,
93      pin61,
94      pin62,
95      pin63
96  };
97
98  /**
99   * @brief The pin direction.
100   */
101  enum Direction
102  {
103      in,
104      out
105  };

```



```

107 /**
108  * @brief The pin mux.
109  */
110 enum Mux
111 {
112     fun1 = 0, ///< GIO
113     fun2 = bit0,
114     fun3 = bit1,
115     fun4 = bit1|bit0
116 };
117
118 /**
119  * @brief The pin sync selection.
120  */
121 enum Sync
122 {
123     syncToSysClk = 0 ///< sync to system clock
124 };
125
126 /**
127  * @brief The C2000 pin base class.
128  *
129  * @tparam id port ID
130  */
131 template<port::Id id>
132 class PinBase
133 {
134 public:
135     /**
136      * @brief Initializes the port if required.
137      *
138      * @param bitNr bit number
139      * @param dir pin direction
140      * @param mux pin mux
141      * @param isPullupEn true if pull-up enabled
142      * @param sync sync to specified clock
143      */
144     PinBase(unsigned int bitNr,
145             Direction dir,
146             Mux mux,
147             bool isPullupEn,
148             Sync sync);
149
150     /**
151      * @brief Sets the direction of the pin.
152      *
153      * @param bitNr bit number
154      * @param dir pin direction
155      */
156     void setDirection(unsigned int bitNr, Direction dir);
157
158 private:
159     enum
160     {
161         muxBitWidth = 2, ///< bit width per mux setting
162         syncBitWidth = 2, ///< bit width per sync setting
163         pudBitWidth = 1 ///< bit width per pud setting
164     };
165
166     static bool initDonePort; ///< port initialized flag
167 };
168
169 /**
170  * @brief Template meta programming to get the port ID
171  * ↳ derived from the pin ID.
172  *
173  * @tparam pinId pin ID
174  */
175 template<Id pinId>
176 struct GetPort
177 {
178     static const port::Id id = pinId < port::numPins ? port::a
179     ↳ : port::b;
180 };
181
182 /**
183  * @brief Abstracts C2000 pins.
184  *
185  * @tparam id pin ID
186  */
187 template<Id id>
188 class Pin : public PinBase<GetPort<id>::id>
189 {
190 public:
191     /**
192      * @brief Initializes the pin to the default
193      * ↳ configuration.
194      *
195      * @param dir pin direction (default: in)
196      * @param mux pin mux (default: GIO)
197
198      * @param isPullupEn true if pull-up enabled (default:
199      * ↳ enabled)
200      * @param sync sync to specified clock (default: sync to
201      * ↳ system clock)
202      */
203     Pin(Direction dir = in,
204          Mux mux = fun1,
205          bool isPullupEn = true,
206          Sync sync = syncToSysClk);
207
208     /**
209      * @brief Sets the direction of the pin.
210      *
211      * @param dir pin direction
212      */
213     void setDirection(Direction dir);
214
215     /**
216      * @brief Sets the pin.
217      */
218     void set();
219
220     /**
221      * @brief Clears the pin.
222      */
223     void clear();
224
225     /**
226      * @brief Toggles the pin.
227      */
228     void toggle();
229
230     /**
231      * @brief Returns the state of the pin.
232      *
233      * @return pin state
234      */
235     bool get() const;
236 };
237
238 //
239 // template meta programming
240 //
241 /**
242  * @brief Template meta programming to get the bit number
243  * ↳ derived from the pin ID.
244  *
245  * @tparam id pin ID
246  */
247 template<Id id>
248 struct GetBitNr
249 {
250     static const unsigned int nr = id < port::numPins ? id :
251     ↳ id - port::numPins;
252 };
253
254 //
255 // implementation
256 //
257 template<port::Id id>
258 bool PinBase<id>::initDonePort = false;
259
260 template<port::Id id>
261 PinBase<id>::PinBase(unsigned int bitNr,
262                     Direction dir,
263                     Mux mux,
264                     bool isPullupEn,
265                     Sync sync)
266 {
267     // Ensures that each port is initialized only once.
268     if(!initDonePort)
269     {
270         port::Port<id> port(true);
271         initDonePort = true;
272     }
273
274     ProtHwRegAccess::allow();
275
276     //configure sync
277     if(bitNr >= (port::numPins/syncBitWidth))
278         reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id
279         ↳ >::sel2)->setSubValue<syncBitWidth>(
280             sync, bitNr - port::numPins/syncBitWidth);
281     else
282         reinterpret_cast<HwReg<uint32_t>*>(port::GetAddrReg<id
283         ↳ >::sel1)->setSubValue<syncBitWidth>(
284             sync, bitNr);

```



```

282 //configure mux
283 if(bitNr >= (port::numPins/muxBitWidth))
284     reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<id>
    ↳ >::mux2)->setSubValue<syncBitWidth>(
285         mux, bitNr - port::numPins/muxBitWidth);
286 else
287     reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<id>
    ↳ >::mux1)->setSubValue<syncBitWidth>(
288         mux, bitNr);
289
290 //configure pull-up
291 reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<id>::
    ↳ pud)->setSubValue<pudBitWidth>(
292     isPullupEn, bitNr);
293
294 ProtHwRegAccess::disallow();
295
296 setDirection(bitNr, dir);
297 }
298
299 template<port::Id id>
300 void PinBase<id>::setDirection(unsigned int bitNr, Direction
    ↳ dir)
301 {
302     ProtHwRegAccess::allow();
303
304     if(dir == in)
305         reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<id>
    ↳ >::dir)->clearBits(
306             static_cast<uint32_t>(1) << bitNr);
307     else
308         reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<id>
    ↳ >::dir)->setBits(
309             static_cast<uint32_t>(1) << bitNr);
310
311     ProtHwRegAccess::disallow();
312 }
313
314 template<Id id>
315 Pin<id>::Pin(Direction dir,
316             Mux mux,
317             bool isPullupEn,
318             Sync sync) :
319     PinBase<GetPort<id>::id>(GetBitNr<id>::nr, dir, mux,
    ↳ isPullupEn, sync)
320 {
321 }
322
323
324 template<Id id>
325 void Pin<id>::setDirection(Direction dir)
326 {
327     PinBase<id>::setDirection(GetBitNr<id>::nr, dir);
328 }
329
330 template<Id id>
331 inline void Pin<id>::set()
332 {
333     // reinterpret cast is used instead of placement new,
334     // because C2000 compiler does not reduce the new operator
335     ↳ to zero instructions
336     reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<
    ↳ GetPort<id>::id>::set)->setBits(
337         static_cast<uint32_t>(1) << GetBitNr<id>::nr);
338 }
339
340 template<Id id>
341 inline void Pin<id>::clear()
342 {
343     // reinterpret cast is used instead of placement new,
344     // because C2000 compiler does not reduce the new operator
345     ↳ to zero instructions
346     reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<
    ↳ GetPort<id>::id>::clr)->setBits(
347         static_cast<uint32_t>(1) << GetBitNr<id>::nr);
348 }
349
350 template<Id id>
351 inline void Pin<id>::toggle()
352 {
353     // reinterpret cast is used instead of placement new,
354     // because C2000 compiler does not reduce the new operator
355     ↳ to zero instructions
356     reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<
    ↳ GetPort<id>::id>::toggle)->setBits(
357         static_cast<uint32_t>(1) << GetBitNr<id>::nr);
358 }
359
360 template<Id id>
361 inline bool Pin<id>::get() const
362 {
363     // reinterpret cast is used instead of placement new,
364     // because C2000 compiler does not reduce the new operator
365     ↳ to zero instructions
366     return reinterpret_cast<HwReg<uint32_t*>>(port::GetAddrReg<
    ↳ <GetPort<id>::id>::dat)->areBitsSet(
367         static_cast<uint32_t>(1) << GetBitNr<id>::nr);
368 }
369
370 } // namespace pin
371
372 } // namespace csl
373
374 #endif /* CSL_PIN_H_ */

```

4. Bei der Umsetzung mit Optimierungsstufe O3 wurden alle Funktionen von Pin inline, d.h. es wird direkt mit Bitmasken auf den Registern operiert, ohne Overhead zu verursachen.

(a) Die Adresse des Registers GPADAT wird ins Register XAR4 geladen. Bit 10 im High Byte (Bit 26) wird überprüft und je nach Testergebnis wird gesprungen. Ab Zeile 1354 in Debug/main.asm.

```

1  ...
2  MOVL      XAR4,#28608      ; [CPU_ARAU] |60|
3  TBIT      **XAR4[1],#10    ; [CPU_ALU] |60|
4  B         $C$L24,TC        ; [CPU_ALU] |60|
5  ...

```

(b) Die Adresse des Registers GPACLEAR wird ins Register XAR4 geladen. Der Inhalt des Registers wird dann direkt OR verknüpft mit 0x40 (Bit 6) um den LED 1 Pin auf 0 zu setzen. Ab Zeile 1363 in Debug/main.asm.

```

1  ...
2  $C$L24:
3  MOVL      XAR4,#28612 ; [CPU_ARAU] |38|
4  $C$L25:
5  OR        **XAR4[0],#64 ; [CPU_ALU] |38|
6  ...

```

12 Lab 12 Zufallszahlengeneratoren

12.1 Aufgabe 1: Wahrscheinlichkeitsverteilungen

Implementieren Sie die Klasse `RandIntGen`, die Ihnen unterschiedlich verteilte ganze Zahlen liefert. Als User Interface genügt die Console. In der Klasse sollen keinerlei Userinteraktionen (`cin`, `cout`, etc.) vorhanden sein. Die Klasse soll die folgenden Methoden anbieten:

```
getUniform(); // liefert eine gleichverteilte ganze Zahl
getNormal(); // liefert eine normalverteilte ganze Zahl
getWeibull(); // liefert eine Weibull-verteilte ganze Zahl
```

1. Definieren Sie die Schnittstelle. Überlegen Sie sich dabei, welche Methoden notwendig sind, und wie die Parameter und Returnwerte aussehen müssen, inkl. die korrekte Anwendung von `const`.
2. Implementieren Sie Stubs der Klasse.
3. Implementieren Sie ein Testprogramm, das die Häufigkeit der einzelnen Werte ermittelt und stellen Sie diese Resultate in einem Histogramm (intern oder extern z.B. mit Excel/Matlab/gnuplot) dar.
4. Führen Sie dieses Programm aus.
5. Implementieren Sie die Klasse (Methode um Methode).
6. Verwenden sie die Pseudo-random number generators von C++ und vergleichen Sie die Verteilungen mit den Verteilungen generiert mit der Klasse `RandIntGen`. Informationen über die Pseudo-random number generators finden sie hier: <https://en.cppreference.com/w/cpp/numeric/random>

Hinweise:

- Verwenden Sie die in den Vorlesungsunterlagen beschriebenen Formeln.
- Die Weibullverteilung besitzt zwei reelle Parameter α und β . Aus der Gleichverteilung erhält man eine Weibull-verteilte Zahl w wie folgt:
 1. i. Erzeuge eine gleichverteilte Zahl u im Bereich $[0, 1[$
 2. $w = \beta(-\log(1.0 - u))^{\frac{1}{\alpha}}$
- Seien Sie vorsichtig bei der Umrechnung von Gleitpunktwerten in ganzzahlige Werte. Überlegen Sie sich, wann `lround()` verwendet werden kann und wann `static_cast<int>`. Zur Erinnerung: bei der zweiten Methode wird ohne zu runden einfach der ganzzahlige Teil genommen.
- Bei gewissen Verteilungen müssen Sie `lround()` nehmen, bei anderen `static_cast<int>`.

12.1.1 Lösung

```
1 //===== 27
2 // Name      : Distributions.cpp 28
3 // Author     : Reto Bonderer    29
4 // Version    :                  30
5 // Copyright  : (c) HSR R. Bonderer 31
6 // Description: Tests various integer random distributions 32
7 //===== 33
8 34
9 #include <iostream> 35
10 #include <iomanip> 36
11 #include <random> 37
12 #include "RandIntGen.h" 38
13 39
14 using namespace std; 40
15 41
16 int main() 42
17 { 43
18     enum 44
19     { 45
20         minValue = 1, // minimal value for random numbers 46
21         maxValue = 20, // maximal value for random numbers 47
22         stdDev = 2, // normal distribution standard deviation 48
23         mean = 11, // normal distribution mean 49
24         wbAlpha = 2, // weibull alpha 50
25         wbBeta = 3, // weibull beta 51
26         iterations = 20000 // number of iterations 52
```

```
};
int histogram[maxValue - minValue + 1] = {0};
RandIntGen rg(time(0));
std::random_device rDev;
std::default_random_engine rgStd(rDev());
int r;

// uniform distribution
cout << "Uniform distribution" << endl;
for (int i = 0; i < iterations; ++i)
{
    r = rg.getUniform(minValue, maxValue);
    ++histogram[r - minValue];
}
for (int i = 0; i < maxValue - minValue + 1; ++i)
{
    cout << histogram[i] << " ";
}
cout << endl;

// uniform distribution using std library random generators
for (int i = 0; i < maxValue - minValue + 1; ++i)
{
    histogram[i] = 0;
}
```

```

53     cout << "Uniform distribution (std library)" << endl;
54     std::uniform_int_distribution<int> uniformDist(minValue,
55         ↪ maxValue);
56     for (int i = 0; i < iterations; ++i)
57     {
58         r = uniformDist(rgStd);
59         ++histogram[r - minValue];
60     }
61     for (int i = 0; i < maxValue - minValue + 1; ++i)
62     {
63         cout << histogram[i] << " ";
64     }
65     cout << endl;
66
67     // normal distribution
68     for (int i = 0; i < maxValue - minValue + 1; ++i)
69     {
70         histogram[i] = 0;
71     }
72     cout << "Normal distribution" << endl;
73     for (int i = 0; i < iterations; ++i)
74     {
75         r = rg.getNormal(mean, stdDev);
76         if (r >= 0 && r <= maxValue-minValue)    // r could be
77             ↪ beyond array limits
78         ++histogram[r];
79     }
80     for (int i = 0; i < maxValue - minValue + 1; ++i)
81     {
82         cout << histogram[i] << " ";
83     }
84     cout << endl;
85
86     // normal distribution using std library random generators
87     for (int i = 0; i < maxValue - minValue + 1; ++i)
88     {
89         histogram[i] = 0;
90     }
91     cout << "Normal distribution (std library)" << endl;
92     std::normal_distribution<double> normalDist(mean, stdDev);
93     for (int i = 0; i < iterations; ++i)
94     {
95         r = std::round(normalDist(rgStd));
96         if (r >= 0 && r <= maxValue-minValue)    // r could be
97             ↪ beyond array limits
98         ++histogram[r];
99     }
100     for (int i = 0; i < maxValue - minValue + 1; ++i)
101     {
102         cout << histogram[i] << " ";
103     }
104     cout << endl;
105
106     // Weibull distribution
107     for (int i = 0; i < maxValue - minValue + 1; ++i)
108     {
109         histogram[i] = 0;
110     }
111     cout << "Weibull distribution" << endl;
112     for (int i = 0; i < iterations; ++i)
113     {
114         r = rg.getWeibull(wbAlpha, wbBeta);
115         if (r >= 0 && r <= maxValue-minValue)    // r could be
116             ↪ beyond array limits
117         ++histogram[r];
118     }
119     for (int i = 0; i < maxValue - minValue + 1; ++i)
120     {
121         cout << histogram[i] << " ";
122     }
123     cout << endl;
124
125     // Weibull distribution using std library random generators
126     for (int i = 0; i < maxValue - minValue + 1; ++i)
127     {
128         histogram[i] = 0;
129     }
130     cout << "Weibull distribution (std library)" << endl;
131     std::weibull_distribution<double> weibullDist(wbAlpha,
132         ↪ wbBeta);
133     for (int i = 0; i < iterations; ++i)
134     {
135         r = std::round(weibullDist(rgStd));
136         if (r >= 0 && r <= maxValue-minValue)    // r could be
137             ↪ beyond array limits
138         ++histogram[r];
139     }
140     for (int i = 0; i < maxValue - minValue + 1; ++i)
141     {
142         cout << histogram[i] << " ";
143     }
144     cout << endl;
145
146     return 0;
147 }

```

```

1  /*
2  * Randgen.h
3  *
4  * Created on: 19.03.2015
5  * Author: rbondere
6  */
7
8  #ifndef RANDINTGEN_H_
9  #define RANDINTGEN_H_
10
11  class RandIntGen
12  {
13  public:
14     RandIntGen(int seed = 0);
15
16     // returns a uniformly distributed random int value
17
18     int getUniform(int low,           // lowest random value
19         int high) const;             // highest random
20                                     ↪ value
21
22     // returns a normally distributed random int value
23     int getNormal(double mean,        // mean (expected
24         ↪ value)
25         double sdev) const;          // standard deviation
26
27     // returns a Weibull distributed random int value
28     int getWeibull(double alpha,
29         double beta) const;
30 };
31
32 #endif /* RANDINTGEN_H_ */

```

```

1  /*
2  * Randgen.cpp
3  *
4  * Created on: 19.03.2015
5  * Author: rbondere
6  */
7  #include <cstdlib>
8  #include <cmath>
9  #include "RandIntGen.h"
10
11  using namespace std;
12
13  RandIntGen::RandIntGen(int seed)
14  {
15     srand(seed);
16  }
17
18  int RandIntGen::getUniform(int low,
19     int high) const
20  {
21     return static_cast<int> (static_cast<double> (rand()) / (
22         ↪ RAND_MAX + 1.0)
23         * (high - low + 1)) + low;
24     // '(RAND_MAX + 1.0)' ist notwendig, damit 0 <= u < 1.0 (
25         ↪ ohne 1.0)
26     // '+ 1' ist notwendig, weil nach int konvertiert wird.
27     // Vor Konversion gibt es dadurch bei [1,20] Zahlen im
28         ↪ Bereich [0.0,20.0]
29     // Nach Konversion werden nur die ganzzahligen Anteile
30         ↪ verwendet
31     // Das hinterste "+ low" muss ausserhalb des Typecasts sein,
32         ↪ andernfalls
33     // ergibt es ein Ueberhoehung bei 0, falls low < 0 (alle
34         ↪ Zahlen in

```

```

29 // [-0.9999, +0.9999] wuerden zu 0, d.h. doppelt so viele
30 // ↳ wie erlaubt)
31 }
32 int RandIntGen::getNormal(double mean,
33                           double sdev) const
34 {
35     double v1;
36     double v2;
37     double s;
38     do
39     {
40         v1 = 2 * static_cast<double> (rand()) / RAND_MAX - 1.0;
41         v2 = 2 * static_cast<double> (rand()) / RAND_MAX - 1.0;
42         s = v1 * v1 + v2 * v2;
43     } while (s >= 1.0 || s == 0.0);
44     double temp = sqrt(-2.0 * log(s) / s);

```

```

45     return lround(v1 * temp * sdev + mean);
46     // bei truncation (ohne lround(), sondern mit (int)) wird
47     // ↳ der Zufallswert
48     // bei 0 doppelt gewertet!!
49 }
50 int RandIntGen::getWeibull(double alpha,
51                             double beta) const
52 {
53     double u;
54     double w;
55     u = static_cast<double> (rand()) / (RAND_MAX + 1.0); // 0.0
56     // ↳ <= u < 1.0
57     w = beta * pow(-log(1.0 - u), 1 / alpha);
58     return static_cast<int> (w);

```

13 Lab 13 Security

13.1 Aufgabe 1: Hashing

Im Verzeichnis ./Vorgabe/Hash finden Sie drei JPG-Files, die alle dieselbe Grösse und denselben Timestamp haben. Zwei Dateien sind identisch, eine ist unterschiedlich. Sie sollen herausfinden, welche Datei unterschiedlich ist. Sie könnten diese Aufgabe mit dem Linux-Befehl diff lösen. Sie könnten die Dateien ebenfalls Byte für Byte binär vergleichen. Sie sollen nun aber diese Aufgabe mit Hilfe einer Hashfunktion lösen. Wenn die Dateien identisch sind, haben Sie denselben Fingerprint.

13.1.1 Lösung

Im Windows Explorer kann unter dem Kontextmenu CRC SHA ein Hash Fingerprint einer Datei erstellt werden, z.B. SHA-256. Unter Linux besteht für SHA-256 der Befehl sha256sum. Die SHA-256 Hashes der einzelnen Dateien sind:

img3.jpg: SHA256: EF5CDDDD830BBCBF544FFEB18E3B28B4D19D65A5AE16F4618D742683940F9E64D

img4.jpg: SHA256: EF5CDDDD830BBCBF544FFEB18E3B28B4D19D65A5AE16F4618D742683940F9E64D

img5.jpg: SHA256: 5E8EDB576C3CF31013ED0858E35C280BA5B667164DA2991C1AE2ECADF90975A2B

Aus den erhaltenen Hashes sieht man, dass die Dateien img3.jpg und img4.jpg identisch sind, img5.jpg aber abweicht. In img5.jpg wurde nur ein einziges Byte abgeändert, die Metainformation "Photoshop" wurde in "Photishop" abgeändert, das eigentliche Bild ist identisch. Man sieht eindrücklich, dass die Änderung eines einzigen Bytes einen völlig anderen Hash ergibt. Das ist unter anderem die Stärke einer guten Hashfunktion.

13.2 Aufgabe 1: Verschlüsselten Text entschlüsseln

Eine typische Verschlüsselungsmethode wandelt die Buchstaben eines Textfiles in ASCII und führt Buchstabe um Buchstabe eine XOR-Verknüpfung mit einem Wert aus einem geheimen Schlüssel durch. Die XORFunktion ist sehr schnell und sie ist symmetrisch, d.h.

```
1 //=====
2 // Name      : CipherFreq.cpp
3 // Author    : Reto Bonderer
4 // Version   :
5 // Copyright : (c) Reto Bonderer
6 // Description : Attacks based on the char frequencies
7 // you may use the following knowledge for your attack:
8 // (1) key is 3 lowercase letters (a-z)
9 // (2) key is repeated as long as needed and xored byte by
10 //     byte
11 // (3) the plaintext contains mixed cases, digits, punctuation
12 //     and spaces
13 // (4) the plaintext is in English, the 10 most common letters
14 //     in the English
15 //     language are, in order of decreasing frequency: e, t, a
16 //     , o, i, n, s, h, r, d, l.
17 //=====
18
19 #include <iostream>
20 #include <iomanip>
21 using namespace std;
22
23 static const char ciphertext[] =
24 // to long to show
25
26 struct CharFreq
27 {
28     char ch;
29     unsigned int count;
30 };
31
32 // counts frequencies of a ciphertext character for the (
33 // chrPos+1)th key character
34 void countFrequencies(const char* cText,          // ciphertext
35                      unsigned int cNumber,       // cText size
36                      unsigned int chrPos,        // position of
37                      // key char (0..2)
38                      CharFreq* f,               // array of
39                      // char counters
40                      unsigned int charNumber);  // number of
41                      // array elems in f
42
43 // returns the key character from the most frequent ciphertext
44 // character in f
45 char getKeyChar(const CharFreq* f,              // array of
46                unsigned int charNumber);       // number of
47                // array elems in f
48
49 // prints the first 'number' chars of the decrypted text
50 void printText(char first, char second, char third, const char
51               // * cText, unsigned int number);
52
53 int main(void)
54 {
55     char key[3];
56
57     cout << "Here is the key:" << endl;
58     for (unsigned int k = 0; k < 3; ++k) // keylength = 3
59     {
60         CharFreq chFreq[256] = {{0, 0}}; // stores the
61         // frequencies of the chars
62
63         // count freqs of (k+1)th key character
64         countFrequencies(ciphertext, sizeof(ciphertext), k, chFreq
65                          // , sizeof(chFreq)/sizeof(chFreq[0]));
66         key[k] = getKeyChar(chFreq, sizeof(chFreq)/sizeof(chFreq
67                          // [0]));
68         cout << key[k];
69     }
70
71     cout << endl << endl << "Here is the full text:" << endl;
72     printText(key[0], key[1], key[2], ciphertext, sizeof(
73               // ciphertext));
74
75     return 0;
76 }
77
78 void countFrequencies(const char* cText, unsigned int cNumber,
79                      unsigned int chrPos, CharFreq* f, unsigned int charNumber)
80 {
81     unsigned int top = 0; // index of next char to store
82     for (unsigned int i = 0; i < cNumber; i+=3) // 3: keylength
```

```

69 {
70     unsigned int j;
71     for (j = 0; j <= top; ++j)
72     {
73         if (cText[i+chrPos] == f[j].ch) // found entry
74         {
75             ++(f[j].count);
76             break;
77         }
78     }
79     if (j > top) // didn't find entry
80     {
81         f[top].ch = cText[i+chrPos];
82         f[top].count = 1;
83         ++top;
84     }
85 }
86
87 char getKeyChar(const CharFreq* f, unsigned int charNumber)
88 {
89     // in fact: the most frequent character is the space
90     ↪ character, 2nd is 'e' in English
91     unsigned int count = f[0].count;
92     unsigned int pos = 0;
93

```

```

94     for (unsigned int i = 1; i < charNumber; ++i)
95     {
96         if (f[i].count > count)
97         {
98             count = f[i].count;
99             pos = i;
100         }
101     }
102     return f[pos].ch ^ 0x20; // space is the most frequent
103 }
104
105 void printText(char first, char second, char third, const char
106     ↪ * cText, unsigned int number)
107 {
108     char* pch = const_cast<char*>(cText);
109     for (unsigned int i = 0; i < number/3; ++i)
110     {
111         char ch1 = first ^ *pch++;
112         char ch2 = second ^ *pch++;
113         char ch3 = third ^ *pch++;
114
115         cout << ch1 << ch2 << ch3;
116     }
117     cout << endl;

```

$\text{ciphertextByte} = \text{plaintextByte} \oplus \text{key}$ und

$\text{plaintextByte} = \text{ciphertextByte} \oplus \text{key}$

Je länger ein Passwort ist und je mehr unterschiedliche Zeichen es enthält, desto sicherer ist das Passwort. In diesem Beispiel nehmen wir ein Passwort, das nur aus 3 kleinen Buchstaben (a-z) besteht. Der Schlüssel besteht aus der fortlaufenden Aneinanderreihung dieses Passworts. In ./Vorgabe/Cipher finden Sie einen ciphertext, der mit einem Passwort bestehend aus 3 Kleinbuchstaben verschlüsselt wurde. Ihre Aufgabe besteht darin, das Passwort zu knacken und den Text zu entschlüsseln. Zur Lösung können Sie die folgenden Kenntnisse nutzen:

- Das Passwort besteht aus drei Kleinbuchstaben, der Schlüssel wird durch fortlaufendes Aneinanderreihen des Passworts generiert
- Die Verschlüsselung geschieht mittels XOR-Funktion
- Der unverschlüsselte englische Text besteht nur aus Gross- und Kleinbuchstaben, Ziffern, Leerzeichen und Satzzeichen

Als Angriffsvariante können Sie beispielsweise eine oder mehrere der folgenden Strategien anwenden:

- Trial and error
- Brute force (es gibt $26 * 26 * 26 = 17'576$ Möglichkeiten für den Schlüssel)
- Intuition, Raten, etc.
- Häufigkeitsanalyse der Buchstaben. Die 10 häufigsten Buchstaben der englischen Sprache in abnehmender Reihenfolge sind: e, t, a, o, i, n, s, h, r, d, l.
- Erraten einer Teillösung und fortlaufendes Erraten von zusätzlichen Buchstaben
- Weitere intelligente Methoden (beachten Sie, dass der Plaintext aus lesbaren Zeichen besteht)